بسم الله الرحمن الرحيم

جامعة السّودان المفتوحة

برنامج الحاسوب

نظم موزعة

رمز المقرر ورقمه: تقن 4013

إعداد المادة العلمية: د. عبد الرحمن أحمد محمد عثمان

التحكيم العلمي: د. أحمد صلاح الدين عبد الله

التصميم تعليمي : أ. ايهاب عبد الحي محد

التدقيق اللغوي : أ. الهدي عبد الله محمد محمد أحمد

التصميم الفني : أ. أماني الأمين مبارك

منشورات جامعة السودان المفتوحة، الطبعة الأولى 2008م

جميع الحقوق محفوظة لجامعة السودان المفتوحة، لا يجوز إعادة إنتاج أيّ جزء من هذا الكتاب، وبأيّ وجه من الوجوه، إلاّ بعد الموافقة المكتوبة من الجامعة.

مقدمة المقرر

الحمد لله بجميع محامده كلها ماعلمت منها وما لم اعلم على جميع نعمه كلها ماعلمت منها وما لم أعلم عدد خلقه كلهم ماعلمت منهم وما لم أعلم.

اللهم صلي على سيدنا محمد السلطان الاعظم والنائب العام الأكرم باب الرحمة المفتوح عند إنغلاق جميع الأبواب في وجوه المذنبين وقبس الرجاء الملموح عند سيطرة اليأس في قلوب المعذبين الباسط يد الإحسان لكل تآئب ليأخذ بيمينه إلى الصف الأول والمستغفر للغافلين من أمته لينقذهم بجاهه من الهبوط إلى الدرك الاسفل فطوبي لكل منتمي إليه و ياغرة عين المكثرين من الصلاة والسلام عليه.

عزيزي الدارس أرحب بك لدراسة هذا المقرر المسمى النظم الموزعة

يحتوي هذا المقرر على إحدي عشر وحدة تحتوي كل وحدة على مقدمة توضح محتويات الوحدة، وتبدأ بأهداف توجهك في دراسة الوحدة، تتخلل الوحدة عدد من التدريبات المقصود منها تعمق فهم المادة، وكذلك عدد من أسئلة التعيين الذاتي التي ستحصل على إجابتها بشكل مباشر من نص الوحدة. وقد أُختتمت كل وحجة دراسية بذكر المصطلحات وشرح معناها كما زيلت كل وحجة بإجابات التدريبات والمراجع.

الوحدة الأولى من هذا المقرر تتناول التطور التاريخي للحاسبات وتوضح كيف كانت أجهزة الحاسب الآلي كبيرة الحجم وباهظة الثمن حتى عام 1985، حتى إن أجهزة الحاسب الصغير mini computer كانت تكلف عشرات الآلاف من الدولارات، ولا تمتلكها إلا المنظمات والهيئات الكبيرة، وحتى في هذه الأماكن كانت توجد منفصلة وغير مرتبطة مع بعضها. ثم بدأ تطوير المعالجات القوية والسريعة في منتصف الثمانينات بدءاً من المعالجات ذات 8 بت إلى المعالجات ذات 64 بت و ما فوقها. الوحدة الثانية تتناول تعريف **الأنظمة الموزعة**.

والوحدة الثالثة معمارية النظام الموزعة التي سنتعرف فيها على العتاد المستخدم لبناء النظم الموزعة من أجهزة حواسب منفصلة، وكذلك نتناول طريقتان لتوصيل هذه الحاسبات

مـع بعضـها الـبعض، طريقـة الـذاكرة مشـتركة وتسـمى الـنظم متعـددة المعالجـات **multiprocessors**.أمـا الوحـدة الرابعـة فتعـدد برمجيـات الـنظم المـوزع وفـي الوحـدة الخامسة هيكلة النظام الموزع فسنتعرف فيها على تقسيم التطبيق إلى مستويات ونوضح العلاقة بين الخادم والعميل، فأحيانا يكون الخادم عميلاً لخادم آخر فهو من جانب خادم ومن جانب آخر عميل. وأيضا نوضح فيها تقسيم تطبيقات الخادم والعميل والتي تربط المستخدمين بقواعد البيانات ومستوياتها الثلاثة أما الوحدات الوحدة السادسة والوحدة السابعة.

والوحدة الثامنة فتبحث في مفاهيم النظم الموزعة الشفافية Transparency والانفتاحية (Openness) والتوسعية (Scalability).

الوحدة التاسعة نبين الكائنات الموزعة و تتحدث عن فكرة الكائنات الموزعة المبنية على نداء الإجراء البعيد (RPC)، ثم نرى كيف يطبق مفهومه على الكائنات الموزعة. واستعمال نداء الإجراء البعيد في تقنية الخادم/العميل، فهنالك برنامج بجهاز يطلب تنفيذ إجراء بجهاز آخر.

الوحدة العاشرة والحادية عشر تقدم تطبيق عملي للنظم الموزعة يركز على طبقة RMI الوسطى طبقة خدمات الويب الوسيطة.

وقد تميز المقرر بسهولة أسلوبه وشمولية طريقة عرضه للمادة متفقة بذلك مع مبدأ التعلم عن بعد . راجيين أن نكون قد وفقنا بالوصول إلى الأهداف الموضوعة لهذا المقرر.

الأهداف العامة للمقرر

عزيزي الدارس بعد فراغك من هذا المقرر ينبغي أن تكون قادراً على أن:

1. تتعرف على النظم الموزعة
2. تشرح معمارية النظام الموزعة
3. تبين بمفاهيم النظم الموزعة الشفافية والانفتاحية والتوسعية.
4. تفهم فكرة الكائنات الموزعة المبنية على نداء الإجراء البعيد (RPC)
5. تطبق النظم الموزعة بالتركيز على طبقة RMI الوسطى بلغة الجافا

محتويات المقرر

الوحدة	الموضوع	الصفحة
الوحدة الأولى	التطور التاريخي للحاسبات	1
الوحدة الثانية	الأنظمة الموزعة	23
الوحدة الثالثة	معمارية النظام الموزع	42
الوحدة الرابعة	برمجيات النظم الموزع	60
الوحدة الخامسة	هيكلة النظام الموزع	92
الوحدة السادسة	الشفافية Transparency	107
الوحدة السابعة	الانفتاحية (Openness)	120
الوحدة الثامنة	التوسعية (Scalability)	137
الوحدة التاسعة	الكائنات الموزعة	149
الوحدة العاشرة	طبقة RMI الوسيط	162
الوحدة الحادية عشر	طبقة خدمات الويب الوسيطة	263


الوحدة الأولى
التطور التاريخي للحاسبات

محتويات الوحدة

الموضوع	رقم الصفحة
المقدمة	3
الأهداف	3
1. تطور الحاسبات	4
1.1 صناعة العتاد	5
2.1 صناعة الشبكات	12
3.1 صناعة البرمجيات	**14**
2. العصور الثلاثة	16
1.2 عصر الحاسب المركزي (حاسب واحد لمستخدمين كثر)	16
2.2 عصر الحاسبات الشخصية (واحد لواحد)	**17**
3.2 عصر التعددية(عدد من المعالجات/الحاسبات لمستخدم واحد)	**17**
3. التطبيقات التي تحتاج إلى أجهزة فائقة السرعة	**18**
1.3 لإنجاز مثل هذه التطبيقات لدينا واحد من خيارين	**18**
2.3 من التطبيقات التي تحتاج سرعة عالية	**19**
الخلاصة	20
لمحة مسبقة عن الوحدة القادمة	21
اجابات التدريبات	22
مسرد المصطلحات	23
المراجع	24

المقدمة

تمهيد

عزيزي الدارس،

مرحباً بك إلى هذه الوحدة، التي سنتناول فيها التطور التاريخي للحاسب الآلي وكيفية تأثير ذلك على صناعة العتاد وصناعة الشبكات، وكذلك تأثيرها على البرمجيات

منذ صناعة أول حاسب وحتى يومنا هذا يظل هنالك ثابت واحد لم ولن يتغير بتغير التكنولوجيا و بتغير الزمن، هو طبيعة النفس البشرية الطموحة والتواقة لصعود كل قمة، فهي لا تفتر أبداً عن سعيها الشره دوماً نحو الأفضل. وفي القسم الثاني من الوحدة سنتناول بداية ظهور الأنظمة الموزعة.

وفي القسم الثالث منه سنتناول مفهوم المشاركة الزمنية. أما القسم الأخير فيتحدث عن طرق تحسين أداء الحاسب الآلي.

عزيزي الدارس في هذه الوحدة قدمنا بعض الأمثلة والتدريبات البسيطة لترسيخ المفاهيم الموضحة، نرجو أن تستفيد منها وأن تشاركنا في نقدها وتقويمها.

أهداف الوحدة



عزيزي الدارس بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً علي أن:

1. تبين أثر التطور في الحواسب في البرمجيات.
2. تعرف بداية ظهور الأنظمة الموزعة وعلاقتها مع الشبكات.
3. تبين معني المشاركة الزمنية.
4. تذكر ثلاث طرق لتحسين أداء الحاسب.

1. تطور الحاسبات

عزيزي الدارس، بدأ الإنسان بصناعة حاسب: **ضخم – بطئ – مكلف.**

وبسرعة شديدة وفي زمن وجيز بدأت تتغير هذه العوامل الثلاث حتى وصلنا في يومنا هذا إلى حاسب: **صغير– سريع– رخيص.**

تطور الحاسب لم يكن في العتاد لوحده (hardware) وإنما في كل شئ مرتبط به، وإلا لما أفاد تطور العتاد لوحده.

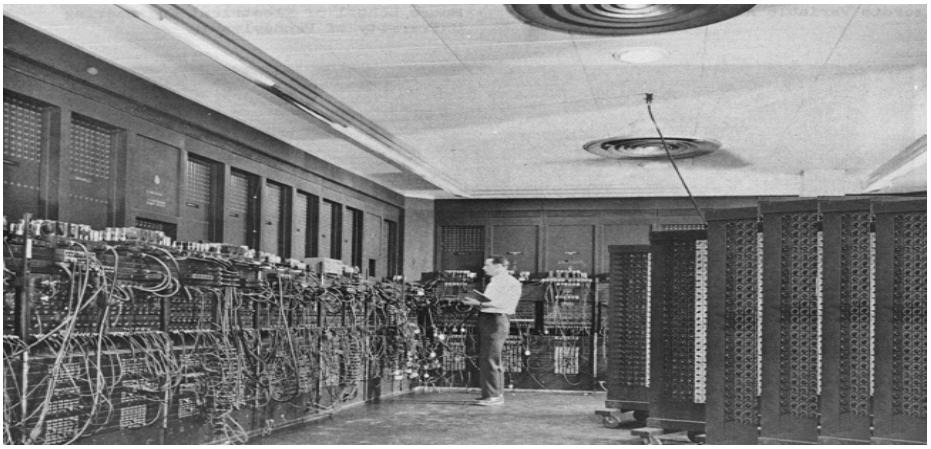
التطور كان في ثلاثة مجالات (بما فيها العتاد) هي:

التطور في صناعة العتاد	التطور في مجال الشبكات	التطور في صناعة البرمجيات

حيث لا تنفك هذه المجالات عن بعضها البعض.

1.1 صناعة العتاد

بدأت الحاسبات كبيرة الحجم (بحجم غرفة كبيرة)، وباهظة السعر بحيث لا يمكن لفرد امتلاكها، وبطيئة (تستخدم فقط للعمليات الحسابية). انظر الشكل (1–1).



شكل (1–1) : الحاسب انياك 1

إنياك (**ENIAC – Electronic Numerical Integrator And Computer**)

قام العالم جون موشلي (عالم فيزيائي) وساعده المهندس بريسبر إكيرت، ببناء أول حاسب رقمي إلكتروني، كان ذلك في عام 1945. هذا الحاسب كانت **مواصفاته** كالتالي:

- وزنه 27000 كيلو (سبعة وعشرون ألف كيلو فقط لا غير !).
- يحتوي على 18000 (ثمانية عشر ألف) صمام مفرغ (**vacuum tubes**).
- تتعطل شهرياً حوالي 2000 (ألفا) صمام مفرغ مما يتطلب استبدالها.
- صمم ليقوم بعمليات الجيش الأمريكي الحسابية لتصميم الأسلحة الذرية.
- ليس له مقدرة على تخزين البرامج، فلابد من إعادة برمجته لكل مهمة جديدة.
- الطاقة الكهربية التي يحتاجها عالية.

بعدها بفترة قامت نفس المجموعة بصناعة الحاسب يونيفاك (UNIVAC)، وفي العام 1957 كان هنالك حوالي 46 حاسباً من هذا النوع مستخدمة.

1.1.1 تطور العتاد في العصر الحديث

عزيزي الدارس هذا ما نقوله نحن عنها، فماذا قال عنها أصحابها حين صنعوها ؟!

تحدثوا عن إنياك فرحة به في يونيو 1947، فقالوا: "إن التطور اليومي والتوسع في استخدام المسائل الحسابية المعقدة جعل هنالك حاجة ماسة للسرعة. فلا توجد آلة تنجز و تحل هذه العمليات الحسابية المعقدة. الآن وبظهور هذه الأجهزة الحديثة الحالية قلّ الكثير من الوقت المستغرق في حل مثل هذه المسائل، فبدلا من أن يستغرق الحل شهوراً أو أياماً أصبح ينجز في ثوانٍ".

الآن الحاسبات الآلية سريعة جداً، رخيصة جداً، وصغيرة جداً .



شكل رقم (1–2) : حاسب كفي من شركة HP



رقم (1–3) : أصغر حاسب لينكس في العالم (يسمى picotux 100)



رقم (1-4) : أصغر حاسب شخصي في العالم.

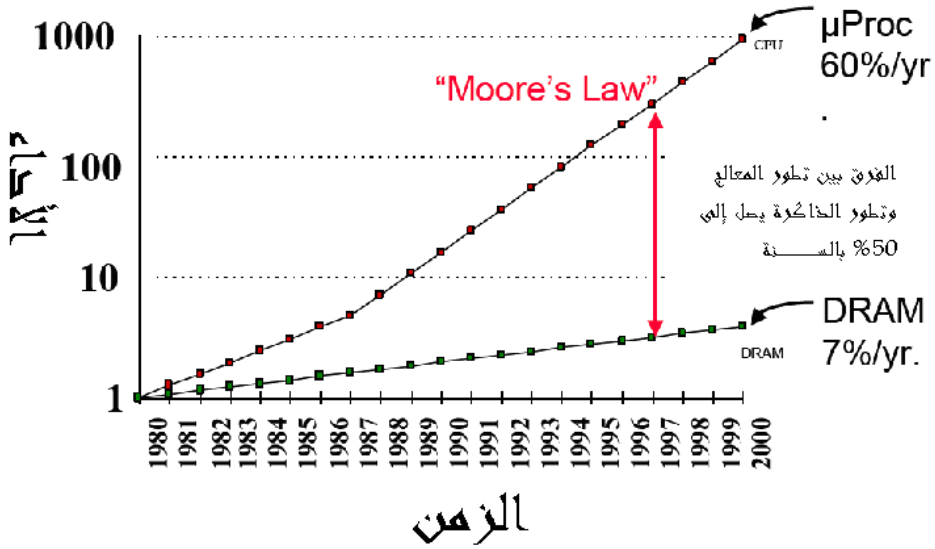
أصغر حاسب شخصي

هو حاسب شخصي صفاته كالآتي:

- **o** حجمه 2×2×2.2 بوصة !. **o** اسمه Space Cube.
- **o** سرعة معالجه 300MHz. **o** ذاكرته حجمها 64MB.
- **o** يستطيع التعامل مع شاشات بدقة تصل إلى 1280×1024 نقطة (pixel).
- **o** سعره حوالي 325 دولار. **o** تاريخ الإنتاج (منتصف يناير 2007)

2.1.1 قانون مور

هنالك فرضية تقول "إن صناعة العتاد تتضاعف سرعتها كل 18 شهراً"، هذه الفرضية تسمى قانون مور (Moore's law)، استمر هذا القانون صحيحاً لفترة طويلة. ولكن إلى متى سيظل هذا القانون صحيحاً ؟



شكل رقم (1-5) : فرضية قانون مور من 1980 وحتى عام 2000

3.1.1 العقبات تقف أمام استمرارية زيادة سرعة المعالج

عزيزي الدارس قريبا ستصل سرعة المعالجات إلى نهايتها، أي قد لا يكون هنالك حل لزيادة السرعة بزيادة سرعة المعالج)، **نذكر منها:**

1. إن **سرعة الحاسب ككل مرتبطة بسرعة أبطأ جهاز به.** ونلاحظ من الشكل (1-5) أن سرعة المعالج تزيد بنسبة 60% سنوياً ، بينما تزيد النسبة للذاكرة بـ 7% فقط بالسنة.تعتبر الذاكرة مؤثراً رئيسياً على سرعة الحاسب، فكأنك امتلكت سيارة سريعة (مثلا بسرعة 240 كليو/ساعة) ولكن الطريق لا يسمح باستخدام هذه السرعة (لا تستطيع القيادة بأعلى من 80 كيلو/ساعة)، يلاحظ هنا أن 160 كيلو/ساعة من سرعة السيارة غير مستغلة (ترتبط سرعة السيارة بسرعة الطريق). بنفس القدر يتعامل المعالج مع البيانات التي تحضر من الذاكرة، بالتالي سنفقد جزءاً كبيراً من سرعة المعالج بسبب الزمن المستغرق في نقل المعلومة من الذاكرة إلى المعالج (ترتبط سرعة المعالج بسرعة وصول البيانات من الذاكرة).

لجعل المعلومة تنتقل أسرع لابد من تقليل المسافة المقطوعة بين المعالج والذاكرة.

2. كم سرعة معالجك الآن ؟ وكم تتوقع أن تصل سرعة المعالجات ؟ 4 غيغا هيرتز ؟ 100 غيغا هيرتز ؟ أم أكثر أم أقل ؟

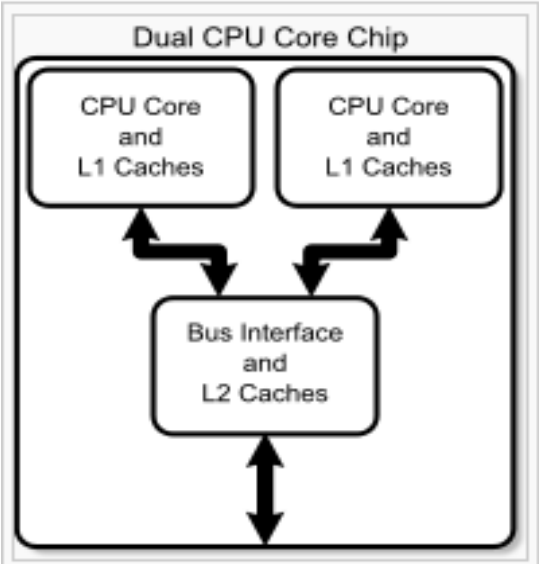
إذا كنت تمتلك حاسباً بسرعة 1 MHz، فهذا يعني أن في كل نانو ثانية سيقوم المعالج بعمل بدورة (cycle) واحدة (أو بمعنى آخر بليون دورة (clock cycle) في الثانية الواحدة).

الضوء يستطيع عبور 30 سنتمتراً (حوالى قدم) في النانو ثانية (nanosecond) أو في 1 من البليون من الثانية. والمعلوم أن الإشارات الرقمية تنتقل بسرعة الضوء.

لذلك الدارات الإلكترونية التي تنتج سرعة المعالج (clock speeds) من الأفضل ألا يزيد حجمها (على الأقل عُشرها) عن 30 سنتمتراً. هذا يعني أن أكبر حجم للدارة الإلكترونية يجب ألا يتعدى 3 سنتمتر.

إذا أردنا تصميم معالج بسرعة 100MHz، فهذا يعني أنه لابد لهذا المعالج من إنتاج الدورة الواحدة (clock cycle) في كل 1 من المائة من النانو ثانية. هذا يعني أن الإشارة ستصل إلى 3 ملمتر في النانو ثانية (تنتقل بسرعة الضوء). لذلك من الأفضل أن يكون حجم قلب المعالج 0.3 ملمتر. وقد يكون من العسير جداً صناعة معالج بهذا الحجم لكنه – مع وجود التقنيات الحديثة – أمر ممكن.

من هنا نستنتج أنه كلما قلصنا حجم الدارات الإلكترونية زادت سرعة المعالج، ولكن إلى متى سنظل نقلص في الحجم ؟ بالتأكيد سنصل نقطة يصبح فيها تقليص الحجم مستحيلاً ، وبالتالي ستصبح زيادة سرعة المعالج وتحسين أداء الحاسب ذي المعالج الواحد مستحيلاً . بمعنى آخر ستصل سرعة الحاسب إلى نهايتها، ولن نستطيع زيادة الميغاهيرتز (MHz)، وهذا واضح الآن في سوق الحاسبات، فقد بدأت حالياً الكثير من الشركات المنتجة للعتاد تتجه نحو زيادة عدد المعالجات في الحاسب بدلاً من زيادة سرعة المعالج، فظهر ما يسمى المعالجات متعددة النواة (multi-core CPU)، شكل رقم (1–6).



شكل رقم (1–6): معالج ثنائي النواة

أسئلة تقويم ذاتي

- ما فرضية مور في مجال سرعة الحاسب ؟
- هل نعتبر فرضية مور لسرعة الحاسب ما تزال صحيحة ؟ و لماذا ؟
- ما مواصفات أصغر حاسوب شخصي ؟

تدريب (1)



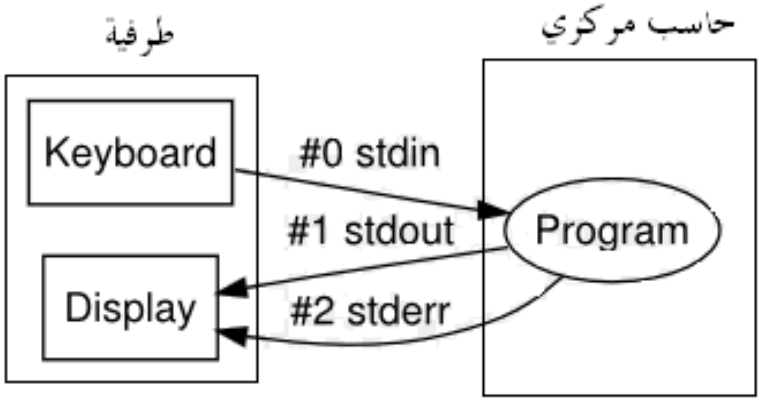
أجب بلا أو نعم :

1. أول حاسب آلي كان وزنه 27000 كيلو
2. حاسب إنياك يحتوي على (ثمانية عشر ألف) صمام مفرغ.
3. في الأجهزة القديمة تتعطل شهرياً حوالي 20صمام مفرغ.
4. أجهزة الحاسبات صممت لتقوم بعمليات الجيش الأمريكي الحسابية لتصميم الأسلحة الذرية.
5. الأجهزة القديمة ليس له المقدرة على تخزين البرامج.
6. الأجهزة القديمة لابد من إعادة برمجتها لكل مهمة جديدة.
7. الطاقة الكهربية التي يحتاجها حاسب يونيفاك عالية .
8. أصغر حاسب شخصي حجمه 2×2×202بوصة.
9. سرعة الحاسب ككل مرتبطة بسرعة أبطأ جهاز به.
10. لجعل المعلومة تنتقل أسرع لابد من تقليل المسافة المقطوعة بين المعالج والذاكرة.

2.1 صناعة الشبكات

بدأت فكرة الشبكات عن طريق حاسب مركزي (mainframe) واحد، ترتبط به طرفيات تمكن عدد كبير من المستخدمين من الاستفادة من هذا الحاسب بالتشارك فيه عن طريق التقسيم الزمني (time sharing).

الشكل (1−7) يوضح كيف تتعامل الطرفيات من الحاسب المركزي، حيث يتم تنفيذ البرنامج بالجهاز المركزي بينما تقوم الطرفية بإدخال البيانات وإظهار المخرجات (بما فيها الأخطاء).



شكل رقم (1–7): طرفية تتعامل مع برنامج (program) بالجهاز المركزي

ثم تطورت فكرة الشبكات مع ظهور الحواسيب الشخصية (PCs) حيث رُبِطت هذه الحواسب الشخصية مع بعضها البعض بشبكة (وأصبح التطور في نوعية الشبكة من حيث الكوابل والبرامج مستمر إلى يومنا هذا) ، فانتقلنا إلى شبكات متنوعة، سلكية ولاسلكية تربط العديد من الأجهزة المركزية والشخصية والمتنقلة مع بعضها البعض.

فمن منا يستطيع اليوم الاستغناء عن الشبكات ؟ لا يوجد أحد (صفر شخص). فالإنترنت والمعاملات والجوازات والحجوزات والبنوك وصرافات ATM، كلها تعتمد اعتماداً كلياً على الشبكات. حتى أصبح العالم كله مرتبطاً وغطت الشبكات الأرض بكوابلها والسماء بترددها، وعمَّت القرى والحضر.

1.2.1 فوائد الشبكات

يمكن الاستفادة من الشبكات في مجالات عديدة مثل:

- التشارك في الموارد من عتاد وبرامج وبيانات.
- القوة، حيث يمكن استمرارية النظام حتى ولو تعطلت أجزاء منه.
- توزيع الأعباء على أجهزة الشبكة لتتعاون في إنجاز المهام الكبيرة (load balancing).
- الوصول للمعلومات من أي مكان (مثلا يمكنك تصفح الإنترنت من بريطانيا أو من اليابان).

الشبكات في تطورٍ مستمر شأنها في ذلك شأن العتاد. هنالك فرضية للشبكات تقول: "إن سرعة نقل البيانات في الشبكات تتضاعف كل 9 أشهر"، هذا يعني (حسب الفرضية) أن التطور في الشبكات أسرع مرتين من التطور في العتاد (والتطور في العتاد لا يوازيه أي تطور في أي مجال آخر). من هنا يمكننا القول بأن التطور السريع في الشبكات هو في صالح النظم الموزعة.

3.1 صناعة البرامج

بدأت الحاسبات بلا برامج ونظم تشغيل على نغيض ما عليه الحال اليوم، حيث يحتاج التعامل مع الحاسب إلى مهندس خبير أو مبرمج خطير حتى يستطيع الاستفادة والتعامل مع الحاسب، فالعمل كله يتم بلغة الآلة، ثم جاءت فكرة تعيين شخص مختص يسمى المشِلّغ يجلس بقرب الحاسب ليساعد المبرمجين والمستخدمين على التعامل مع الحاسب حيث يقوم هذا المشغِلّ باستلام برامج المبرمجين وبيانات المدخلين ويقوم هو بإدخالها وتشغيلها ثم يسلمهم النتائج مطبوعة على ورق.

بعدها بدأت عملية الإحلال تدريجياً (حيث صممت برامج تؤدي بعض المهام التي كان يقوم بها المشِلّغ وبدأت أعباؤه تقل تدريجيا والبرامج البديلة في ازدياد، وهكذا إلى أن استبدل المشِلّغ تماما ببرامج تقوم بكل ما كان يقوم به هو. هذه البرامج البديلة يسرت

للمستخدم الاستفادة من الحاسب وسميت ببرامج نظم التشغيل، حيث وصلنا درجة بحمد الله تمكن أي شخص من التعامل مع الحاسب بمنتهى السهولة.

نلحظ أن التطور في المجالات الثلاثة متداخل ومتبادل، فقد يتطور العتاد لتلبية طلبات المبرمجين (البرامج) أو قد تتطور الشبكات لمواكبة التطور في العتاد، أو قد يقود تطور البرامج إلى ابتكار عتاد جديد. أيضا هنالك سباق بين صناعة العتاد وصناعة البرمجيات ، فكلما ظهر عتاد جديد تفنن المستخدم في استغلاله وقد يطلب المزيد "إن الطعام يقوي شدة النهم"، وقد يفكر الإنسان في برمجيات واستخدامات لا يستطيع العتاد الحالي تلبيتها فينتظر بزوغ عهد جديد بعتاد جديد يلبي تطلعاته أو يحاول الاستفادة من العتاد الحالي **بالتحسين في الخوارزميات وطريقة كتابة البرامج**.

1.3.1 طرق تحسين البرمجيات

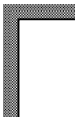
في حياتنا العادية قد نستخدم طرقاً عديدة لتحسين الأداء مثل:

- أن نعمل بقوة ونشاط أكثر (work harder).
- نستخدم أساليب متطورة في تجويد العمل (work smarter).
- أو نطلب مساعدة أشخاص آخرين لينجزوا معنا العمل (get help).

أيضاً يمكننا استخدام نفس الطرق لنحصل على فائدة أكبر من عتاد الحاسب، مثل:

- استبدال الحاسب بآخر أسرع (أو تبديل أجزاء منه بأخرى سريعة).
- استغلال موارد الحاسب بالصورة المثلى: مثل ما يقوم به نظام التشغيل من الاستفادة القصوى من المعالج طول الوقت (CPU scheduling)، أو إضافة أنابيب انسيابية في المعالج (pipeline) لتمكينه من تنفيذ أوامر أكثر (instruction parallelism).
- استخدام أكثر من حاسب لإنجاز العمل (تتعاون الحاسبات في إنجاز العمل فيقوم كل حاسب بتنفيذ جزئية من العمل).

أسئلة تقويم ذاتي



- يمكن تقسيم التطور في الحاسب إلى ثلاثة مجالات رئيسية، ما هي؟
- ما العصر الذي ظهرت فيه الأنظمة الموزعة؟ وهل لهذا علاقة مع الشبكات؟

2. العصور الثلاثة

عزيزي الدارس،

التطور أعلاه يمكن تقسيمه إلى ثلاثة عصور رئيسية كان التحول فيها واضحا حيث اختلفت طريقة استخدام الحاسب بين كل عصر وآخر.

هذه العصور هي:

عصر الحاسبات المركزية.	عصر الحاسبات الشخصية	عصر الحاسبات

1.2عصر الحاسب المركزي (حاسب واحد لمستخدمين كثر)

بدأ الحاسب باهظ الثمن لا يستطيع شخص عادي امتلاكه ، فكان الحل التشارك في استخدامه عن طريق طرفيات (terminal) (لوحة مفاتيح وشاشة لكل مستخدم)، بهذه الطريقة نمكن أكبر عدد ممكن من المستخدمين الاستفادة من حاسب واحد (mainframe).

يقوم برنامج بالحاسب بإعطاء كل طرفية فترة زمنية وجيزة للاستفادة من الحاسب و تبديل الفرص بين المستخدمين فيما يسمى المشاركة الزمنية (time sharing).

2.2 عصر الحاسبات الشخصية (واحد لواحد)

في هذه الحقبة الزمنية أصبحت الحاسبات صغيرة وقليلة التكلفة، حيث يمكن لكل شخص شراء حاسب شخصي و استخدامه في بيته أو في مكتبه. وبالتالي لا حاجة إلى برامج مشاركة زمنية، ولا حاجة إلى طرفيات وتزاحم حول حاسب واحد، فكان ظهورنظام التشغيل الأحادي في هذا العصر.

3.2 عصر التعددية (عدد من المعالجات/الحاسبات لمستخدم واحد)

ظهرت الشبكات التي تربط الحاسبات الشخصية فيما بينها وتُمكِّن المستخدمين من التشارك في العتاد والبرامج ، فحلَّت مكان الحاسبات المركزية القديمة.

هنا توسع الإنسان في طموحه وبدأ يصمم ويطلب برامج تحتاج إمكانيات أعلى من طاقة الحاسبات الشخصية، فكان الحل زيادة عدد المعالجات في الحاسب لتوفي الطلب أو استخدام أكثر من حاسب عبر الشبكة لإنجاز العمل.**في هذا العصر كان ميلاد الأنظمة الموزعة موضوع هذا المقرر.**

أسئلة تقويم ذاتي



- ماذا نعني بالمشاركة الزمنية ؟
- هنالك ثلاث طرق لتحسين أداء الحاسب، اذكرها.
- متى ظهر عصر الأنظمة الموزعة؟

تدريب (2)



رتب القائمة (ب) مع ماينا سبها من القائمة (أ)

القائمة (أ)	القائمة (ب) المرتبة	القائمة (ب) الغير مرتبة
1800 صمام مفرغ		Space Cube
حجم الحاسب 2×2×202 بوصة		الحاسب مركزي
مكلفة جداً		الحاسبات الفائقة
تشارك الزمن		الحاسب إنياك

3. التطبيقات التي تحتاج إلى أجهزة فائقة السرعة

عزيزي الدارس،

هنالك العديد من التطبيقات التي تحتاج إلى قوة معالجة عالية جداً، لا يمكن للحاسبات العادية أن تنجزها.

1.3 لإنجاز مثل هذه التطبيقات لدينا واحد من خيارين هما:

1. استخدام الحواسب الفائقة (super computers) وهي مكلفة جداً .
2. استخدام عدد كبير من الحاسبات العادية التي توزع التطبيق بينها لتنجزه متعاونة مع بعضها البعض. أحياناً نطلق على هذه الطريقة "حاسب الفقراء الفائق". فالذي لا يستطيع شراء حاسب فائق يمكنه تجميع حاسبات عادية يُكوّن بها حاسب فائق خيالي (Virtual Super Computer).

2.3 من التطبيقات التي تحتاج سرعة عالية

- النمذجة و المحاكاة (simulation and modeling).
- التعرف على الكلام (speech recognition).
- معالجة الصور (image processing).
- التنبؤ بالطقس (Climate Modeling).
- استغلال (تعدين) البيانات (Data Mining).

الخلاصة

عزيزي الدارس تعرفنا في هذه الوحدة على التطور في الحواسيب، وبسرعة شديدة وفي زمن وجيز حتى وصلنا في يومنا هذا إلى حاسب: صغير- سريع- رخيص. وأن التطور في الحاسب لم يكن في العتاد لوحده (hardware) وإنما في كل شئ مرتبط به، و إلا لما أفاد تطور العتاد لوحده.

التطور كان في ثلاثة مجالات (بما فيها العتاد) هي: التطور في صناعة العتاد، التطور في مجال الشبكات، التطور في صناعة البرمجيات.

تطور العتاد وبرامجه قادنا إلى التوسع في استخدامه وإقحامه في كل صغيرة وكبيرة. التطور في مجال الشبكات جعل بالإمكان الاستعانة (بعد الله سبحانه وتعالى) بحاسبات بعيدة عنا واستخدامها كأنها معنا في بيتنا أو في مكتبنا. وهذا ما يسعى له النظام الموزع.

لمحة مسبقة عن الوحدة القادمة

عزيزي الدارس، ستتناول إنْ شاء الله الوحدة التالية تعريف النظم الموزعة وكذلك مجالات استخدامها. ونجيب عن السؤال لم نحتاج إلي النظم الموزعة؟ و ما هي الصفات التي يجب أن تتوفر في الأنظمة الموزعة؟ كما سنتناول المخاطر التي تواجهنا في استخدام الأنظمة الموزعة ومن أهمها مشاكل الشبكة، وسنبين معمارية وخصائص الشبكات التي تستخدمها النظم الموزعة.

كذلك سنتعرف في الوحدة القادمة على المجالات المتعددة التي تستخدم في تقنياتها النظم الموزعة وأخيرا سنتناول عدداً من الأمثلة تتناول الأنظمة الموزعة بطبيعتها.

فما الأنظمة الموزعة؟ و ما قصتها ؟

هذا ما سنتعرف عليه في الوحدة القادمة بإذن الله.

إجابات التدريبات

تدريب (2)

رتب القائمة (ب) مع ماينا سبها من القائمة (أ)

القائمة (أ)	القائمة (ب) المرتبة	القائمة (ب) الغير مرتبة
1800 صمام مفرغ	الحاسب إنياك	
حجـــم الحاســـب 2×2×202 بوصة	Space Cube	
مكلفة جداً	الحاسبات الفائقة	
تشارك الزمن تشارك الزمن	الحاسب مركزي	

أجب بلا أو نعم :

رقم السؤال	1	2	3	4	5	6	7	8	9	10
الإجابة	نعم	نعم	لا	نعم	نعم	نعم	نعم	نعم	نعم	نعم

مسرد المصطلحات

صناعة العتاد

بدأت الحاسبات كبيرةَ الحجم (بحجم غرفة كبيرة) وباهظَة السعرِ بحيث لا يمكن لفرد امتلاكها، وبطيئةً (تستخدم فقط للعمليات الحسابية).

قانون مور

قانون يقول بفرضية "إن صناعة العتاد تتضاعف سرعتها كل 18 شهراً"، هذه الفرضية تسمى قانون مور (Moore's law) .

صناعة الشبكات

بدأت فكرة الشبكات عن طريق حاسب مركزي (mainframe) واحدٍ، ترتبط به طرفيات تمكّن عدداً كبيراً من المستخدمين من الاستفادة من هذا الحاسب بالتشارك فيه عن طريق التقسيم الزمني (time sharing).

صناعة البرامج

بدأت الحاسبات بلا برامج ونظم تشغيل على نغيض ما عليه الحال اليوم، حيث يحتاج التعامل مع الحاسب إلى مهندس خبير أو مبرمج خطير حتى يستطيع الاستفادة والتعامل مع الحاسب، فالعمل كله يتم بلغة الآلة، ثم جاءت فكرة تعيين شخص مختص يسمى المشِلّغ يجلس بقرب الحاسب ليساعد المبرمجين والمستخدمين على التعامل مع الحاسب حيث يقوم هذا المشغِّل باستلام برامج المبرمجين وبيانات المدخلين ويقوم هو بإدخالها وتشغيلها ثم يسلمهم النتائج مطبوعة على ورق.

المراجع

1- 1.Coulouris, G. et al., (2004), Distributed Systems, Concepts and Design, Addison Wesley
2- Farley, J. (1998), Java Distributed Computing, O'reilly & Associates
3- Fedorov, A, (2002), A Programmer Guide to .NET, Addison Wesley
4- Ince, D. (2002), Developing Distributed and E-Commerce Applications, Pearson Education
5- Liu, M. (2004), Distributed Computing Principles and Applications, Pearson Education
6- Schonauer, W. (1998), Scientific Supercomputing: Architecture and use of Shared and Distributed Memory Parallel Computers, Willi Schonauer, Germany
7- Tanenbaum, A., (2002), The Amoeba Distributed Operating System, Vrije Universiteit Technical Report, URL ftp://ftp.cs.vu.nl/pub/amoeba/Intro.ps.Z
8- Tanenbaum , A. AND Van Steen, M. (2002), Distributed Systems Principles and Paradigms, Prentice Hall
9- Tanenbaum, A. (1995), Distributed Operating Systems, 1/e, Prentice Hall, USA
10- Osman, A., (may 2006), A Multi-level Web-based Parallel Processing System, TRANSACTIONS ON ENGINEERING, COMPUTING AND TECHNOLOGY VOLUME 13 MAY 2006 ISSN 1305-5313
11-Forouzan, A. (2007), Data Communications and Networking, Mc Graw Hill, Fourth Edition
12-Tanenbaum, A., (2001), Modern Operating Systems, Prentice Hall, Second Edition
13-Tanenbaum, A., (1999), Structred Computer Organization, Prentice Hall, Fourth Edition


الوحدة الثانية
الأنظمة الموزعة

محتويات الوحدة

الموضوع	الصفحة
المقدمة	28
تمهيد	28
أهداف الوحدة	**29**
1. تعريف النظم الموزعة	**30**
1.1 تعريف النظام الموزع عتاداً	30
2.1 تعريف النظام الموزع برامجياً	31
3.1 الاستنتاجات العشرة من التعريف السابق	31
2. معمارية النظم الموزعة	33
1.2 معمارية النظام الموزع المبني على نظام الشبكات	33
2.2 معمارية النظام الموزع المبني على نظام البرمجيات	34
3. لماذا نحتاج إلى الأنظمة الموزعة	34
1.3 انخفاض أسعار الحاسبات في استمرار دائم، وزيادة سرعة المعالجة في ارتفاع دائم	34
هنالك بعض التطبيقات التي لا يمكن تنفيذها على الحاسبات العادية.	35

4. صفات الأنظمة الموزعة	37
1.4 الشفافية (اخفاء التفاصيل)	37
2.4 المرونة	38
3.4 الاعتمادية	38
4.4 الأداء	38
5.4 التوسعية	38
6.4 التباينية	39
7.4 الحماية والتأمين	39
8.4 تجاوز الأعطال	39
9.4 الانفتاحية	40
10.4 التزامنية	40
الخلاصة	43
لمحة مسبقة عن الوحدة القادمة	43
إجابات التدريبات	44
مسرد المصطلحات	45
المراجع	47

المقدمة

تمهيد

عزيزى الدارس، مرحباً بك إلى هذه الوحدة

والتي بعنوان "الأنظمة الموزعة"، وهي تعد الوحدة الأساسية لهذا المقرر والتي سنتعرض فيها بالتفصيل لكيفية عمل الأنظمة الموزعة.

تنقسم هذه الوحدة إلى أربعة أقسام رئيسة، حيث يشرح القسم الأول الإجابة عن السؤال المهم والأساس "ما الأنظمة الموزعة ؟" تأتي في القسم الأول من هذه الوحدة والذي يشرح تعريف الأنظمة.

نتمنى أن تتمكن عزيزي الدارس من الإجابة عن السؤال الأساسي وهو في ذات الوقت، الهدف الأساس من هذه الوحدة في القسم الثاني سنتناول مجالات استخدام الأنظمة الموزعة و نبين لم نحتاج إليها. أما القسم الثالث فسيناقش الصفات التي يجب أن تتوفر في الأنظمة الموزعة فيها. وفي القسم الأخير من هذه الوحدة سنتناول المخاطر التي تواجهنا في استخدام الأنظمة الموزعة، ومن أهمها مشاكل الشبكة. وسنبين معمارية وخصائص الشبكات التي تستخدمها النظم الموزعة. يتخلل هذه الوحدة عدد من التدريبات وأسئلة التقويم الذاتي نتمنى عليك عزيزي الدارس الاهتمام بها ومحاولة الإجابة عنها جميعاً لأنها ستساعدك في ترسيخ المفاهيم الأساسية لهذه الوحدة. كما نتمنى أن تتأكد من أن ما جاء بهذه الوحدة قد حقق أهدافها، وأن تساعدنا بنقدك وتقويمك لهذه الوحدة. والله نسأل لك التوفيق والنجاح.

أهداف الوحدة



عزيزي الدارس بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

1. تعرف النظام الموزع.
2. تذكر صفات النظم الموزعة.
3. تبين ماذا نعني بقابلية النظام للتوسع.
4. تعدد مخاطر الأنظمة الموزعة.
5. تحدد المجالات التي نستخدم فيها النظم الموزعة.
6. تبين ثلاثة أمثلة لأنظمة موزعة بطبيعتها.

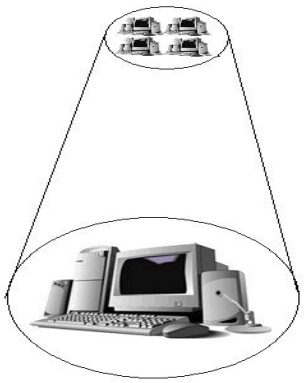
1. تعريف النظم الموزعة

عزيزي الدارس،دعنا في البداية نضع تعريفاً للأنظمة الموزعة ننطلق من خلاله لما نريد.

هنالك تعريفات عديدة للأنظمة الموزعة قد تختلف عن بعضها البعض اختلافاً كبيراً، ولكننا وضعنا هنا تعريفا نحاول أن يكون هو الأكثر شيوعاً وأكثر ملاءمة لما نحن بصدده.

1.1 تعريف النظام الموزع عتاداً

مجموعة من الحاسبات المنفصلة عن بعضها البعض، المتحدة مع بعضها البعض تكون حاسباً واحداً كبيراً افتراضياً، شكل رقم (2-1).



شكل رقم (2-1): النظام الموزع كعتاد



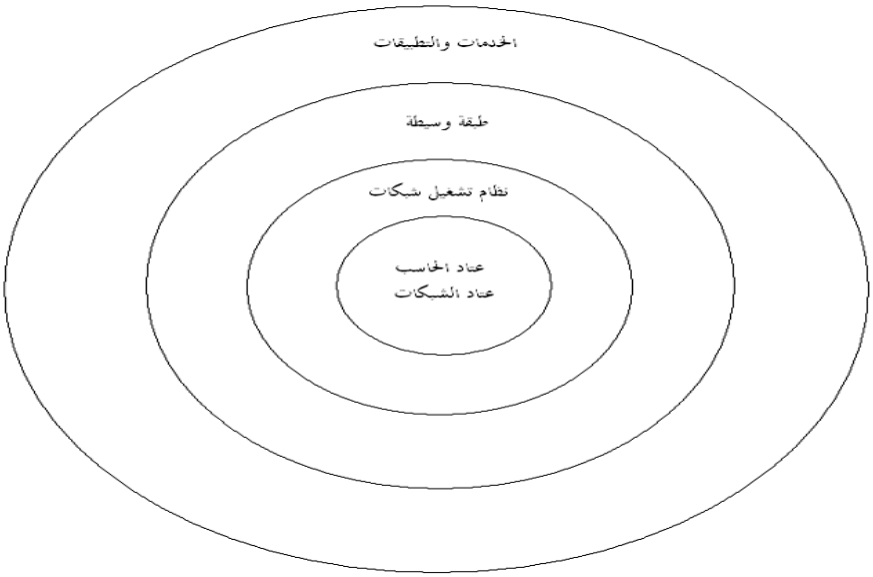
2.1 تعريف النظام الموزع برامجاً

مجموعة من البرامج الصغيرة المنتشرة في الشبكة والتي تعمل معاً لتكون ظاماً برمجياً موحَّداً يعمل على مشكلة واحدة.

3.1 الاستنتاجات العشرة من التعريف السابق

- الحاسبات المنفصلة قد تكون منفصلة عن بعضها انفصالاً جزئياً، أي عدة معالجات لديها ذاكرة مشتركة فيما يسمى تعدد المعالجات (multiprocessors)، أو قد تكون منفصلةً عن بعضها البعض انفصالاً تاماً، أي أن هنالك عدة حواسيب لكل حاسب ذاكرة خاصة به فيما يسمى تعدد الحاسبات (multicomputers).
- تعدد المعالجات غالباً ما تتصل فيه الحاسبات ببعضها البعض عن طريق ناقل بيانات (data bus) مشترك.
- تعدد الحاسبات تتصل فيه الحاسبات ببعضها البعض عن طريق شبكة.

- ليس هنالك شبكة بعينها تستخدم للنظم الموزعة ولكن كلما زادت كفاءة الشبكة وسرعتها تحسن عمل النظام الموزع والعكس صحيح (**إذ أنها تمثل البيئة التي ينبني عليها النظام**).
- النظم الموزعة (من تسميتها)، هي أجزاء (components) موزعة في حاسبات عديدة وتتصل مع بعضها البعض دون أن يعلم المستخدم بذلك.
- يتعامل المستخدم مع واجهة النظام الموزع وهي التي تدير باقي أجزاء النظام دون أن يظهر هذا للمستخدم (إخفاء هذا عن المستخدم فيما يسمى الشفافية).
- قد يكون النظام الموزع جزءاً من نظام التشغيل (Operating System) أو طبقة قائمة بذاتها (middleware) تبنى على نظام التشغيل.
- نظام التشغيل الذي يوفر خدماته مباشرة للنظام الموزع يسمى نظام التشغيل الموزع (**distributed operating system (DOS)**).
- في حالة وجود طبقة مخصصة للنظام الموزع (middleware) ، تبنى هذه الطبقة على نظام تشغيل الشبكات (Network Operating System(NOS)).
- في كل الحالات سواء استخدمنا نظام تشغيل شبكات (NOS) أم نظام تشغيل موزع (DOS)، يجب على نظام التشغيل أن يدير عتاد الحاسبات وعتاد الشبكات (computer and network hardware).



شكل رقم (2-2): معمارية النظام الموزع المبني على نظام تشغيل شبكات

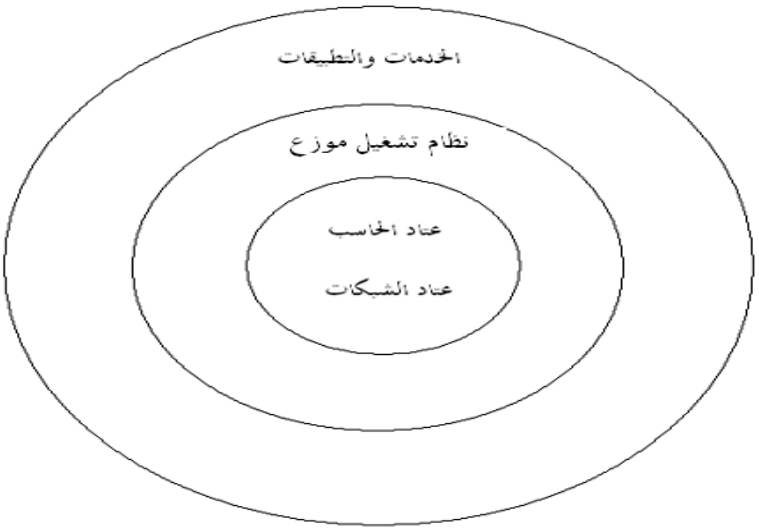
2. معمارية النظم الموزعة

1.2 معمارية النظام الموزع المبني على نظام تشغيل شبكات

عزيزي الدارس، من الشكل (2-2) نلاحظ الآتي:

منصة عمل النظام الموزع (platform) تتكون من :

- عتاد الشبكات والحاسبات .
- نظام تشغيل يدير هذا العتاد.
- الطبقة الوسيطة (middleware) التي توجد في الشكل (2-2) تساعد نظام تشغيل الشبكات على توفير بيئة ملائمة للنظام الموزع.
- قد يكون نظام التشغيل الذي يدير العتاد، هو نظام تشغيل موزع، في هذه الحالة لا نحتاج إلى الطبقة الوسيطة (middleware)، فنظام التشغيل الموزع هو الذي يوفر البيئة الملائمة للنظام الموزع، الشكل رقم (2-3).



شكل رقم (2-3): معمارية النظام الموزع المبني على نظام تشغيل موزع

2.2 معمارية النظام الموزع المبني على نظام تشغيل موزع

تُبنى تطبيقات النظم الموزعة في الطبقة العليا.

3. لماذا نحتاج إلى الأنظمة الموزعة

هنالك أسباب عديدة ساعدت وأسهمت في بروز الأنظمة الموزعة منها:

1.3 انخفاض أسعار الحاسبات في استمرار دائم، وزيادة سرعة المعالجة في ارتفاع دائم

فتقريبا كل 18 شهراً تتضاعف سرعة الحاسب مع ثبات السعر. فمن حاسب كبير ضخم بطئ يكلف ملايين الدولارات و ينفذ بضع أوامر في الثانية إلى حاسب صغير سريع ينفذ ملايين الأوامر في ميكرو من الثانية و يكلف بضع مئات من الدولارات.

نستثني من هذا الانخافض في السعر الحاسبات فائقة السرعة مثل السوبر كمبيوتر (super computers) حيث صممت لتكون بسرعة هائلة مما جعلها مكلفة في السعر. وتستخدم غالباً في المراكز البحثية الضخمة والتي تمتلك بواسطة دولة أو جامعة. ويندر ما يشتري شخص عادي أو مؤسسة صغيرة هذه الحاسبات لغلائها و لعدم الحاجة لسرعتها الفائقة.

هنالك بعض التطبيقات التي لا يمكن تنفيذها على الحاسبات العادية.

مثال :

إذا أردنا تنفيذ برنامج يقوم بالتنبؤ بحالة الطقس ليوم غد ، فعلى هذا البرنامج أن يمدنا بمخرجاته قبل وقت كاف ليستفيد منها المستخدم، ولكن عند تنفيذ هذا البرنامج في جهاز عادي مهما كانت سرعته فإنه يعطي مخرجاته بعد 48 ساعة (بعد يومين) مما يجعل النتيجة غير ذات جدوى إذ أصبحت ليوم أمس.

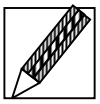
3.3 بعض التطبيقات بطبيعتها موزعة، لذلك تحتاج إلى نظام موزع، مثل النظام البنكي، نظام الحجز على خطوط الطيران، وغيرها من الأمثلة.

أسئلة التقويم الذاتي



اذكر مخاطر الأنظمة الموزعة.
اذكر اثنين من المجالات التي نستخدم فيها النظم الموزعة.
اذكر ثلاث أمثلة لأنظمة موزعة بطبيعتها. عرف النظام الموزع.

تدريب (1)



أجب بلا أو نعم

1. هنالك عدة حواسيب لكل حاسب ذاكرة خاصة به فيما يسمى تعدد الحاسبات.
2. تعدد المعالجات غالباً ما تتصل فيه الحاسبات ببعضها البعض عن طريق ناقل بيانات مشترك.
3. ليس هنالك شبكة بعينها تستخدم للنظم الموزعة ولكن كلما زادت كفاءة الشبكة وسرعتها قل عمل النظام الموزع والعكس صحيح.
4. اللوحة الأساسية للحاسب تمثل البيئة التي ينبني عليها النظام.
5. شبكة الحاسب تمثل البيئة التي ينبني عليها النظام.
6. نظام التشغيل للحاسب يمثل البيئة التي ينبني عليها النظام.
7. النظم الموزعة (من تسميتها)، هي أجزاء موزعة في حاسبات عديدة وتتصل مع بعضها البعض دون أن يعلم المستخدم بذلك.
8. قد يكون النظام الموزع جزءاً من نظام التشغيل (Operating System) أو طبقة قائمة بذاتها (middleware) تبنى على نظام التشغيل.

4. صفات الأنظمة الموزعة

لنطلق على النظام نظاماً موزعاً يجب أن يوفر للمستخدم وتطبيقاته الكثير من الصفات،وإذا اجتمعت كلها في نظام موزع معين فسيكون هذا النظام نظاماً موزعاً مثالياً . نذكر من هذه الصفات التالي:

1. الشفافية أو إخفاء التفاصيل transparency.
2. المرونة flexibility.
3. الإعتمادية reliability.
4. الأداء performance.
5. التوسعية scalability.
6. التباينية heterogeneity.
7. الحماية والسرية security.
8. تجاوز الأعطال failure handling.
9. الانفتاحية openness.
10. التزامنية concurrency.

فيما يلي سنتحدث عن كل صفة باختصار، وسنفرد أبواباً كاملة بالكتاب لأكثر الصفات أهمية مثل صفة الشفافية والتوسعية.

إذن هذه الصفات ستكون أهدافاً عندما نبني نظاماً موزعاً.

1.4 الشفافية (إخفاء التفاصيل)

يجب أن يكون النظام الموزع قادراً على إخفاء تفاصيل حقيقة وأن أجزاءه و مصادره موزعة في حاسبات عديدة، فيبدو النظام الموزع للمستخدم كأنه نظام واحد على حاسب واحد، فلا يهتم المستخدم ولا يعرف الآتي:

أ. أين توجد الأجهزة والبرامج ؟ ب. من أي مكان يتوصل إليه ؟ ج . في أي مكان توجد ؟

د .هل تستخدمها لوحدك أم معك آخرون ؟ هـ .هل تعطل جزء منها أم لا ؟

2.4 المرونة

إذا جعلنا النظام سهل التعديل والاستخدام و من اليسير تطويره و تحسينه نكون قد أكسبناه مرونة.

3.4 الاعتمادية

الاعتمادية هي مقدرة النظام على الاستمرار في كل الظروف، وغالباً هذه الخاصية لا تتحقق إلا بخاصية أخرى هي التكرار (replication).

4.4 الأداء

لتحسين أداء النظام الموزع يجب أن يُبْنَى بصورة جيدة وعلى بيئة جيدة.

هنالك العديد من العوامل التي تؤثر على الأداء مثل:

نوعية الشبكة التي يعمل عليها النظام – نوعية العتاد. – طريقة تصميم النظام.

التكرار (replication). – التطابق (consistency). – وغيرها من العوامل.

5.4 التوسعية

أحد أهم أهداف مطوري النظم الموزعة هو قابلية النظام للتوسع والتمدد دون أن يؤثر هذا على الأداء ودون أن يسبب اختناقاً في الموارد.

التوسع يمكن أن يكون في : زيادة عدد مستخدمي النظام. إضافة موارد جديدة بسهولة ويسر.

عدم تأثير بعد المورد عن المستخدم.

6.4 التباينية

يمكن بناء نظام موزع من عتاد حاسب وعتاد شبكات مختلفة وبرامج و نظم تشغيل متباينة ، ولا يؤثر هذا في أداء النظام ، مثلا النظم المبنية على الويب تمكن المتصفح من الوصول إليها من أي حاسب بغض النظر عن:

1. نوع حاسب العميل (المستخدم).- نوعية نظام التشغيل الذي يعمل به.- نوع الشبكة المستخدمة.- نوع الخادم الذي يخزن به الموقع.-نوعية نظام التشغيل بالخادم.

7.4 الحماية والتأمين

لحماية الموارد المشتركة والمعلومات المهمة المتبادلة بين أجزاء النظام لابد من وسيلة تأمين للنظام مثل الموثوقية والتشفير وغيرها من وسائل الحماية.

8.4 تجاوز الأعطال

أحد الصفات التي تميز نظام الحاسب الفردي عن النظام الموزع هو العطل الجزئي (partial failure). ويحدث العطل الجزئي عند ما يتعطل مكون من مكونات النظام الموزع. قد يؤثر هذا العطل في أداء بقية مكونات النظام الموزع. عند تصميم النظام الموزع علينا أن نجعله قادراً على العمل في حالة حدوث عطل جزئي به حيث لا يؤثر هذا العطل على أداء النظام ككل. أي أنه إذا كان بمقدور النظام العمل والاستمرارية حتى و لو تعطلت أجزاء منه (في العتاد أو في الشبكات أو في البرامج) ، فهذا يعني أن النظام لديه خاصية تجاوز الأعطال.

9.4 الانفتاحية

يجب على النظام الموزع أن يوفر خدمات وفق أسس ومعايير تحدد صيغة ومعاني هذه الخدمات، مثل البروتوكولات في شبكات الحاسوب حيث تكون عبارة عن قوانين تحكم شكل ومحتوى ومعنى الرسائل في الشبكات.

10.4 التزامنية

يجب تصميم النظام بحيث يسمح للمستخدمين بالوصول للموارد المشتركة من عتاد و بيانات تزامنيا بأمان (دون أن يحتكر تطبيق مورد ويحجزه عن بقية التطبيقات).

ما المجالات التي نستخدم فيها النظام الموزع؟

1. كل المجالات التي تستخدم فيها الشبكات هي محل استخدام النظم الموزعة، مثل: التشارك في البيانات، الاستفادة من الأجهزة الموجودة والتشارك فيها خاصة التي تكلف كثيرا ويصعب توفرها لكل المستخدمين.
2. بيئة خصبة للتعاون والمشاركة في كل شيء مثل المشاركة في الملفات، تبادل الرسائل والمعلومات، الحجز في خطوط الطيران، السحب من حسابك بالبنك.
3. يمكن نقل جزء من البرامج والعمليات التي يزدحم بها جهاز ما إلى جهاز آخر سريع أو غير مزدحم فيما يسمى ترحيل العملية (process migration).
4. الويب التي تمثل أكبر نظام موزع على البسيطة.
5. الحوسبة المتوازية (parallel processing) تمكننا من استخدام الحاسبات في تسريع إنجاز عمل ما عن طريق تقسيم هذا العمل إلى أجزاء، وتوزيع كل جزئية في حاسب ليتم العمل في كل الأجزاء في وقت واحد (تتعاون الحاسبات مع بعضها في إنجاز عمل واحدٍ فيما يشبه النفير (الفزع).

مخاطر الأنظمة الموزعة

1. الاعتمادية (reliability) أو استمرارية عمل الأجهزة غير مضمون مئة بالمئة.
2. ازدحام الشبكة بزيادة عدد المستخدمين قد يؤدي إلى بطء عمل النظام ككل.
3. كثرة الطلبات على الجهاز (الجهاز الرئيس server) قد تؤدي إلى اختناقه وقد يصبح بطيئاً جداً وربما يتوقف.
4. نحتاج دوماً للحماية والسرية في النظام الموزع لأنه مبني على نظام الشبكات. حيث لا يمكن الوثوق بالاتصالات التي تتم بين أجزاء النظام الموزع.

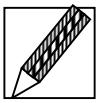
يقال إنه لا يمكنك ضمان سلامة معلوماتك إلا في حالة واحدة وهي ألا يكون لديك كيبل متصل بحاسوبك ما عدا كيبل واحد هو الكيبل الذي يمد الحاسب بالتيار الكهربائي.

أسئلة تقويم ذاتي



اذكر خمساً من صفات النظم الموزعة.
ماذا نعني بقابلية النظام للتوسع ؟

تدريب (2)



أجب بلا أو نعم:

1. كل المجالات التي تستخدم فيها الشبكات هي محل استخدام النظم الموزعة.
2. يمكن نقل جزء من البرامج والعمليات التي يزدحم بها جهاز ما إلى جهاز آخر سريع أو غير مزدحم فيما يسمى بترحيل العملية (process migration).
3. الحوسبة المتوازية تمكننا من استخدام الحاسبات في تسريع إنجاز عملٍ ما عن طريق تقسيم هذا العمل إلى أجزاء وتوزيع كل جزئية .
4. الاعتمادية هي الحالة التي يكون فيها عمل الأجهزة غير مضمون مئة بالمئة.
5. ازدحام الشبكة بزيادة عدد المستخدمين قد لا يؤدي إلى بطء عمل النظام ككل.
6. كثرة الطلبات على الجهاز/الأجهزة الرئيسية ((server(s)) لا يؤثر على النظم الموزعة .

الخلاصة

عزيزي الدارس، تمكنا في هذه الوحدة من الإجابة عن السؤال الأساسي وهو في ذات الوقت الهدف الأساسي من الوحدة. في القسم الثاني منها تعرفنا على مجالات استخدام الأنظمة الموزعة و أهمية الحوجة إليها. وأما في القسم الثالث ناقشنا الصفات التي يجب أن تتوفر في الأنظمة الموزعة فيها. والقسم الأخير من هذه الوحدة كان عن المخاطر التي تواجهنا في استخدام الأنظمة الموزعة ومن أهمها مشاكل الشبكة: مشاكل ازدحام الشبكة، والمعمارية، والاعتمادية المتوفرة، وعدم توفر الخصوصية والسرية التي تستخدمها النظم الموزعة.

لمحة مسبقة عن الوحدة القادمة

عزيزي الدارس سنتعرف في الوحدة القادمة على الفرق الجوهري بين نوعي العتاد المستخدمين لبناء النظم الموزعة، وكذلك سنبين الحل للاختناق الذي ينشأ نتيجة لاستخدام المعالجات ذات الناقل المشترك. أيضاً سنأخذ أمثلة لأنظمة تعمل بنظام (switched multicomputers)، وكذلك أمثلة لأنظمة تعمل بنظام الناقل المشترك.

وفي نهاية الوحدة سنتعرف على كيفية معالجة التباين بين الأجهزة.

عزيزي الدارس سنكون على شوق معك في الانتقال للوحدة التالية لمعرفة المزيد والمفيد عن الأنظمة الموزعة.

إجابات التدريبات

تدريب رقم (1)

الرقم	1	2	3	4	5	6	7	8	9	10
الاجابة	نعم	نعم	لا	لا	نعم	لا	نعم	نعم	نعم	نعم

تدريب رقم (2)

رقم السؤال	1	2	3	4	5	6
الإجابة	نعم	نعم	نعم	لا	لا	لا

مسرد المصطلحات

النظام الموزع عتاداً

مجموعة من الحاسبات المنفصلة عن بعضها البعض، المتحدة مع بعضها البعض لتكوّن حاسباً واحداً كبيراً افتراضياً .

النظام الموزع برامجاً

مجموعة من البرامج الصغيرة المنتشرة في الشبكة والتي تعمل معاً لتكوّن نظاماً برمجياً موحداً يعمل على مشكلة واحدة.

الشفافية أو إخفاء التفاصيل transparency

يجب أن يكون النظام الموزع قادراً على إخفاء تفاصيل حقيقة أن أجزاءه و مصادره موزعة في حاسبات عديدة، فيبدو النظام الموزع للمستخدم كأنه نظام واحد على حاسب واحد.

الاعتمادية reliability

الاعتمادية هي مقدرة النظام على الاستمرار في كل الظروف، وغالباً هذه الخاصية لا تتحقق إلا بخاصية أخرى هي التكرار (replication).

التوسعية scalability

أحد أهم أهداف مطوري النظم الموزعة هو قابلية النظام للتوسع والتمدد دون أن يؤثر هذا على الأداء ودون أن يسبب اختناقاً في الموارد.

الحماية والسرية security

لحماية الموارد المشتركة والمعلومات المهمة المتبادلة بين أجزاء النظام لابد من وسيلة تأمين للنظام مثل الموثوقية والتشفير وغيرها من وسائل الحماية.

تجاوز الأعطال failure handling

أحد الصفات التي تميز نظام الحاسب الفردي عن النظام الموزع هو العطل الجزئي (partial failure). ويحدث العطل الجزئي عند ما يتعطل مكون من مكونات النظام الموزع. قد يؤثر هذا العطل في أداء بقية مكونات النظام الموزع.

الانفتاحية openness

يجب على النظام الموزع أن يوفر خدماتٍ وفق أسس ومعايير تحدد صيغة ومعاني هذه الخدمات، مثل البروتوكولات في شبكات الحاسوب حيث تكون عبارة عن قوانين تحكم شكل ومحتوى ومعنى الرسائل في الشبكات.

التزامنية concurrency

يجب تصميم النظام بحيث يسمح للمستخدمين بالوصول للموارد المشتركة من عتاد و بيانات تزامنيا بأمان (دون أن يحتكر تطبيق مورد ويحجزه عن بقية التطبيقات).

المراجع

1- .Coulouris, G. et al., (2004), Distributed Systems, Concepts and Design, Addison Wesley

2- Farley, J. (1998), Java Distributed Computing, O'reilly & Associates

3- Fedorov, A, (2002), A Programmer Guide to .NET, Addison Wesley

4- Ince, D. (2002), Developing Distributed and E-Commerce Applications, Pearson Education

5- Liu, M. (2004), Distributed Computing Principles and Applications, Pearson Education

6- Schonauer, W. (1998), Scientific Supercomputing: Architecture and use of Shared and Distributed Memory Parallel Computers, Willi Schonauer, Germany

7- Tanenbaum, A., (2002), The Amoeba Distributed Operating System, Vrije Universiteit Technical Report, URL ftp://ftp.cs.vu.nl/pub/amoeba/Intro.ps.Z

8- Tanenbaum , A. AND Van Steen, M. (2002), Distributed Systems Principles and Paradigms, Prentice Hall

9- Tanenbaum, A. (1995), Distributed Operating Systems, 1/e, Prentice Hall, USA

10- Osman, A., (may 2006), A Multi-level Web-based Parallel Processing System, TRANSACTIONS ON ENGINEERING, COMPUTING AND TECHNOLOGY VOLUME 13 MAY 2006 ISSN 1305-5313

11-Forouzan, A. (2007), Data Communications and Networking, Mc Graw Hill, Fourth Edition

12-Tanenbaum, A., (2001), Modern Operating Systems, Prentice Hall, Second Edition

13-Tanenbaum, A., (1999), Structred Computer Organization, Prentice Hall, Fourth Edition


الوحدة الثالثة
معمارية النظام الموزع

محتويات الوحدة

الموضوع	الصفحة
المقدمة	51
تمهيد	51
أهداف الوحدة	**52**
1. طرق توصيل الأجهزة في النظم الموزعة	**53**
2. النظام متعدد المعالجات	55
1.2 عيوب نظام المعالجات المتعددة	56
3. الأنظمة متعددة الحاسبات المتجانس	58
4.الأنظمة متعددة الحاسبات المتباينة	62
الخلاصة	64
لمحة مسبقة عن الوحدة القادمة	65
مسرد المصطلحات	66
المراجع	67

المقدمة

تمهيد

عزيزي الدارس،

مرحبا بك في الوحدة الثالثة من المقرر التي سنتعرف فيها على العتاد المستخدم لبناء النظم الموزعة من أجهزة حواسب منفصلة، وكذلك نتناول طريقتين لتوصيل هذه الحاسبات مع بعضها البعض، طريقة الذاكرة المشتركة وتسمى النظم متعددة المعالجات **multiprocessors**.

وطريقة الذواكر الخاصة وتسمى النظم متعددة الحاسبات **multicomputers**.

وسنتعرف على الفرق الأساسي بين طرق التوصيل للنظم متعددة المعالجات ومتعددة الحاسبات.

وفي القسم الثاني سنتاول الأنظمة متعددة الحاسبات المتجانسة وكيف أن الاجهزة تكون مستقلة عن بعضها البعض، وكذلك سنتاول أمثلة لهذا النعوع من الانظمة.

وفي القسم الأخير سنتعرف على الأنظمة متعددة الحاسبات المتباينة من حيث: نوع المعالج، حجم الذاكرة، اتساع مخارج الدخل والخرج. وقد يكون من عناصر هذا النظام جهاز متعدد المعالجات **(multiprocessor system)**.

أيضاً قد تتصل أجزاء هذا النظام بشبكات متباينة ونظم تشغيل متباينة.

عزيزي الدارس نتمنى أن تستفيد من هذه الوحدة وتساعدنا في نقدها وتصحيحها

أهداف الوحدة



عزيزي الدارس بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

1. تفرق بين نوعي العتاد المستخدمين لبناء النظم الموزعة .
2. تعرف الحل للاختناق الذي ينشأ نتيجة لاستخدام المعالجات ذات الناقل المشترك.
3. تذكر الأنظمة التي تعمل بنظام الناقل المشترك.
4. تبين كيف تتم معالجة التباين بين الأجهزة .

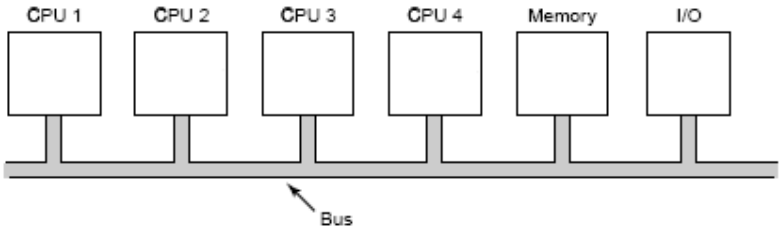
1. طرق توصيل الأجهزة في النظم الموزعة

عزيزي الدارس، في السابق **كانت النظم تعمل مركزياً** (centralized system) وهي **نظم المعالج الواحد** (single processor system) وهذه غالباً ما تكون في حاسب واحد وعلى الأكثر يكون مرتبطاً مع بعض الطرفيات، حيث يخدم هذا الحاسب مجموعة من المستخدمين ، ثم انتقلنا إلى عالم الحاسب الشخصي حيث أصبح بمقدور كل فرد استخدام حاسب لوحده، و أخيراً وليس آخراً أصبح الشخص الواحد يستطيع أن يتعامل مع مجموعة من الحاسبات تقوم بخدمته كأنها جهاز واحد.في تعريفنا السابق للنظام الموزع ذكرنا أنه عبارة عن حاسبات منفصلة (independent) عن بعضها البعض و متصلة (connected) ببعضها البعض.

هذا يعني أن النظام الموزع يعمل على أكثر من حاسب، و تحتاج هذه الحاسبات لطريقة لتتصل مع بعضها البعض، أي أن الحاسبات هي العناصر التي يتألف منها النظام الموزع ، بينما الاتصالات تعتبر الوسيلة الضرورية لربط هذه العناصر لتكوين وحدة واحدة متماسكة. بافتراض أن النظم الموزعة تتألف من أجهزة حواسب منفصلة، هنالك طريقتان لتوصيل هذه الحاسبات مع بعضها البعض، هما:

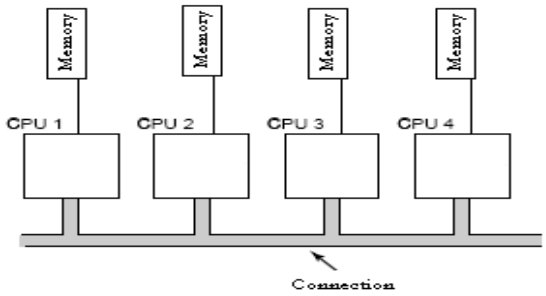
1. **باستخدام ذاكرة مشتركة وتسمى النظم متعددة المعالجات** multiprocessors.
2. باستخدام ذواكر خاصة وتسمى النظم متعددة الحاسبات multicomputers .

الفرق الأساسي بين الاثنين هو أنه في الأول هنالك عنوان حقيقي (فيزيائي) واحد تتشارك فيه كل المعالجات CPUs، مثلاً إذا كتب معالج القيمة 7 في العنوان 2020 فإن أي معالج آخر سيحصل على ذات القيمة إذا أحضر محتوى الذاكرة ذات العنوان 2020.



شكل رقم (3–1):معالجات تتشارك في ذاكرة واحدة عبر ناقل

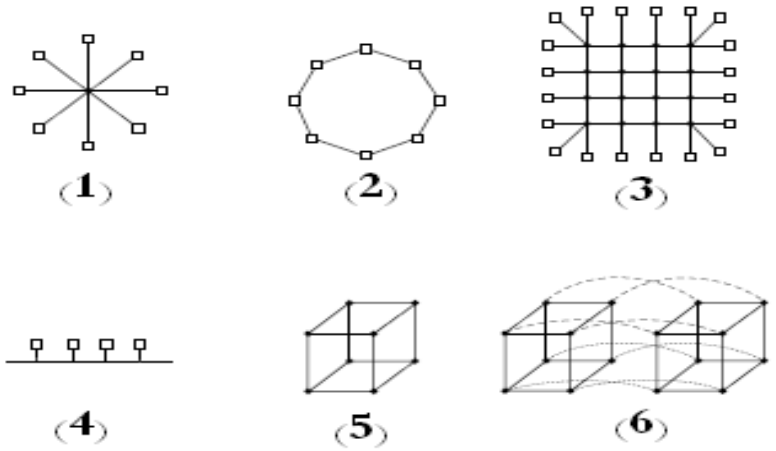
أما في النظام الثاني فلكل جهاز ذاكرته الخاصة، أي إذا كتب معالج ما قيمة في العنوان رقم 2020، فلن يحصل معالج آخر بالنظام على هذه القيمة إذا نظر في 2020. ذلك لأن العنوان 2020 للمعالج الأول هو بذاكرة المعالج الأول بينما العنوان 2020 للمعالج الثاني هو في ذاكرة المعالج الثاني.



شكل رقم (3–2): حاسبات متعددة (كل حاسب بذاكرة خاصة) تتصل برابط

النوعان أعلاه يمكن ربطهما بطرق مختلفة مثل:

(3) في شكل شبكي **(grid)**	(2) في شكل حلقة **(ring)**	(1) عن طريق مفتاح **switch**
(6)مكعب مركب**(hypercube)**	(5) مكعب **(cube)**	(4) في شكل ناقل **(bus)**



شكل رقم (3-3): أشكال ربط مختلفة

2. النظام متعدد المعالجات multiprocessor

عزيزي الدارس، عبارة عن مجموعة من المعالجات تتشارك في ذاكرة واحدة بحيث يكون بمقدور أي معالج الوصول مباشرة لهذه الذاكرة المشتركة والتعامل معها.

قد ترتبط هذه المعالجات عن طريق ناقل مشترك (bus)، ترتبط على نفس الناقل الذاكرة المشتركة. الناقل المشترك قد يكون شبكة أو ناقلاً في لوحة رئيسية (motherboard) تركب فيها كل المعالجات والذاكرة المشتركة.

وبما أن الذاكرة المشتركة واحدة، فإن كل معالج يضع قيمة في موقع ما بالذاكرة، تستطيع بقية المعالجات الوصول لتلك القيمة والتعامل معها من قراءة أو تعديل أو غيرها.

الذاكرة التي تمتلك هذه الخاصية نسميها (coherent).

المشكلة الرئيسية في هذا النوع **هو أن زيادة عدد المعالجات يخلق اختناقاً على الخط الناقل مما يؤثر على الأداء بصورة واضحة جداً**

إذن فما الحل؟

الحل هو وضع ذاكرة مخبأة (cache memory) بين الخط الناقل والمعالج، هذه الذاكرة المخبأة توضع بها البيانات الحديثة التي أخذت من الذاكرة الرئيسية، و بالتالي عند ما يطلب المعالج معلومة سيأخذها من هذه الذاكرة المخبأة إن وجدت فيها، وإذا لم توجد بها فسيضطر لاستخدام الخط الناقل لإحضار المعلومة من الذاكرة الرئيسية.

هذه الطريقة **قللت كثيراً من الزحام** الذي كان موجودا بالناقل الرئيسي. ولكن هل هذا الحل حل نهائي؟ لالقد ولّد هذا الحل مشكلة أخرى هي عدم توافقية (consistency problem) البيانات.

افترض أن معالجين قاما بنسخ قيمة من الذاكرة الرئيسية إلى ذاكرتيهما المخبأتين، ثم قام المعالج الأول بتعديل هذه القيمة بالذاكرة الرئيسية، عند ما يطلب المعالج الثاني هذه القيمة سيحصل عليها من ذاكرته المخبأة (لن يأخذ القيمة الحديثة المعدلة بواسطة المعالج الأول)، هنا يظهر عدم توافقية القيم.

هذا يقود لذاكرة غير متماسكة (incoherent)، حيث يصعب برمجة النظام مع وجود هذه المشكلة.

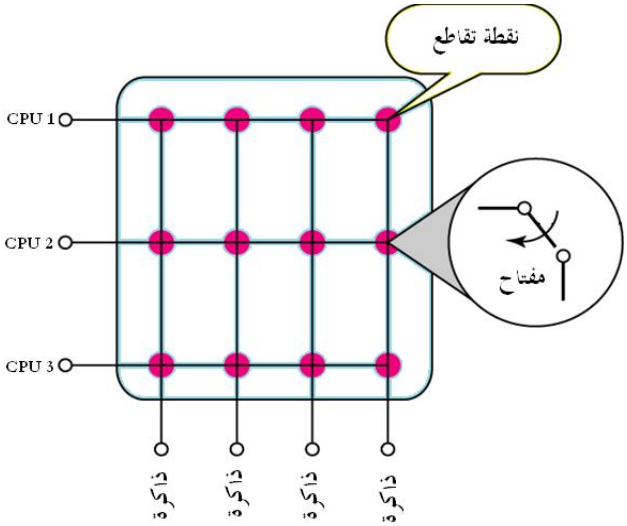
عملية التخبئة (caching) تستخدم بكثرة في النظام الموزع ولابد من التعامل معها والتحايل عليها كما سنرى لاحقاً .

1.2 عيوب نظام المعالجات المتعددة

عدم قدرته على التوسع (ليس به خاصية قابلية التوسع scalability) حتى مع استخدام الذاكرة المخبأة، فإذا أردنا بناء نظام متعدد المعالجات لأكثر من 256 معالجاً فلا بد من طريقة مختلفة لربط المعالجات مع الذاكرة الرئيسية المشتركة، لأن الخط الناقل المشترك سيزدحم ببيانات المعالجات وكثرتها تسبب اختناقاً فيه.

أحد الحلول

هو **تقسيم الذاكرة الرئيسية المشتركة** إلى ذواكر كثيرة متعددة وربط كل مجموعة من المعالجات مع واحدة من هذه الذواكر بطريقة مفاتيح متقاطعة (crosspoint switching) كما في الشكل ادناه.

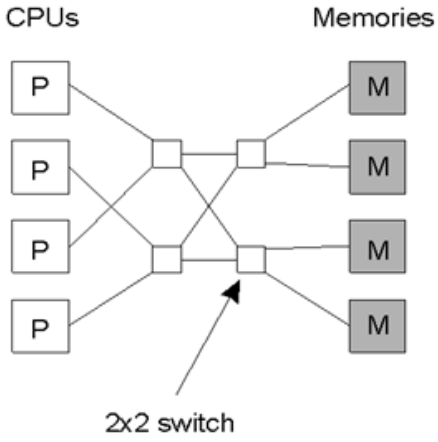


شكل رقم (3-4): مفاتيح تقاطع لتوصيل المعالجات مع الذواكر

كل مفتاح تقاطع (crosspoint switch) هو عبارة عن مفتاح يمكن فتحه و غلقه بواسطة جهاز معين. حيث يتم غلق هذا المفتاح في حالة طلب المعالج الوصول للذاكرة.

من مزايا هذا النظام أنه يسمح لأكثر من معالج أن يتعامل مع أكثر من ذاكرة في نفس الوقت، و لا ينتظر معالج معالجاً آخر للوصول لذاكرة معينة كما في الطريقة الأولى إلا في حالة طلب معالجين الوصول لذاكرة معينة في نفس الوقت، في هذه الحالة على معالج أن ينتظر الآخر حتى يفرغ من استخدام هذه الذاكرة.

عيوب هذه النوع أنه إذا أردت بناء مفاتيح لعدد "ن " معالج لربطها مع عدد "ن" ذاكرة مشتركة فإنني أحتاج إلى ن×ن مفتاح للتوصيل وهذا كثير. الحل هو طريقة توصيل مختلفة تسمى مفتاح المقاطعات متعدد المستويات (omega network) كما في الشكل (3-5) حيث يقل عدد المفاتيح كثيراً .



شكل رقم (3-5): مفاتيح مقاطعات متعددة المستويات

3. الأنظمة متعددة الحاسبات المتجانسة

(Homogenous Multicomputer Systems)

عزيزي الدارس، مقارنة بتعدد المعالجات ، يعتبر بناء نظام متعدد الحاسبات أسهل، حيث لكل حاسب ذاكرته المحلية المرتبطة معه مباشرة. قد تكون هنالك مشكلة واحدة فقط هي كيف سيتم تبادل البيانات بين الحاسبات.

هذه الأجهزة عادة ما تكون مستقلة عن بعضها البعض، لذلك لابد من رابط بينها يصل بعضها ببعض. هذا الرابط أو الشبكة لابد أن تكون جيدة الأداء. قد يكون الرابط ناقلاً مشتركاً تتصل عبره كل الأجهزة مثل الاثيرنت السريع (Fast Ethernet)، الذي تصل سرعته 100 ميغابت في الثانية. استخدام الناقل (bus) سريع ولكنه غير قابل للتوسع ، فكلما زاد عدد الحاسبات في الشبكة قل الأداء. السبب في قلة الأداء هي طريقة عمل شكل الناقل (bus) حيث يتم إذاعة الرسالة في الشبكة، وهذه الطريقة تجعل حاسباً واحداً فقط يستخدم الناقل في اللحظة المعينة. تخيل طوابير الحاسبات التي تنتظر دورها للوصول إلى الناقل.

من هنا نستنتج أنه إذا كان لدينا نظام به عدد حاسبات كبير (يفوق المائة جهاز مثلاً)، فلا بد من تغيير شكل الشبكة (network topology) من شكل الناقل (bus) إلى شكل آخر يتحمل عدداً كبيراً من الحاسبات.

الشكل الأفضل للشبكة هنا هو النظام المبني على الرابط المفتاحي (switch). حيث بدلاً من إذاعة الرسالة يتم توجيهها (routing).

أسئلة تقويم ذاتي

1. اذكر الفرق الجوهري بين نوعي العتاد المستخدمين لبناء النظم الموزعة.
2. ما الحل للاختناق الذي ينشأ نتيجة لاستخدام المعالجات ذات الناقل المشترك؟

1.3 الحاسبات التي تعمل بنظام (switched multicomputers)

1.1.3 Massively parallel processors (MPPs)

حيث تعتبر **COWS** حاسبات سوبر ضخمة تتألف من آلاف المعالجات التي لا تختلف كثيراً عن معالجات الحواسيب الشخصية، غير أن الرابط بين المعالجات يكون عالي الجودة والتصميم، بحيث لا يكون هنالك بطء به (low latency)، و أن يكون متسعاً بما فيه كفاية (high bandwidth).

مثال

الحاسب العملاق Blue Gene/L يعتبر من أسرع الحاسابات التي ظهرت في نوفمبر 2007. حيث يتكون من 65,536 حاسب مرتبطة بتوصيلات فائقة السرعة، وكل حاسب به معالجان (يستطيع أن يقوم بمهمتين في وقتٍ واحد)، أو بمعنى آخر

2×65,536، انظر الشكل (3-6).



شكل رقم (3-6): حاسب عملاق (super computer) **يسمى** Blue Gene/L

Clusters of workstations (COWs) 2.1.3

هذا النوع عبارة عن حواسيب شخصية عادية مرتبطة بعضها ببعض بشبكة سريعة مثل (Myrinet). تتميز MPPs على COWs برابط الاتصال، فأداء الرابط في النوع الأول عالٍ جداً مقارنة مع رابط النوع الثاني (رغم أن النوع الثاني يعتبر عالياً مقارنة مع الشبكات العادية مثل الإيثرنت السريع (fast Ethernet)).

تعتبر COWs أقل كلفة من MPPs، لذلك يمكننا القول إن الأداء له ثمنه، فالذي لا يستطيع امتلاك MPPs يمكنه استخدام COWs، ويمكن تقليل تكلفة COWs أكثر إذا استخدمنا شبكة إيثرنت لأنها أقل بكثير من Myrinet (وكذلك أبطأ بكثير).



شكل رقم (3-7): COWs تستخدم الإيثرنت السريع بجامعة السودان للعلوم والتكنولوجيا



شكل رقم (3-8): COWs تستخدم Myrinet بالجامعة المفتوحة بهولندا

4. الأنظمة متعددة الحاسبات المتباينة

(Heterogeneous Multicomputer Systems)

تعتمد الأنظمة الموزعة في عملها غالباً على حواسيب متباينة من حيث نوع المعالج ، حجم الذاكرة، اتساع مخارج الدخل والخرج، وقد يكون من عناصر هذا النظام جهاز متعدد المعالجات (multiprocessor system).

أيضاً قد تتصل أجزاء هذا النظام بشبكات متباينة ونظم تشغيل متباينة.

في هذا النظام المتباين قد لا تستطيع التطبيقات افتراض نظام واحد متماسك ، فليس هنالك ضمانات بوجود خدمات متشابهة في كل أجزاء النظام وكذلك متوقع أن يختلف الأداء بين كل جزء وآخر من النظام. لذلك لابد من طبقة برمجية تهتم بهذا التباين و تعالجه بحيث نوفر على المطور جهد التعامل مع التباين، ونجعله يركز كل جهده على تطوير التطبيقات التي تستفيد من النظام الموزع. هذه الطبقة تخفي هذا التباين عن التطبيقات وتتعامل هي معه في الخفاء فيما يسمى الشفافية (transparency).

أسئلة تقويم ذاتي



اذكر أمثلة لأنظمة تعمل بنظام (switched multicomputers).

اذكر أمثلة لأنظمة تعمل بنظام الناقل المشترك.

كيف تتم معالجة التباين بين الأجهزة ؟

الخلاصة

عزيزي الدارس، **لقد عرفنا في هذه الوحدة أن النظام الموزع يعمل على أكثر من حاسب،** و تحتاج هذه الحاسبات لطريقة لتتصل مع بعضها البعض، أي أن الحاسبات هي العناصر التي يتألف منها النظام الموزع، بينما الاتصالات تعتبر الوسيلة الضرورية لربط هذه العناصر لتكوين وحدة واحدة متماسكة. أيضا لقد بينا في هذه الوحدة أن النظم الموزعة تتألف من أجهزة حواسب منفصلة، وتعرفنا إلى طرق توصيل هذه الحاسبات مع بعضها البعض،ومن أهم هذه الطرق **استخدام ذاكرة مشتركة وتسمى النظم متعددة المعالجات** multiprocessors.

استخدام ذواكر خاصة وتسمى النظم متعددة الحاسبات multicomputers.

لمحة مسبقة عن الوحدة القادمة

عزيزي الدارس في الوحدة التالية سنتناول النظام الموزع و الطبقات البرمجية التي توفر للمستخدم بيئة سهلة لبناء تطبيقات موزعة. وسنتعرف على أنواع نظام التشغيل الموزع بناءاً على العتاد ونفرق بين نظام التشغيل أحادي المعالجات ونظام التشغيل متعدد المعالجات

يمكن أن يكون النظام الموزع نظام تشغيل قائمٍ بذاته، فهو هنا يقوم بدور نظام التشغيل العادي بالإضافة لدوره كنظام موزع يوفر الشفافية للتطبيقات الموزعة للتعامل مع العتاد كأنه جهاز واحد.

وقد ينبني النظام الموزع كطبقة فوق نظام تشغيل، وهو بهذا يكمل النقص الموجود بنظام التشغيل ويوفر الاثنان معاً بيئة سهلة لبناء تطبيقات موزعة.

مسرد المصطلحات

النظام متعدد المعالجات **multiprocessor**

عبارة عن مجموعة من المعالجات تتشارك في ذاكرة واحدة بحيث يكون بمقدور أي معالج الوصول مباشرة لهذه الذاكرة المشتركة والتعامل معها.

الأنظمة متعددة الحاسبات المتجانسة(Homogenous Multicomputer Systems)

مقارنة بتعدد المعالجات ، يعتبر بناء نظام متعدد الحاسبات أسهل، حيث لكل حاسب ذاكرته المحلية المرتبطة معه مباشرة. قد تكون هنالك مشكلة واحدة فقط هي كيف سيتم تبادل البيانات بين الحاسبات.

حاسبات سوبر **COWS**

وهي حاسبات ضخمة تتألف من آلاف المعالجات التي لا تختلف كثيراً عن معالجات الحواسيب الشخصية، غير أن الرابط بين المعالجات يكون عالي الجودة والتصميم، بحيث لا يكون هنالك بطء به (low latency) و أن يكون متسعاً بما فيه كفاية (high bandwidth).

الحاسب العملاق Blue Gene/L

يعتبر من أسرع الحاسابات التي ظهرت في نوفمبر 2007. حيث يتكون من 65,536 حاسب مرتبطة بتوصيلات فائقة السرعة، وكل حاسب به معالجان (يستطيع أن يقوم بمهمتين في وقتٍ واحد)، أو بمعنى آخر 65,536×2.

المراجع :

1- 1.Coulouris, G. et al., (2004), Distributed Systems, Concepts and Design, Addison Wesley

2- Farley, J. (1998), Java Distributed Computing, O'reilly & Associates

3- Fedorov, A, (2002), A Programmer Guide to .NET, Addison Wesley

4- Ince, D. (2002), Developing Distributed and E-Commerce Applications, Pearson Education

5- Liu, M. (2004), Distributed Computing Principles and Applications, Pearson Education

6- Schonauer, W. (1998), Scientific Supercomputing: Architecture and use of Shared and Distributed Memory Parallel Computers, Willi Schonauer, Germany

7- Tanenbaum, A., (2002), The Amoeba Distributed Operating System, Vrije Universiteit Technical Report, URL ftp://ftp.cs.vu.nl/pub/amoeba/Intro.ps.Z

8- Tanenbaum , A. AND Van Steen, M. (2002), Distributed Systems Principles and Paradigms, Prentice Hall

9- Tanenbaum, A. (1995), Distributed Operating Systems, 1/e, Prentice Hall, USA

10- Osman, A., (may 2006), A Multi-level Web-based Parallel Processing System, TRANSACTIONS ON ENGINEERING, COMPUTING AND TECHNOLOGY VOLUME 13 MAY 2006 ISSN 1305-5313

11-Forouzan, A. (2007), Data Communications and Networking, Mc Graw Hill, Fourth Edition

12-Tanenbaum, A., (2001), Modern Operating Systems, Prentice Hall, Second Edition

13-Tanenbaum, A., (1999), Structred Computer Organization, Prentice Hall, Fourth Edition


الوحدة الرابعة
برمجيات النظم الموزعة

محتويات الوحدة

الموضوع	الصفحة
المقدمة	**71**
تمهيد	**71**
أهداف الوحدة	**71**
1. تعريف برمجيات النظم الموزعة	**72**
2. من مهام برمجيات النظام الموزع	**73**
3. أنواع نظم التشغيل	**73**
4. هيكل نظام التشغيل:	**77**
1.4 النواة Kernel	**77**
2.4 مساحة المستخدم (الغلاف) shell	77
5. أنظمة الذاكرة المشتركة الموزعة (DSM)	87
الخلاصة	**100**
لمحة مسبقة عن الوحدة القادمة	**101**
مسرد المصطلحات	**102**
المراجع	**104**

المقدمة

تمهيد

عزيزي الدارس، مرحبا بك في الوحدة الرابعة من مقرر "النظم الموزعة" والتي تبحث في برمجيات النطم الموزعة.

في القسم الأول النظام الموزع هو طبقات برمجية توفر للمستخدم بيئة سهلة لبناء تطبيقات موزعة. يمكن أن يكون النظام الموزع نظام تشغيل قائمٍ بذاته، فهو هنا يقوم بدور نظام التشغيل العادي بالإضافة لدوره كنظام موزع يوفر الشفافية للتطبيقات الموزعة للتعامل مع العتاد كأنه جهاز واحد. في القسم الثاني سنتعرف على مهام برمجيات النظام الموزع.

وفي القسم الثالث نبين **أنواع نظم التشغيل** و**هدف نظم التشغيل الأساسي**. أما القسم الرابع فيتناول **هيكل نظام التشغيل** ومكوناته وبعدها نعرض مشاكل أنظمة التشغيل المتعددة.

أهداف الوحدة

عزيزي الدارس بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

1. تعرف نداء الإجراء البعيد **(RPC)**.
2. توضَّح المهمة الأساسية للطبقة الوسيطة.
3. تشرح مهام برمجيات النظم الموزعة.
4. تحلل نظام الملفات الموزع.

1. تعريف برمجيات النظم الموزعة

عزيزي الدارس، يتضح شكل أي نظام من خلال برمجياته و يظل العتاد خلف الكواليس كالجندي المجهول يعمل بلا كلل ولا ملل. نظم التشغيل لا تشذ عن هذه القاعدة فهي توفر الواجهة التي من خلالها نستخدم العتاد، فهو وسيط بيننا وبين الحاسب

قاعدة أخرى تقول إنه كلما تعقدت البرمجيات وُبذِل فيها الجهد الكبير كان الناتج برمجيات متميزة وسهلة الاستخدام.

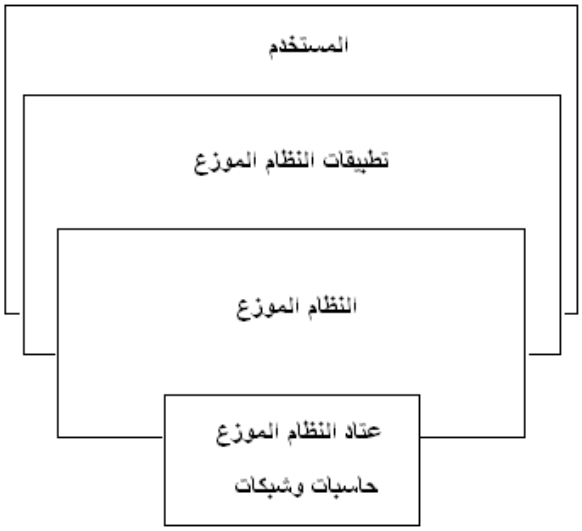
وكذلك الجهد المبذول لبناء نظم موزعة فهو لجعل بناء التطبيقات الموزعة سهلاً جداً . فسهولة الاستخدام هدف أساسي وراء تصميم النظم الموزعة وكل برنامج. أيضاً **الشفافية** التي **نخفي بها تباين وتعقيدات العتاد** (من أجهزة وشبكات) هي هدف يساعد في إزالة صعوبة وتعقيدات التعامل مع الأجهزة والشبكات.

بناء تطبيق موزع من غير وجود نظام موزع، يشبه إلى حدٍ ما استخدام الحاسب من غير نظام تشغيل.

النظام الموزع هو طبقات برمجية توفر للمستخدم بيئة سهلة لبناء تطبيقات موزعة.

يمكن أن يكون النظام الموزع نظام تشغيل قائمٍ بذاته، فهو هنا يقوم بدور نظام التشغيل العادي بالإضافة لدوره كنظام موزع يوفر الشفافية للتطبيقات الموزعة للتعامل مع العتاد كأنه جهاز واحد.

وقد ينبني النظام الموزع كطبقة فوق نظام تشغيل، وهو بهذا يكمل النقص الموجود بنظام التشغيل. ويوفر الاثنان معاً بيئة سهلة لبناء تطبيقات موزعة.



شكل رقم 104

2. من مهام برمجيات النظام الموزع:

- **إدارة عتاد النظام الموزع** بتمكين المستخدمين وتطبيقاتهم من التشارك في الموارد (resources)، مثل المعالجات، الذواكر، الأجهزة الطرفية وعتاد الشبكة. أيضا يوفر النظام التشارك في البيانات بأنواعها.
- إخفاء تفاصيل العتاد وتباينه بتوفير حاسب واحد افتراضي بنظام واحد وليس عدة أجهزة بأنظمة مختلفة.

3. أنواع نظام التشغيل

عزيزي الدارس، ما رأيك لو أهديناك حاسباً بدون ويندوز أكس بي، أو بدون لينكس؟

بالتأكيد سترميه على قارعة الطريق إن لم تجد له نظام تشغيل.

فنظام التشغيل أساس كل شي في الحاسب سواءً كان هذا الحاسب شخصياً، أو مركزياً، أو نقالاً، أو مجموعة حاسبات مرتبطة بشبكة.

لا يستطيع المستخدم العادي التعامل مع الحاسب في غياب نظام التشغيل، وعلى المبرمج الكبير والمهندس القدير بذل الوقت الوفير لتنفيذ برنامج صغير (في غياب نظام التشغيل). إذن **هدف نظم التشغيل الأساسي سهولة استخدام العتاد.**

كذلك لا تستطيع تطبيقاتنا العمل على الحاسب في غياب نظام التشغيل فهو مهم لها كأهميته للمستخدم. فلا التطبيقات ولا المستخدم يستطيع التعامل مع العتاد مباشرة.

و يمكن تقسيم نظم التشغيل إلى نوعين هما:

3. نظام تشغيل معالج واحد uniprocessor operating system (UOS).

ينقسم بدوره إلى نوعين هما :

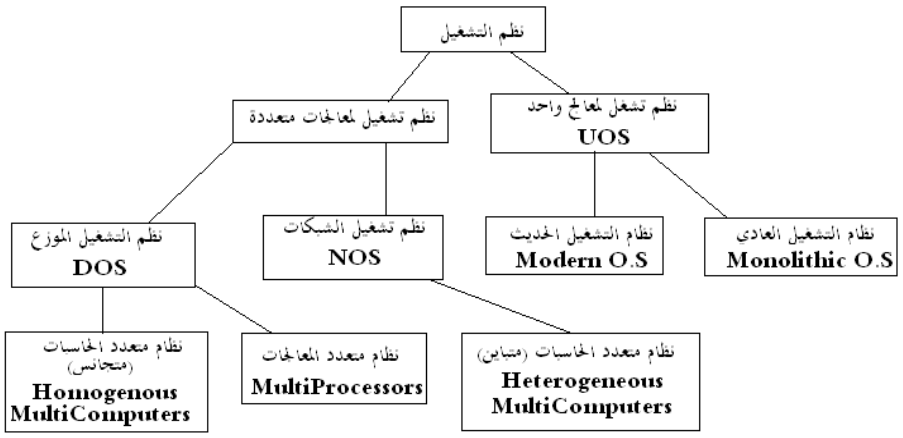
1.3 عادي (monolithic).

2.3 حديث **(modern)**.

2.3 نظام التشغيل الذي يدير عدة معالجات ينقسم إلى:

1.2.3 نظام تشغيل موزع **(DOS)**.

2.2.3 نظام تشغيل شبكات **(NOS)**.



شكل رقم (4-1): أنواع نظم التشغيل

1.2.3 نظم التشغيل الموزعة (Distributed Operating System (DOS)

هذا النوع يدير الموارد لتبدو وكأنها جهاز واحد كبير، فهو بذلك يوفر خاصية الشفافية التي تخفي جزئيات العتاد والبرامج وانتشارها في الشبكة.

ينقسم نظام التشغيل الموزع إلى نوعين بناءً على العتاد الذي يديره:

1.2.3 .1 نظام تشغيل يدير معالجات متعددة (multiprocessors).

1.2.3 .2نظام تشغيل يدير أجهزة حاسوب متعددة متجانسة (homogenous multicomputers).

شبكة الحاسوب قد تربط أجهزة حاسوب:
متشابهة في العتاد والبرامج (متجانسة).
مختلفة في العتاد /أو مختلفة في البرامج (متباينة). متعدد المعالجات.

طريقة إدارة العتاد في نظام التشغيل الموزع تشبه إلى حد كبير تلك التي في نظام التشغيل العادي، لذلك سندرس كيف يقوم نظام التشغيل العادي بذلك، ثم نوضح الفروقات بين النوعين.

2.2.3. نظم تشغيل الشبكات (Network Operating System (NOS)

هذا النوع هو **عبارة عن مجموعة من الأجهزة والموارد المنفصلة** عن بعضها البعض، ولكل جهاز نظام تشغيله الخاص والذي يدير موارده. تتعاون هذه الأجهزة لتوفر خدماتها لبعضها البعض.تعامل هذا النوع من نظم التشغيل مع الحاسبات المتعددة المتباينة (Heterogeneous Multicomputers).

يعمل نظام تشغيل الشبكات بنفس الطريقة التي يعمل بها نظام التشغيل العادي (uniprocessor)، ولكن يختلف عنه في كونه يتيح خدمات الأجهزة المحلية (local services) للأجهزة البعيدة (remote clients).

تفتقر نظم تشغيل الشبكات إلى العديد من الخصائص والخدمات التي يحتاجها النظام الموزع مثل خاصية الشفافية (transparency).

يمكن إضافة هذه الخصائص التي لاتوجد بنظم تشغيل الشبكات والتي تمنعه من دعم عمل النظم الموزعة، وذلك عن طريق جمعها في طبقة برمجية تسمى الطبقة الوسيطة (middleware) ووضعها على نظام تشغيل الشبكات، بهذه الطريقة يصبح النظام قادراً على توفير بيئة مناسبة للنظم الموزعة.

نظم تشغيل المعالج الواحد ((uniprocessor operating system (UOS)

يقوم نظام التشغيل هنا بإدارة عتاد حاسب به معالج واحد. وتوفير بيئة سهلة للمستخدم وتطبيقاته للاستفادة من العتاد بصورة مثلى.

الهدف الأساسي وراء نظام التشغيل هو إدارة عتاد الجهاز والسماح للمستخدمين من الاستفادة القصوى منه والتشارك فيه. تشارك التطبيقات في المورد يتم بتعدد المهام **(multitasking)**. حيث يتم تحويل المورد بين التطبيقات بسرعة (كل تطبيق يأخذ نصيباً زمنياً في استخدام المورد)، وكلما زادت التطبيقات قل نصيب كل تطبيق في استخدام المورد.

من أهم متطلبات مشاركة الموارد هو حماية التطبيقات من بعضها البعض. فمثلاً لو كان لدينا تطبيقان "أ" و "ب"يعملان معاً في نفس الوقت وفي نفس الذاكرة ، فلاُ يسمح للتطبيق "أ" أن يتعدى على بيانات التطبيق "ب" والعكس.

فتعمل التطبيقات ضمن الحدود والصلاحيات التي يوفرها لها نظام التشغيل ولا تتعداها. أيضاً لا يسمح للتطبيق بالوصول مباشرة للعتاد وإنما يتم ذلك عن طريق استدعاء دوال بنظام التشغيل تسمى نداءات النظام **(system calls)**.

إذن فنظام التشغيل سكرتير العتاد فلا تصل التطبيقات للعتاد بدون أخذ مواعيد وموافقة منه.

4. هيكل نظام التشغيل

عزيزي الدارس، نجد هيكل نظام التشغيل مكوناً من طبقتين رئيسيتين هما :

1. قلب نظام التشغيل أو نواته **(Kernel)**.
2. مساحة المستخدم **(User)** أو الغلاف أو الواجهة.

1.4 النواة Kernel

هي البرامج التي تقوم بإدارة العتاد من معالج وذاكرة وطرفيات وأقراص وغيرها. وهي التي تتحمل بالذاكرة الرئيسية عند فتحك للحاسب وهي التي تشغل وتدير باقي التطبيقات. وهي لطبيعة عملها تستطيع الوصول لأي جزء من العتاد وتنفيذ أي أوامر دون أن يمنعها أو يحدها شيء (هنا مسموح بتنفيذ كل أنواع الأوامر بما فيها التي تتعامل مباشرة مع جميع أجزاء الذاكرة وجميع المسجلات).

2.4 مساحة المستخدم (الغلاف) shell

هي الواجهة التي يتعامل معها المستخدم والتطبيقات، ومن خلالها نستطيع تنفيذ تطبيقاتنا، لذلك أحيانا تسمى: واجهة المستخدم، أو وضع المستخدم، أو مساحة المستخدم. وهنا لا يسمح بالوصول المباشر للعتاد، فلا يمكن الوصول للذاكرة التي يقبع فيها نظام التشغيل مثلاً، ولا يمكن التعامل مع كل المسجلات. فيحدد نظام التشغيل الجزء المسموح الوصول إليه من الذاكرة والمسجلات التي يمكن للتطبيقات الوصول إليها (يمنع الوصول مباشرة إلى مسجلات الأجهزة (device registers)).

عند ما ينفذ المعالِج برامج نظم التشغيل يكون في وضع النواة ، ويتغير لوضع المستخدم عند تنفيذ برامج المستخدم.

التحول من وضع النواة إلى وضع المستخدم يتم عبر نداءات النظام (system calls). حيث توفر هذه النداءات خدمات التعامل مع العتاد للتطبيقات.فمثلاً إذا أراد تطبيق قراءة بيانات من ملف مخزن بالقرص، فإنه سينادي دالةً بنظام التشغيل تقوم باحضار المعلومات من القرص.

جعل النواة **تحتوي على كل برامج إدارة العتاد** مثل برامج جدولة المهام ، برامج إدارة الذاكرة ، برامج نظام الملفات والتعامل مع الأقراص وغيرها من البرامج يسبب مشاكل. فالنواة ستكون: O كبيرة. O مستقرة في الذاكرة. O في مكان واحد بالذاكرة.

في هذه الحال، يصبح تعديل جزء من نظام التشغيل صعباً ومعقداً ويتطلب الآتي:

- إجراء التعديل.
- إغلاق الجهاز.
- إعادة ترجمة نظام التشغيل (recompile).
- إعادة تنصيب نظام التشغيل (setup)

هذا النوع من نظام التشغيل نطلق عليه نظام التشغيل الرتيب (monolithic operating system). ويعتبر **غير جيد** لأنه **يفتقر للآتي:**

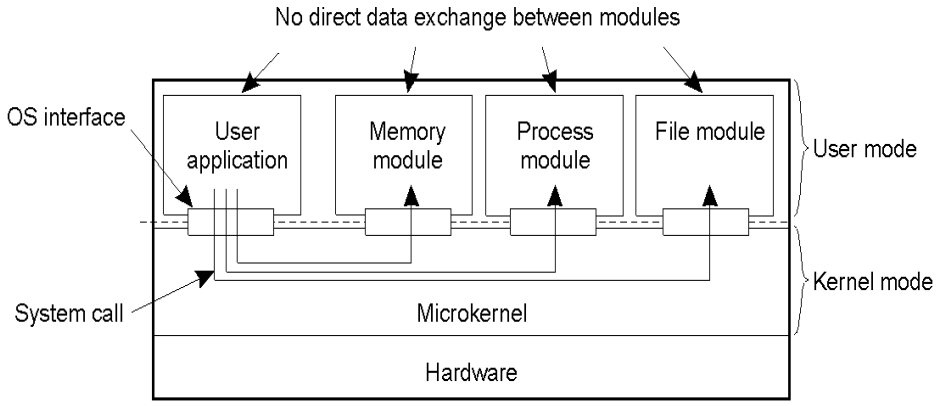
- الانفتاح (openness).
- الاعتمادية (reliability).
- الصيانة (maintainability).

نظام التشغيل الحديث (modern operating system)

يمكن تغيير نظام التشغيل ليصبح أكثر مرونة وذلك بتعديل النواة في نظام التشغيل العادي، فيصبح النظام الحديث كالآتي:

- تحويل البرامج التي تدير العتاد من النواة إلى مساحة المستخدم.
- جعل البرامج الضرورية لنظام التشغيل في النواة.

الآن أصبحت النواة صغيرة الحجم لاتحتاج إلى مساحة كبيرة بالذاكرة، لذلك سميت النواة المصغرة (small microkernel).



شكل رقم (4-2): نظام التشغيل الحديث

1.1.4 فوائد النواة المصغرة (microkernel)

- استخدام النواة المصغرة جعل نظام التشغيل مرناً ومن السهل تعديله واستبداله، ذلك لأن الجزء الكبير من نظام التشغيل ينفذ في وضع المستخدم. بالتالي ليس هنالك حاجة لإعادة تشغيل النظام.
- إمكانية تحويل جزء نظام التشغيل الموجود في وضع المستخدم إلى أجهزة أخرى (portable).
- استخدام النواة المصغرة أضفى على نظام التشغيل الأحادي صفة تمكنه من التوسع ليخدم عدة معالجات بسهولة.

عيوب النواة المصغرة (microkernel)

يكثر الاتصال بين أجزاء نظام التشغيل نتيجة لوجود برامج إدارة العتاد بمساحة المستخدم مما يقلل الكفاءة.

نظم التشغيل متعددة المعالجات (multiprocessor operating system)

يختلف نظام التشغيل هنا عن سابقه في أنه يدير عدة معالجات، وتوضع البيانات التي يحتاجها نظام التشغيل لإدارة العتاد في ذاكرة مشتركة. الجدول التالي يوضح مقارنة بين نظام التشغيل الأحادي ونظام التشغيل المتعدد.

نظام التشغيل أحادي المعالجات	نظام التشغيل متعدد المعالجات
يدير معالج واحد	يدير عدة معالجات
يستخدم المعالج ذاكرة واحدة	كل المعالجات تستخدم ذاكرة مشتركة
بيانات إدارة العتاد توجد في ذاكرته	بيانات إدارة العتاد في الذاكرة المشتركة
التعديل في البيانات يتم بواسطة معالج واحد	تعديل البيانات يتم بأكثر من معالج
ليس هنالك وصول متزامن للبيانات	يوجد وصول متزامن للبيانات

شكل رقم (4-3): مقارنة بين نظام التشغيل الأحادي ونظام التشغيل متعدد المعالجات

الوصول المتزامن للبيانات في المعالجات المتعددة هو إمكانية تعديل البيانات بواسطة أكثر من معالج في وقت واحد مما ينتج عنه أخطاء، لذلك على نظام التشغيل هنا حماية الذاكرة من الوصول المتزامن لضمان توافق البيانات (consistency).

نظم التشغيل الأحادية والتي خصصت للحاسبات الشخصية ليس من السهل إدارتها لنظام ذي عدة معالجات، ذلك لأن برامجها صممت لتنفذ في معالج واحد وبالتالي لن تستفيد من العتاد متعدد المعالجات إذا استخدمت نظام تشغيل أحادي.

نظم التشغيل الحديثة صممت منذ البداية لتدعم عدة معالجات وبرامجها تنفذ بالتوازي في كل المعالجات.

يهدف نظام التشغيل المتعدد المعالجات إلى الآتي:

- رفع الكفاءة بدرجة عالية.
- إخفاء عدد المعالجات عن المستخدم والتطبيقات (توفير الشفافية).

تحقيق الشفافية في هذا النظام سهلة إلى حدٍ ما ، ذلك لأن أجزاء التطبيق المختلفة تتصل ببعضها البعض بنفس الطريقة التي تتصل بها في تعدد المهام بنظم التشغيل الأحادية. أي أن التطبيق الذي يعمل في النظام الأحادي بطريقة تعدد المهام يمكن أن يعمل في نظام تشغيل متعدد المعالجات بنفس الطريقة.

ما الفرق الرئيسي بين تعدد المهام و تعدد المعالجات ؟
في تعدد المهام لاتوجد موازاة حقيقية، فالذي ينفذ البرامج معالج واحد ولكنه يحاول أن يوزع زمنه بين البرامج لتبدو وكأنها تعمل بالتوازي، بينما الحقيقة أن المعالج لا يستطيع تنفيذ أكثر من أمر واحد في نفس الوقت. أما في تعدد المعالجات فتتم موازاة حقيقية حيث يعمل كل برنامج على معالج منفصل (كل أمر يمكن أن ينفذ في معالج في نفس الوقت).

كل البيانات بهذا النظام متوفرة بالذاكرة المشتركة، بالتالي تتم كل الاتصالات عبر هذه الذاكرة المشتركة.

مشكلة النظام المتعدد

بما أن الوصول للبيانات متاح لكل المعالجات، فالمشكلة الوحيدة هي حماية البيانات ضد الوصول المتزامن (simultaneous access).

طرق الحماية ضد الوصول المتزامن

هنالك طريقتان يمكن استخدامهما للحماية ضد الوصول المتزامن هما:

- طريقة السيمافور (semaphore).
- طريقة المراقب (monitors).

طريقة السيمافور

نعبر عن السيمافور بعدد صحيح يحتمل قيمتين :

- القيمة الأولى تعبر عن وضع "تحت" (down).
- القيمة الثانية هي وضع "فوق" (up).

دائما نختبر قيمة السيمافور قبل التنفيذ.واعتماداً على القيمة نتخذ الإجراء اللازم.

القيمة الأولى "تحت":

- إذا كانت قيمة السيمافور أكبر من صفر: ننقص قيمة السيمافور ونستمر في العملية.
- إذا كانت القيمة صفراً يتم إيقاف العملية (منعها من التنفيذ) process is blocked.

القيمة الثانية "فوق":

1. إذا وجدت عمليات موقوفة (محجوزة) نقوم بتحريرها لتواصل عملها.
2. إذا لم توجد عمليات موقوفة نزيد قيمة السيمافور.
3. العمليات التي تم تحريرها يمكنها مواصلة عملها بالتحويل من وضع "تحت".

أهم خاصية في نظام السيمافور هو أنه إذا بدأت عملية "تحت" أو عملية "فوق" لايمكن لعملية أخرى الوصول إلى السيمافور حتى تكتمل العملية أو يتم توقيفها (تحجز).

البرمجة باستخدام السيمافور عرضة للأخطاء ، لذلك يفضل استخدامها في حماية البيانات المشتركة فقط.

أمّا استخدامها في البرمجة فينتج عنه شفرات (أكواد) غير مهيكلة (مرتبة) (unstructured code) مثلها مثل استخدام الأمر goto في لغات البرمجة.

لذلك تفضل معظم نظم التشغيل الحديثة طريقة أخرى للحماية غير طريقة السيمافور، هي طريقة المراقب (monitor).

طريقة المراقب (monitor)

المراقب يشبه إلى حد كبير الكائن في اللغات كائنية التوجه، حيث يمثل وحدة برمجية تتألف من متغيرات وإجراءات (procedures).

المتغيرات تكون خاصة وممنوعاً الوصول إليها إلا عبر الإجراءت الموجودة معها في نفس الوحدة البرمجية (monitor) ، مثل عملية التغليف في اللغة كائنية التوجه (encapsulation).

أهم خاصية في المراقب:

لا يمكن استدعاء أكثر من إجراء من إجراءات المراقب في اللحظة الواحدة .

هذه هي الخاصية الأساسية التي تمنع الوصول المتزامن للمتغيرات الموجودة بالمراقب.

مثال (1)

إذا قامت عملية ما باستدعاء إجراء من المراقب (دخلت إلى المراقب)، و حاولت عملية أخرى استدعاء إجراء من نفس المراقب في نفس الوقت، سيتم حجز العملية الثانية حتى تكتمل العملية الأولى (تخرج من المراقب).

لزيادة التوضيح دعنا نستخدم المثال البرمجي التالي.

مثال (2)

البرنامج التالي يوضح كيف يمكننا منع الوصول المتزامن للمتغير **x**.

```
Monitor  mn {
private:        int x=0;
public:         int value(){ return x;}
          void incr() {x++;}
          void decr(){x--;}
}
```

شكل رقم (4-4): برنامج المراقب mn

يوفر المراقب **mn** الحماية على المتغير x، وذلك بمنع الوصول إليه إلا عبر الدوال الموجودة بالمراقب وهي value() ، incr() ، decr().

و بما أنه لا يمكن استدعاء أكثر من إجراء في اللحظة الواحدة. ولا يمكن الوصول للبيانات إلا بواسطة إجراءات (procedures) المراقب؛ نكون قد حققنا عملية منع الوصول المتزامن للمتغير x.

مثال (3)

ماذا يحدث إذا استدعت عملية ما الإجراء ()decr، وكانت قيمة x هي صفر ؟

في هذه الحالة لابد من منع العملية من تنفيذ الإجراء ()decr (حجزها).

لمثل هذه الأغراض يمكن أن يحتوي المراقب على متغيرات شرطية.

المتغيرات الشرطية:

هي عبارة عن متغيرات تحتمل قيمتين، إحداهما تمثل حالة الانتظار (wait) بينما تمثل القيمة الثانية حالة الإشارة (signal).

إذا كانت هنالك عملية ما داخل المراقب، و استدعت حالة الانتظار في المتغير الشرطي، فسيؤدي هذا إلى حجزها (إيقافها) في نفس الوقت إذا كانت هنالك عملية أخرى في حالة الانتظار يمكنها الآن دخول المراقب وبدء العمل.

أثناء عمل العملية الثانية التي هي داخل المراقب، يمكن السماح للعملية الأولى التي تم إيقافها من المواصلة، وذلك بإرسال إشارة عبر المتغير الشرطي الذي تنتظره العملية الأولى.

العملية التي ولدت الإشارة لابد من أن تترك المراقب، وذلك لتجنب وجود أكثر من عملية داخل المراقب (عدم التزامن).

يمكننا تعديل المراقب cr في المثال السابق لتوضيح كيفية عمل متغيرات شرطية في المراقب.

```
Monitor cr{
private:      int x=0;
       int blocked_process=0;
       Condition unblocked;
public:
              int value(){ return x;}
              void incr(){
                     if(blocked_process==0)
                            x++;
                      else   Signal(unblocked);}
```

```
void decr(){
        if (x==0)
                Blocked_process++;
            Wait(unblocked);
Blocked_process--;
        }
else
x--;}
}
```

شكل رقم (4–5): برنامج المراقب cr بعد إضافة متغيرات شرطية

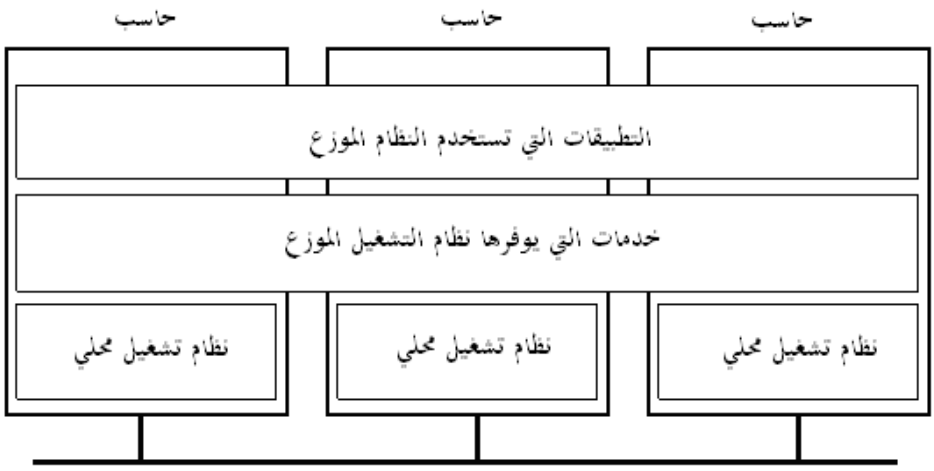
عيوب المراقب

عيبه أنه مبني على لغة برمجة بعينها، مثلاً يوجد بلغة جافا المراقب، ويعتبر كائناً عادياً يقوم بحماية بياناته من الوصول المتزامن عن طريق أوامر التزامن والانتظار (wait) والإشعار (notify) الموجودة بالكائنات. وقد توجد لغة برمجة أخرى تطبق المراقب بطريقة أخرى.

نظم تشغيل الحاسبات المتعددة

multicomputer operating system

نظم تشغيل الحاسبات المتعددة لديها هيكلية مختلفة عن نظم تشغيل المعالجات المتعددة. وهي أكثر تعقيداً منها، لأنه من الصعب وضع البيانات التي يحتاجها نظام التشغيل لإدارة الموارد في مكان مشترك. حيث لا توجد ذاكرة مشتركة هنا. لذلك يتم تبادل البيانات عن طريق الرسائل.



شكل رقم (4-6): نظام التشغيل الموزع

كل جهاز لديه:

1. نواة محلية خاصة به تحتوي على البرامج التي تدير الموارد المحلية لهذا الجهاز (مثل المعالج، الذاكرة ، القرص المحلي).
2. برنامج مخصص لمعالجة الاتصال بين العمليات حيث يتم من خلاله تبادل الرسائل مع الأجهزة الأخرى المرتبطة به.

هنالك طبقة برمجية مبنية ممتدة عبر نظم التشغيل المحلية تخفي تفاصيل الأجهزة وتباين نظمها لتبدو كأنها نظام واحد.

توفر هذه الطبقة الآتي:

- التنفيذ المتوازي و المتزامن للمهام المختلفة.
- ذاكرة افتراضية مشتركة تشبه تلك التي توجد في المعالجات المتعددة.
- تسهيلات أخرى مثل توزيع المهام على المعالجات.
- إخفاء أعطال العتاد ومعالجته.

- إخفاء تفاصيل تخزين البيانات.
- إجراء الاتصال بين العمليات.

أي أن كل ما تتوقعه في نظام التشغيل تجده في هذه الطبقة.

تبادل الرسائل (message passing)

بما أنه لا توجد ذاكرة مشتركة في نظام الحاسبات المتعددة ، فإن نظام التشغيل يوفر تبادل البيانات بين التطبيقات كوسيلة للتشارك في البيانات.

تختلف طريقة تبادل الرسائل باختلاف النظم.

الاتصال المعتمد للرسائل (reliable communication)

من المهم معرفة نوعية الاتصال بين الأجهزة، هل هو معتمد (reliable) أو لا ؟

في الاتصال المعتمد نضمن وصول الرسائل للمستقبل. أما إذا كان الاتصال غير معتمدٍ فيجب على المرسل الانتظار حتى يأتيه إشعار من المستقبل بوصول الرسالة.

5. أنظمة الذاكرة المشتركة الموزعة (DSM)

عزيزي الدارس، كلمة **DSM** هي اختصار للكلمات **distributed shared memory systems**.من خلال التجارب نجد أن برمجة الحاسبات المتعددة أصعب من برمجة المعالجات المتعددة. ففي نظام المعالجات المتعددة تكون عملية الوصول للبيانات المشتركة سهلة، ويمكن استخدام السيمافور أو المراقب لحماية الوصول المتزامن للبيانات. أما في نظام الحاسبات المتعددة فليس هناك غير طريقة واحدة للتشارك في البيانات هي تبادل الرسائل مما يعقد الموضوع ويربطه بالاتصال، وهل هو معتمد أو لا؟

هنالك بحوث كثيرة تحاول محاكاة الذاكرة المشتركة (emulating) في نظام تعدد الحاسبات.

الهدف من هذا المحاكي هو جعل التطبيقات تتعامل مع تعدد الحاسبات بنفس الطريقة التي تتعامل بها مع تعدد المعالجات ، حيث يوفر المحاكي هنا ذاكرة مشتركة خيالية (تشبه التي توجد في النظام متعدد المعالجات).

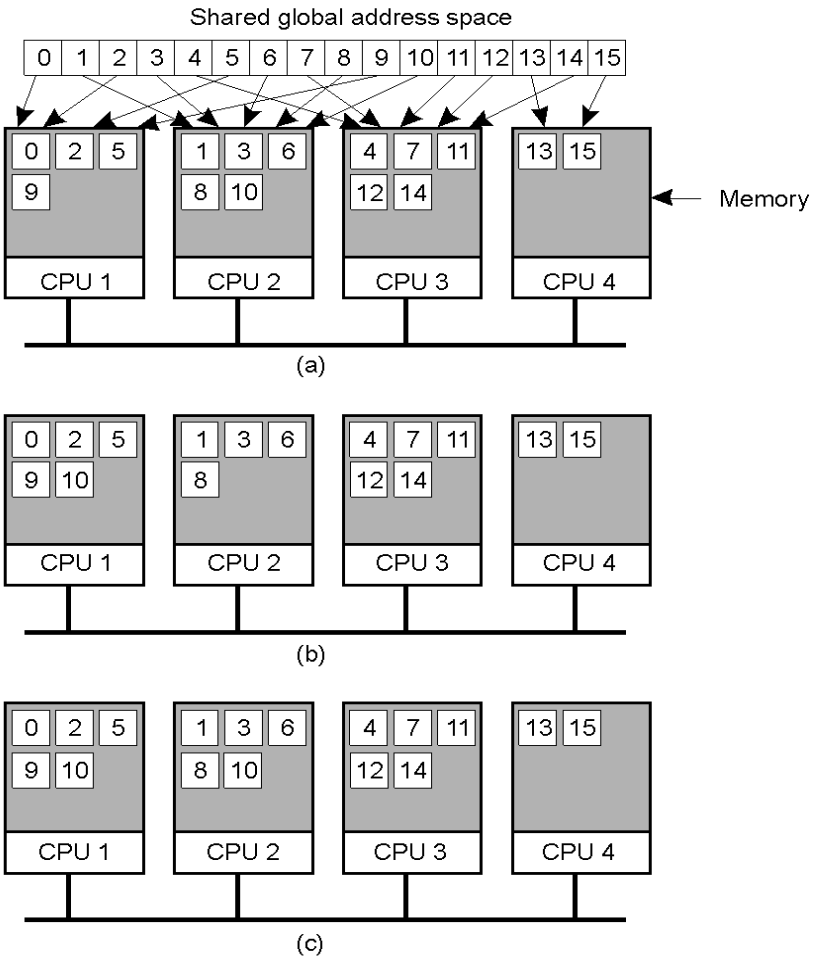
إحدى طرق إنشاء الذاكرة الخيالية هي:

1. استخدام ذواكر كل الأجهزة لتكون ذاكرة خيالية كبيرة.
2. تعمل هذه الذاكرة بنظام الصفحات، حيث تقسم المساحات إلى صفحات (4 كيلو بايت أو 8 كيلو بايت).
3. توجد هذه الصفحات حقيقة في ذواكر المعالجات (حيث لا توجد ذاكرة حقيقية مشتركة).
4. يتعامل المعالج مع المعلومات الموجودة بذاكرته بصورة عادية.
5. إذا احتاج المعالج لمعلومات غير موجودة في ذاكرته المحلية، يحدث الآتي:

a. يرسل المعالج الذي يريد المعلومات رسالة لنظام التشغيل.

b. يبحث نظام التشغيل عن الصفحة المطلوبة ويحضرها للمعالج.

c. يحضر نظام التشغيل الصفحة التي تحتوي المعلومات المطلوبة دون أن يشعر المستخدم بأن هذه الصفحة أحضرت من ذاكرة معالج آخر.



شكل رقم (4-7)

مثال

من الشكل (4-7)، إذا طلب المعالج 1 معلومات من الصفحات 0، 2، 5، 9 ، سيحصل عليها من ذاكرته المحلية.

للحصول على معلومات من صفحات أخرى لابد من الإستعانة بنظام التشغيل. مثلا للحصول على معلومة من الصفحة 10 ، يرسل المعالج طلباً لنظام التشغيل، الذي يقوم بدوره بنقل الصفحة 10 من المعالج 2 إلى المعالج 1، كما في الشكل (4-7).

بعض التحسينات في نظام DSM

عزيزي الدارس، يمكننا تطوير نظام DSM لتحسين الأداء وذلك عن طريق تكرار صفحات القراءة فقط (read only).

مثال

إذا كانت الصفحة 10 في المثال السابق تحتوي معلومات للقراءة فقط وطلبها المعالج 1 فيمكن نسخها من المعالج 2 إلى المعالج 1 (بدلاً من نقلها كما في المثال السابق)، حيث يصبح لدينا نسختان من الصفحة 10، نسخة بالمعالج1 ونسخة ثانية بالمعالج 2، بهذه الطريقة نمكن المعالجين من استخدام نفس الصفحة بدون الحاجة إلى تبادلها بينهما.

1. الطريقة أعلاه يمكن تعميمها لتستخدم مع صفحات القراءة فقط والصفحات الأخرى أيضاً .

إذا كانت الصفحة ليست للقراءة فقط ولكننا نستخدمها لقراءة معلومات منها و لا نحتاج حالياً للكتابة عليها، فيمكن معاملتها معاملة صفحات القراءة فقط وبالتالي يمكن نسخها وتكرارها على كل المعالجات التي تحتاجها.

المشكلة تكمن في إذا قام معالج بتعديل صفحة مكررة، هذا ينتج عنه عدم توافقية في المحتوى، أي أن الصفحات المتكررة قد تختلف محتوياتها بسبب تعديلات متباينة من قبل بعض المعالجات (inconsistency).

لذلك لابد من إجراء معين لضمان توافق البيانات لنفس الصفحة مع نسخها المكررة، فإذا قام معالج بتعديل بيانات على صفحة لابد من إجراء هذا التعديل على كل النسخ المكررة للصفحة.

2. يمكن أيضاً السماح بوجود نسخ مختلفة لنفس الصفحة حتى ولو قام كل معالج بتعديل نستخته مما ينتج عنه نسخ مختلفة (inconsistence). هذه الطريقة بالتجربة وجد أنها تزيد الكفاءة كثيراً . ولكنها تزيد من تعقيد البرمجة، حيث لابد من مراعاة عدم تطابق محتوى الصفحات حين كتابة البرامج.

تعقيد البرمجة التي تقود لها هذه الطريقة تلغي الهدف الرئيسي من بناء نظام DSM، حيث هدفه الأول هو تيسير البرمجة. لذلك تنفي هذه الطريقة هدف **DSM** الرئيسي.

3. تحديد حجم الصفحات يؤثر مباشرة في جودة نظام DSM.

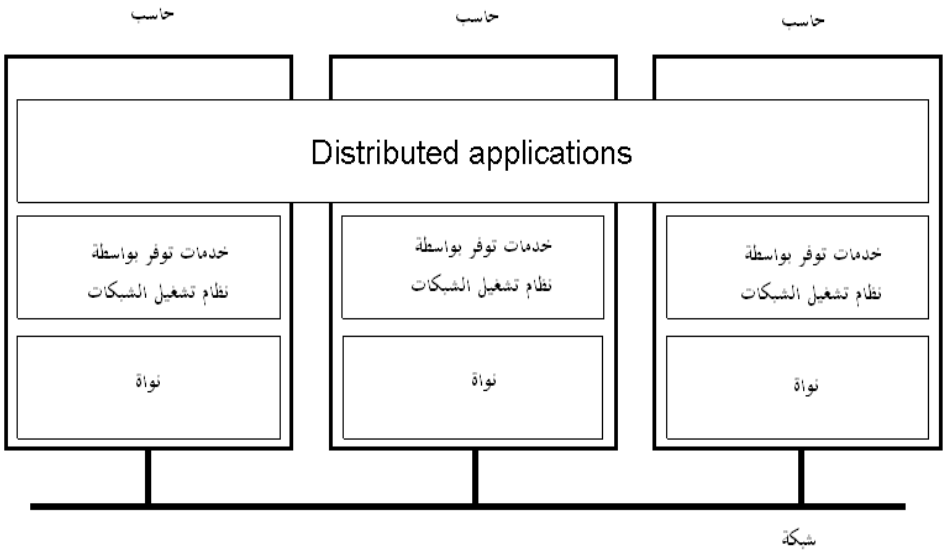
كلفة إرسال صفحة عبر الشبكة يعتمد على إعدادات الإرسال (setting up transfer) وليس على كمية البيانات المرسلة. لذلك إذا كان حجم الصفحة كبيراً فهذا يقلل عدد مرات الإرسال. ولكن من جانب آخر إذا كانت لدينا صفحة تحتوي جزأين من البيانات وكل جزء يحتاجه برنامج في جهاز مختلف على نظام التشغيل، ففي هذه الحال يتم إرسال هذه الصفحة باستمرار بين المعالجين. وجود بيانات في صفحة واحدة لبرنامجين مختلفتين يسمى المشاركة الخطأ (false sharing).

هنالك شد وجذب بين تحسين الأداء مقابل تعقيدات البرمجة في النظام متعدد الحاسبات. حيث نحصل على أداء جيد ولكن ببرمجة معقدة إذا استخدمنا طريقة تبادل الرسائل في النظام. أما إذا استخدمنا نظام DSM فستكون البرمجة سهلة ولكن الأداء ليس بالقدر المطلوب.

نظم تشغيل الشبكات Network Operating System (NOS)

مقارنة بنظام التشغيل الموزع (DOS)، لا يفترض نظام تشغيل الشبكات وجود عتاد متجانس ، ولا يدير النظام كوحدة متماسكة، وإنما هو عبارة عن مجموعة من الحاسبات الأحادية والمستقلة عن بعضها البعض والتي قد تكون مختلفة في كل شيء حتى في نظام التشغيل. هذه الأجهزة متصلة ببعضها البعض عن طريق رابط ما.

يوفر نظام تشغل الشبكات تسهيلات تمكن المستخدمين من الاستفادة من الخدمات المتاحة في هذه الأجهزة.



شكل رقم (4-8)

يعمل نظام تشغيل الشبكات مع أجهزة متباينة، ولكن هذا لا ينفي عمله مع أجهزة متجانسة في العتاد و مستقلة عن بعضها البعض ، و مرتبطة مع بعضها البعض برابط معين.

الخدمات التي يوفرها نظام تشغيل الشبكات

يمكننا وصف نظام تشغيل الشبكات من خلال الخدمات التي يوفرها للتطبيقات والمستخدمين. من الخدمات التي نجدها في معظم نظم تشغيل الشبكات الآتي:

1. خدمة الدخول على جهاز بعيد والتعامل معه كأنه جزء من جهازك المحلي، إذ أنه وبعد الدخول على الجهاز البعيد ترسل أوامرك عبر الشبكة لذاك الجهاز وترى النتائج على جهازك المحلي.

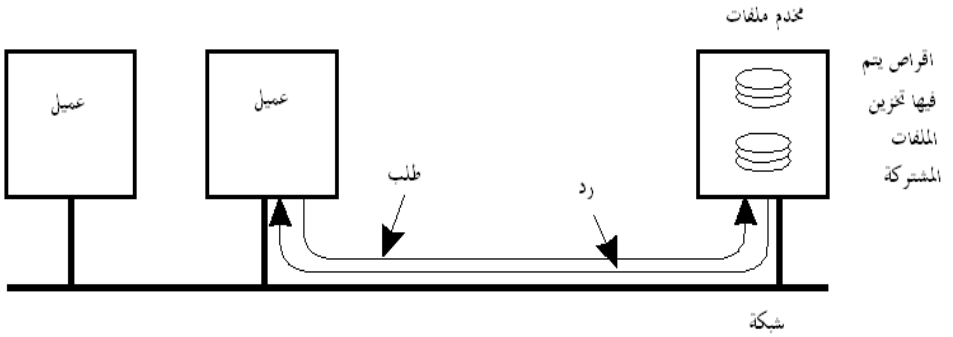
يمكن أن تدخل على أكثر من جهاز بعيد في نفس الوقت وتتعامل مع هذه الأجهزة البعيدة كأنها جزء من جهازك.

2. خدمة نسخ ملفات من أجهزة بعيدة، مثلا الأمر :

Rcp m1:f1 m2:f2

يقوم بنسخ الملف f1 من الجهاز m1 الى الجهاز m2 وبالاسم f2.

3. خدمة نظام الملفات حيثُ يخصَّصُ جهازٌ خادِّم (server) تخزن به الملفات وتتاح لكل الأجهزة. يقوم برنامج نظام الملفات (file server) الموجود بهذا الخادم باستلام طلبات المستخدمين من الأجهزة العميلة (clients)، التي قد تكون لقراءة ملف أو تخزين ملف أو غيره. حيث تنفذ الطلبات بواسطة نظام الملفات ويرسل الرد للأجهزة العميلة، الشكل (4-9).



شكل رقم (4-9)

نظام تشغيل الشبكات هو أكثر بدائية من نظام التشغيل الموزع، الفرق بين الاثنين هو أن الأخير يحاول صبغ النظام بالشفافية (transparency).

عيوب نظام تشغيل الشبكات

عدم وجود شفافية (transparency) في نظام تشغيل الشبكات أوجدت به عيوباً منها:

- صعوبة الاستخدام، فمثلا عملية الدخول على جهاز بعيد، أو نقل الملفات تحتاج إلى خبرة من المستخدم.
- مشاكل إدارية مثل:
- استقلالية الأجهزة. (نظم التشغيل في الشبكة تحتاج أحيانا إلى إدارة مستقلة).

- أحيانا تجد صعوبة في الدخول على جهاز بعيد ما لم يكن لديك حساب (account) في ذاك الجهاز.
- إذا أراد مستخدم التعامل بكلمة مرور واحدة فلابد من تغييرها في كل جهاز.
- إيضاً قد يحتاج المستخدم معالجة أذونات الوصول (access permissions) في كل جهاز.

1. ليس هنالك طريقة بسيطة لتغيير الأذونات لأنها متشابهة في كل مكان.

هذه الطريقة اللامركزية للحماية (security) تجعل من الصعب أحياناً حماية نظام تشغيل الشبكات من الهجمات التخريبية.

مزايا نظم تشغيل الشبكات

يعتبر نظام تشغيل الشبكات سهل التعديل مقارنة مع نظام التشغيل الموزع. فيمكننا إضافة محطة أو إزالة محطة من النظام بسهولة، وأحياناً يتم ذلك فقط بتوصيل المحطة مع كيبل الشبكة و إبلاغ بقية المحطات بوجودها.

أسئلة تقويم ذاتي



1. اذكر ثلاثة أمثلة لنماذج طبقة وسيطة.
2. ما الفرق بين نظام التشغيل الموزع، ونظام تشغيل الشبكات، ونظام التشغيل العادي؟

الطبقة الوسيطة middleware

ليكون النظام الموزع حقيقياً لابد من توفر نقطتين مهمتين فيه:

1. حاسبات مستقلة (independent computers).
2. نظام واحد متماسك (single coherent system).

الأولى نجدها متوفرة في نظام تشغيل الشبكات ويفتقر للثانية، بينما تتوفر الثانية في نظام التشغيل الموزع ولا توجد به الأولى.

فكلا النظامين لا يوفر النظام الموزع الحقيقي.

إذن كيف الوصول إلى النظام الموزع الحقيقي ؟

تحدثا مسبقاً عن صفات الأنظمة الموزعة وأهمها:

o التوسعية. o الانفتاحية. o الشفافية. o سهولة الاستخدام

هذه الصفات إذا تمكنا من توفيرها في نظام فسيصبح تقريبا نظاماً موزعاً حقيقياً .

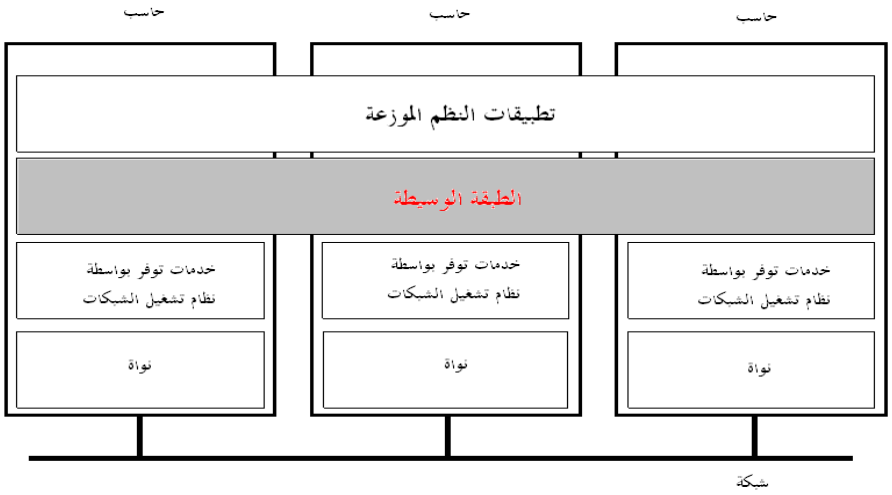
إذا نظرنا إلى نظام تشغيل الشبكات نجد أن به خاصية التوسعية وخاصية الانفتاحية، أما نظام التشغيل الموزع فهو سهل الاستخدام و عالي الشفافية (transparence).

فكلا النظامين ناقص. فما هو الحل؟

الحل يكمن في إضافة طبقة برمجية (layer) لنظام تشغيل الشبكات تضيف إليه الخواص الناقصة مثل سهولة الاستخدام والشفافية.

مكان الطبقة الوسيطة positioning middleware

- كثير من التطبيقات الموزعة تستخدم الواجهات البرمجية التي يوفرها نظام تشغيل الشبكات.مثلاً قد تحتاج إلى إجراء اتصال باستخدام الـ sockets التي تسمح للعديد من العمليات بأجهزة مختلفة لتبادل الرسائل.
- قد تستخدم التطبيقات بعض واجهات (interfaces) نظام الملفات المحلي. نجد هنا أنه من الصعب إخفاء تفاصيل التوزيع (distribution). لذلك الحل هو وضع طبقة بين نظام تشغيل الشبكات والتطبيقات، هذه الطبقة تسمى الطبقة الوسيطة (middleware) وتوضع في الوسط بين نظام تشغيل الشبكات والتطبيقات.



شكل رقم (4-10): الطبقة الوسيطة

كل نظام محلي (هو جزء من نظام تشغيل الشبكات) يفترض أن يوفر إدارة الموارد المحلية بالإضافة إلى توفير أيسر طرق الاتصال التي تربطه بالأجهزة الأخرى، هذا يعني أن الطبقة الوسيطة نفسها لا تدير الأجهزة منفردة وإنما يترك هذا لنظام التشغيل المحلي.

الهدف المهم هو إخفاء التباين الموجود بالنظام عن التطبيقات.

العديد من نماذج الطبقة الوسيطة توفر مجموعة كاملة من الخدمات وهي تشجع استخدامات الواجهات أكثر من أي شي آخر.

المشاكل :الاتفاق على طبقة وسيطة موحدة ليس سهلاً لذلك لابد من منظمة تُعرِّف وتضع المعايير والأسس التي تبني طبقة وسيطة معيارية. هنالك العديد من المعايير التي وضعت ولكنها ليست متوافقة مع بعضها البعض، والأسوأ من ذلك أن المنتجات التي تطبق نفس المعايير (لكن من مصانع مختلفة) من الصعب أن تتعامل مع بعضها البعض.

نماذج الطبقة الوسيطة (middleware models)

لتبسيط عملية تطوير التطبيقات الموزعة وجعل تكاملها أيسر ما يكون، تضع معظم الطبقات الوسيطة نموذجاً لوصف التوزيع والاتصال.

من أبسط النماذج الموجودة التعامل مع كل شيء على أنه ملف. هذه الطريقة كانت أولاً في اليونكس (UNIX).

أمثلة لنماذج طبقات وسيطة

نظام الملفات الموزع: هو عبارة عن طبقة تضيف نوع خدمات واحد لنظام تشغيل الشبكات، حيث يتم دعم شفافية توزيع الملفات. مما ساهم في انتشار هذه الطبقة هو قابليتها للتوسع. يتعامل المستخدم مع الملفات الموزعة كأنها في مكان واحد.

نداء الإجراء البعيد (RPC) يمكن لبرنامج في جهاز نداء إجراء في برنامج آخر بجهاز آخر (بعيد)، وكأن ذاك الإجراء موجود بالجهاز المحلي، فالبرنامج الذي قام بنداء الإجراء لا يميز ما إذا كان الإجراء بعيداً أو قريباً وما إذا كان هنالك اتصال تم أو لا، الفرق الوحيد الذي يحس به المستخدم هو أن تنفيذ الإجراء البعيد يستغرق وقتاً أطول عن نداء الإجراء العادي (ذلك للاتصالات التي تتم والرسائل التي يتم تبادلها).

نداء الكائن البعيد (distributed objects invocations)

هي تطبيق لفكرة نداء الإجراء البعيد أعلاه، لكن على الكائنات (**objects**)، فالطريقة (**method**) البعيدة المستدعاة توجد في كائن بالجهاز البعيد. وهي تعمل بنفس الطريقة حيث لا يميز البرنامج الذي استدعى الطريقة (**method**) هل هي بعيدة أم بالجهاز نفسه (شفافية النداء).

نموذج الوثائق الموزعة (distributed documents)

هذا النموذج هو وراء نجاح الويب. ففي الويب تنظم المعلومات في شكل وثائق حيث تخزن كل الوثائق في الأجهزة بشفافية (لا نعرف موقع الوثيقة الحقيقي).

هذه الوثائق متنوعة المحتوى فقد يكون فيها نصوص، أصوات، فيديو، صور، وغيرها. ونتعامل مع الوثيقة للوصول إليها عبر عنوان URL.

مثال : لفتح الموقع http://www.sustech.edu نكتبه في شريط العنوان بالمتصفح، ونتعامل مع هذا العنوان فقط لنحصل على الوثيقة، فلا نحتاج معرفة معلومات إضافية عن الوثيقة مثل مكان تخزينها :

o في أي بلد o في أي جهاز o ما نظام تشغيل الخادم o ما اللغة المكتوب بها الموقع.
o ما نوع عتاد الخادم o ما شكل الشبكات التي تربطني به.

وإذا تم نقل هذا الموقع من خادم ويب إلى خادم ويب آخر، ببلدة ثانية، بدولة أخرى، بقارة أخرى، بنظام تشغيل مختلف، فسأظل أصل إليه بنفس الطريقة (شفافية المكان).

خدمات الطبقة الوسيطة (middleware services)

هنالك العديد من الخدمات التي توفرها الطبقة الوسيطة نذكر منها:

1. خدمة الشفافية (إخفاء التفاصيل): وذلك بإخفاء تفاصيل الاتصال بين العمليات وتبادل الرسائل في الشبكة. مثل ما توفره طريقة نداء الإجراء البعيد من الوصول للدوال البعيدة بشفافية وما يوفره نظام الملفات الموزع (distributed file systems) من الوصول للبيانات البعيدة بشفافية. فيستطيع المستخدم الوصول لملفات موزعة في أجهزة مختلفة وكأنها في مكان واحد، ولا يحتاج تذكر أماكن هذه الملفات ولا في أي أجهزة تتوزع.
2. خدمة التسمية (naming): تستخدم لإخفاء تفاصيل المورد والوصول إليه بسهولة مثلما يحدث في الويب، حيث يمثل العنوان (**URL**) الاسم الذي يشير إلى مكان الوثيقة بشفافية.
3. الحماية (security): من الخدمات المهمة التي يجب أن تتوافر بالطبقة الوسيطة الحماية.

انفتاح الطبقة الوسيطة (openness)

بما أن النظم الموزعة الحديثة تكون في شكل طبقة وسيطة تُبنى على نظام تشغيل الشبكات، فالتطبيقات الموزعة سوف تُبنى على الطبقة الوسيطة وبالتالي ليس هنالك علاقة مباشرة مع نظام تشغيل الشبكات والنظام الموزع. أي أن التطبيقات تكون مستقلة عن نظام تشغيل الشبكات ولكنها بالمقابل مرتبطة بالطبقة الوسيطة ومعتمدة عليها. وهذا يولد مشكلة هنا وهي أن الطبقة الوسيطة تكون غير منفتحة بالشكل المطلوب.

تحدد درجة انفتاح النظام بالواجهات (interface) التي يوفرها. فلا بد أن تكون الواجهات كاملة لتفي بكل متطلبات المستخدم.

عدم توافر واجهات كاملة يدفع بمطوري الأنظمة لإضافة واجهاتهم الخاصة لإكمال النقص الموجود في واجهات الطبقة الوسيطة. هذا يعني أن كل مجموعة مصممي نظم تضيف واجهاتها، مما يقود إلى تباين الواجهات المضافة. بالتالي تصبح تطبيقات كل جهة معتمدة على واجهاتها التي أضافتها للطبقة، وهذا يؤدي إلى انتفاء خاصية التنقلية **(portability)** بين تطبيقات الطبقة الوسيطة الواحدة.

أسئلة تقويم ذاتي



1. ما المهمة الأساسية للطبقة الوسيطة؟
2. هل تعتقد أن نظام التشغيل الموزع كافٍ لتوفير بيئة ملائمة للأنظمة الموزعة ؟ و لماذا ؟
3. ما مهام برمجيات النظم الموزعة ؟
4. ما نظام الملفات الموزع؟
5. ماذا تعرف عن نداء الإجراء البعيد **(RPC)**؟

الخلاصة

عزيزي الدارس، في هذه الوحدة تعرفنا على نظم التشغيل الموزعة، وهي النوع الذي يدير الموارد لتبدو وكأنها جهاز واحد كبير فهو بذلك يوفر خاصية الشفافية التي تخفي جزئيات العتاد والبرامج وانتشارها في الشبكة وكذلك نظم تشغيل المعالج الواحد، حيث يقوم نظام التشغيل هنا بإدارة عتاد حاسب به معالج واحد. وتوفير بيئة سهلة للمستخدم وتطبيقاته للاستفادة من العتاد بصورة مثلى.أما نظام التشغيل الحديث فهو يمكن من تغيير نظام التشغيل ليصبح أكثر مرونة وذلك بتعديل النواة في نظام التشغيل العادي، فيصبح النظام الحديث كالآتي:تحويل البرامج التي تدير العتاد من النواة إلى مساحة المستخدم.جعل البرامج الضرورية لنظام التشغيل في النواة.كما تناولنا في الوحدة نظم التشغيل متعددة المعالجات حيث يختلف نظام التشغيل هنا عن سابقه في أنه يدير عدة معالجات وتوضع البيانات التي يحتاجها نظام التشغيل لإدارة العتاد في ذاكرة مشتركة. ومن الجداول المرفقة تمكنا من مقارنة بين نظام التشغيل الأحادي ونظام التشغيل المتعدد. وفيما يتعلق بأنظمة الذاكرة المشتركة الموزعة (DSM) تعرفنا من خلال التجارب أن برمجة الحاسبات المتعددة أصعب من برمجة المعالجات المتعددة. ففي نظام المعالجات المتعددة تكون عملية الوصول للبيانات المشتركة سهلة، ويمكن استخدام السيمافور أو المراقب لحماية الوصول المتزامن للبيانات. أما في نظام الحاسبات المتعددة فليس هناك غير طريقة واحدة للتشارك في البيانات هي تبادل الرسائل مما يعقد الموضوع ويربطه بالاتصال، وهل هو معتمد أو لا؟

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس، سنتناول في الوحدة القادمة إن شأ الله الفرق بين التوزيع الأفقي والتوزيع الرأسي لنماذج الخادم والعميل وكذلك سنتعرف على المعمارية ثنائية الطبقات والمعمارية ثلاثية الطبقات وكيف أن فيالأخيرة هذه يصبح الخادم عميلاً . نسمى معمارية الخادم/العميل المعمارية ثلاثية الطبقات (tier).وسنتعرف أيضا على كيفية تخزين طبقة المعالجة في خادم لوحدها، و طبقة البيانات في خادم آخر، و طبقة واجهات المستخدم في جهاز ثالث (جهاز العميل)، بمعنى آخر إن المستويات الثلاثة تكون موزعة في ثلاثة أجهزة. وفي نهايتها سنبين أنواع البروتوكولات التي تستخدم للاتصال بين الخادم والعميل.

مسرد المصطلحات

نظم التشغيل الموزعة ((Distributed Operating System (DOS)

هذا النوع يدير الموارد لتبدو وكأنها جهاز واحد كبير، فهو بذلك يوفر خاصية الشفافية التي تخفي جزئيات العتاد والبرامج وانتشارها في الشبكة.

نظم تشغيل المعالج الواحد ((uniprocessor operating system (UOS)

يقوم نظام التشغيل هنا بإدارة عتاد حاسب به معالج واحد. وتوفير بيئة سهلة للمستخدم وتطبيقاته للاستفادة من العتاد بصورة مثلى.

نظام التشغيل الحديث (modern operating system)

يمكن تغيير نظام التشغيل ليصبح أكثر مرونة وذلك بتعديل النواة في نظام التشغيل العادي، فيصبح النظام الحديث كالآتي:تحويل البرامج التي تدير العتاد من النواة إلى مساحة المستخدم، جعل البرامج الضرورية لنظام التشغيل في النواة.

نظم التشغيل متعددة المعالجات (multiprocessor operating system)

يختلف نظام التشغيل هنا عن سابقه في أنه يدير عدة معالجات وتوضع البيانات التي يحتاجها نظام التشغيل لإدارة العتاد في ذاكرة مشتركة. الجدول التالي يوضح مقارنة بين نظام التشغيل الأحادي ونظام التشغيل المتعدد.

نظم تشغيل الحاسبات المتعددةmulticomputer operating system

نظم تشغيل الحاسبات المتعددة لديها هيكلية مختلفة عن نظم تشغيل المعالجات المتعددة. وهي أكثر تعقيداً منها، لأنه من الصعب وضع البيانات التي يحتاجها نظام التشغيل لإدارة الموارد في مكان مشترك. حيث لا توجد ذاكرة مشتركة هنا. لذلك يتم تبادل البيانات عن طريق الرسائل.

أنظمة الذاكرة المشتركة الموزعة (DSM)

كلمة **DSM** هي اختصار للكلمات **distributed shared memory systems** .من خلال التجارب نجد أن برمجة الحاسبات المتعددة أصعب من برمجة المعالجات

المتعددة. ففي نظام المعالجات المتعددة تكون عملية الوصول للبيانات المشتركة سهلة، ويمكن استخدام السيمافور أو المراقب لحماية الوصول المتزامن للبيانات. أما في نظام الحاسبات المتعددة فليس هناك غير طريقة واحدة للتشارك في البيانات هي تبادل الرسائل، مما يعقد الموضوع ويربطه بالاتصال، وهل هو معتمد أو لا؟

النواة Kernel

هي البرامج التي تقوم بإدارة العتاد من معالج وذاكرة وطرفيات وأقراص وغيرها. وهي التي تتحمل بالذاكرة الرئيسية عند فتحك للحاسب وهي التي تشغل وتدير باقي التطبيقات. وهي لطبيعة عملها تستطيع الوصول لأي جزء من العتاد وتنفيذ أي أوامر دون أن يمنعها أو يحدها شيء.

مساحة المستخدم (الغلاف) shell

هي الواجهة التي يتعامل معها المستخدم والتطبيقات، ومن خلالها نستطيع تنفيذ تطبيقاتنا، لذلك أحيانا تسمى واجهة المستخدم أو وضع المستخدم أو مساحة المستخدم. وهنا لا يسمح بالوصول المباشر للعتاد، فلا يمكن الوصول للذاكرة التي يقبع فيها نظام التشغيل مثلاً، ولا يمكن التعامل مع كل المسجلات. فيحدد نظام التشغيل الجزء المسموح الوصول إليه من الذاكرة والمسجلات التي يمكن للتطبيقات الوصول إليها (يمنع الوصول مباشرة إلى مسجلات الأجهزة (device registers).

نداءات النظام (system calls)

حيث توفر هذه النداءات خدمات التعامل مع العتاد للتطبيقات.فمثلاً إذا أراد تطبيق قراءة بيانات من ملف مخزن بالقرص، فإنه سينادي دالة بنظام التشغيل تقوم بإحضار المعلومات من القرص.

طريقة المراقب (monitor)

المراقب يشبه إلى حد كبير الكائن في اللغات كائنية التوجُّه، حيث يمثل وحدةً برمجية تتألف من متغيرات وإجراءات (procedures).

المراجع

1- 1.Coulouris, G. et al., (2004), Distributed Systems, Concepts and Design, Addison Wesley
2- Farley, J. (1998), Java Distributed Computing, O'reilly & Associates
3- Fedorov, A, (2002), A Programmer Guide to .NET, Addison Wesley
4- Ince, D. (2002), Developing Distributed and E-Commerce Applications, Pearson Education
5- Liu, M. (2004), Distributed Computing Principles and Applications, Pearson Education
6- Schonauer, W. (1998), Scientific Supercomputing: Architecture and use of Shared and Distributed Memory Parallel Computers, Willi Schonauer, Germany
7- Tanenbaum, A., (2002), The Amoeba Distributed Operating System, Vrije Universiteit Technical Report, URL ftp://ftp.cs.vu.nl/pub/amoeba/Intro.ps.Z
8- Tanenbaum , A. AND Van Steen, M. (2002), Distributed Systems Principles and Paradigms, Prentice Hall
9- Tanenbaum, A. (1995), Distributed Operating Systems, 1/e, Prentice Hall, USA
10- Osman, A., (may 2006), A Multi-level Web-based Parallel Processing System, TRANSACTIONS ON ENGINEERING, COMPUTING AND TECHNOLOGY VOLUME 13 MAY 2006 ISSN 1305-5313
11-Forouzan, A. (2007), Data Communications and Networking, Mc Graw Hill, Fourth Edition
12-Tanenbaum, A., (2001), Modern Operating Systems, Prentice Hall, Second Edition
13-Tanenbaum, A., (1999), Structred Computer Organization, Prentice Hall, Fourth Edition


الوحدة الخامسة
هيكلية النظام الموزع

محتويات الوحدة

الموضوع	الصفحة
المقدمة	107
تمهيد	107
أهداف الوحدة	**108**
1. برنامج الاتصال في النظام الموزع	**109**
1.1 تقسيم البرامج في النظام الموزع	110
2. الاتصال	110
1.2 البروتوكول غير المتصل	110
2.2 البروتوكولات موجهة الاتصال	112
3. تقسيم التطبيق إلى مستويات	112
1.3 واجهة المستخدم	113
2.3 المعاجلة	113
3.3 البيانات	114
4.معمارية الخادم والعميل	114
5. المعمارية ثلاثية الطبقات	116
6. التوزيع الرأسي والأفقي	118
الخلاصة	120
لمحة مسبقة عن الوحدة القادمة	121
إجابات التدريبات	122
مسرد المصطلحات	123
المراجع	107

المقدمة

التمهيد

عزيزي الدارس،

مرحبا بك إلى هذه الوحدة التي تأتيك بعنوان "هيكلية النظام الموزع".

تنقسم الوحدة إلى سته أقسام في القسم الأول منها يبين لنا مقدمة عامة عن برامج الاتصال وفي القسم الثاني نتعرف على انواع البروتكولات المستخدمة في الاتصال أما في القسم الثالث فسنتعرف على *تقسيم التطبيق إلى مستويات*. ونوضح العلاقة بين الخادم والعميل، فأحيانا يكون الخادم عميلاً لخادم آخر، فهو من جانب خادم ومن جانب آخر عميل. وأيضا نوضح فيها تقسيم تطبيقات الخادم والعميل والتي تربط المستخدمين بقواعد البيانات ومستوياتها الثلاثة: واجهة المستخدم user interface، المعالجة Processing. البيانات data. وفي القسم الرابع من هذه الوحدة سنبحث في مفهوم معمارية الخادم والعميل وطرق توزيع الطبقات. وسنخصص القسم الخامس للتفصيل في المعمارية ثلاثية الطبقات. أما القسم الاخير من هذه الوحدة فيبحث في الطبقات المتعددة (التوزيع الرأسي والأفقي، ويبين الفرق بين التوزيع الأفقي والتوزيع الرأسي لنماذج الخادم والعميل. عزيزي الدارس نتمنى لك أن تستفيد من هذه الوحدة.

أهداف الوحدة

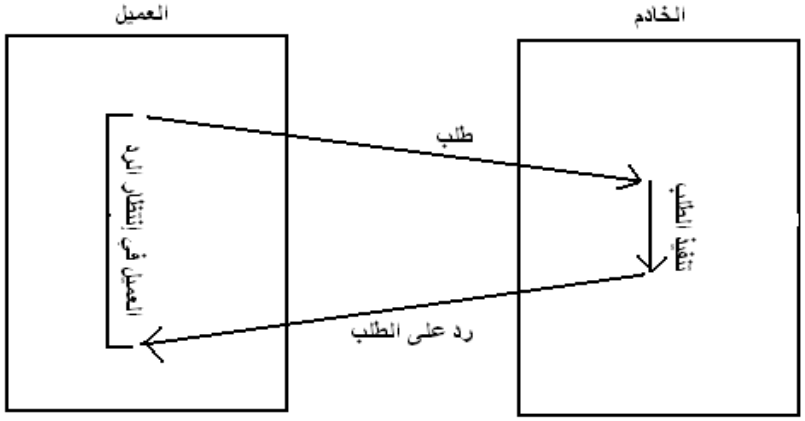


عزيزي الدارس بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

1. تعرف أنواع البروتوكولات المستخدمة في الاتصال.
2. تفرق بين المعمارية ثنائية الطبقات والمعمارية ثلاثية الطبقات.
3. تبين معمارية الطبقات المتعددة.
4. تذكر مثالاً للتوزيع الأفقي.

1. برنامج الاتصال في النظام الموزع

عزيزي الدارس، النظام الموزع في مجمله عبارة عن برامج موزعة على العديد من الأجهزة بالشبكة. وتعمل هذه البرامج مع بعضها البعض. أجزاء النظام الموزع هذه ما هي إلا عمليات (processes) موزعة، ولكي تعمل مع بعضها لابد من أن تتصل ببعضها البعض. بمعنى آخر النظام الموزع هو عمليات منتشرة في أجهزة الشبكة وتتعامل مع بعضها البعض عن طريق الرسائل.هذا يعني أن كل برنامج يطلب خدمة من برنامج آخر بالنظام تكون العلاقة بينهما علاقة عميل مع خادم، فالبرنامج الذي يطلب الخدمة يسمى عميلاً والبرنامج الذي يؤدي الخدمة يسمى خادماً . التعامل بين العميل والخادم يتم عن طريق تبادل الرسائل، فطلب خدمة بواسطة البرنامج العميل هي رسالة، والرد عليها من قِبل البرنامج الخادم هي أيضاً رسالة (طلبٌ و ردّ (request/reply)).



شكل رقم (5-1): الطلب والردّ.

إذا تواجد برنامج العميل وبرنامج الخادم في جهاز واحد يستخدم العميل العنوان المحلي **(localhost)** للتعامل مع الخادم أو مقابله الرقمي 127.0.0.1.

1.1 تقسيم البرامج في النظام الموزع

تنقسم البرامج في النظام الموزع إلى نوعين:

- برامج الخادم (sever): تؤدي خدمة معينة، مثل خدمة نظام الملفات أو خدمة قواعد البيانات.
- برنامج العميل (client): يطلب خدمة من الخادم بإرسال طلب ثم انتظار الرد منه.

2. الاتصال

الاتصال بين العميل والخادم يتم عبر بروتوكول، وهنالك نوعان من البروتوكولات المستخدمة هي:

1.2 البروتوكول غير المتصل connectionless.

2.2 البروتوكول مُوجَّه الاتصال connection-oriented.

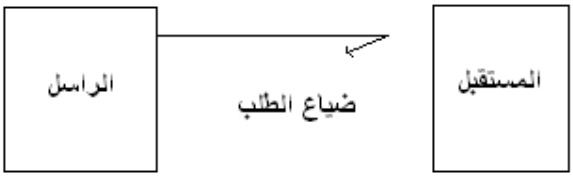
1.2 البروتوكول غير المتصل

إذا كانت الشبكة جيدة (معتمدة (reliable)) إلى حد ما (كما في الشبكات المحلية (LAN)) يمكننا استخدام البروتوكول غير المتصل، **حيث يعمل كالآتي:**

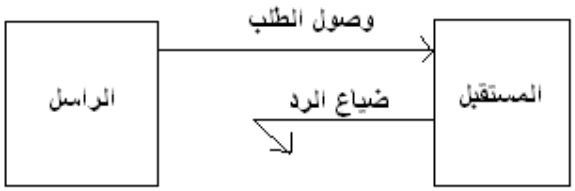
- يقوم العميل بتحزيم الرسالة.
- تعريف نوع الخدمة التي يريد.
- إضافة البيانات اللازمة.
- إرسال الرسالة للخادم الذي يكون دائماً في حالة انتظار طلبات العملاء.
- يستلم الخادم طلب العميل.
- ينفذ المطلوب.
- يرسل الردّ في شكل رسالة للعميل.

يعتبر استخدام هذا البروتوكول جيداً ما دام أن الرسائل لا تضيع أو تعطب، وطريقة الطلب/ الردّ تسير على ما يرام.

ولكن **ماذا يفعل العميل إذا لم يأته ردّ على طلبه** ؟يمكنه تكرير الطلب وذلك بإعادة إرساله مرة ثانية. و لكن قد ينتج عن إعادة الطلب مشكلة أن العميل لا يدري سبب عدم وصول الردّ، هل هو بسبب **ضياع الطلب** أو بسبب **ضياع الردّ** على الطلب ؟



شكل رقم (5-2): ضياع الطلب



شكل رقم (5-3): ضياع الرد

في حالة ضياع الرد فإن إعادة إرسال الطلب سيجعل الخادم يؤدي العمل مرة ثانية.

مثال

في أي نظام بنكي، إذا طلب عميل بالبنك "سحب مبلغ مليون دولار من حسابه" عبر الشبكة، ولم يأت الرد (تم عملية خصم المبلغ بالخادم ولكن ضاع الرد)، سيقوم العميل بإعادة إرسال الطلب مرة ثانية، وبالتالي سيتم سحب مبلغ مليون دولار آخر من الحساب. في مثل هذه النظم لابد من معرفة سبب عدم الرد على الطلب قبل إعادة إرسال الطلب مرة أخرى.

أما إذا كان الطلب هو معرفة الرصيد، فليس هنالك إشكال في إعادة إرسال الطلب لأنه لا يؤثر على المعلومات وإنما هو عرض لمعلومة فقط.

2.2 البروتوكولات موجهة الاتصال

هنالك بعض نظم الخادم والعميل تستخدم هذا البروتوكول. هذا البروتوكول غير مناسب للشبكات المحلية لأنه يقلل الأداء فيها. أما للشبكات الواسعة فيعتبر مناسباً لأن الاتصال فيها غالبا يكون غير جيد (غير معتمد (unreliable)).

كل البروتوكولات المستخدمة لتطبيقات الإنترنت مبنية على اتصال TCP/IP المعتمد.

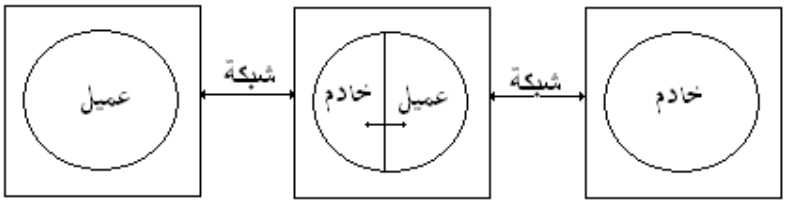
طريقة عمل هذا البروتوكول تكون كالآتي:

- إنشاء اتصال مع الخادم.
- إرسال الطلب عبر الاتصال.
- إغلاق الإتصال.

عملية إنشاء الاتصال قبل الإرسال وإغلاقه بعد الإرسال لكل رسالة يستغرق وقتاً، في حالة الرسائل القصيرة ستكثر هذه العملية مما يزيد في بطء الشبكة.

3. تقسيم التطبيق إلى مستويات

عزيزي الدارس، ليس هنالك فرق واضح بين الخادم والعميل، فأحيانا يكون الخادم عميلاً لخادم آخر، فهو من جانب خادم ومن جانب آخر عميل. مثال لذلك خادم قواعد البيانات الذي يخدم العميل الذي يعبئ نماذج إدخال البيانات، وفي نفس الوقت قد يكون عميلاً لخادمات أخرى (إذا كانت قاعدة البيانات موزعة) حيث يرسل البيانات الواردة من العملاء إلى مخدمات مختلفة لتطبيقها على قواعد البيانات (فهو هنا عميل لمخدمات أخرى)، انظر الشكل (5-4).



شكل رقم (5-4): جهاز يعمل كخادم وعميل في نفس الوقت

يمكن تقسيم تطبيقات الخادم والعميل والتي تربط المستخدمين بقواعد البيانات إلى ثلاث طبقات (مستويات) هي:

1.3 واجهة المستخدم user interface.

2.3 المعالجة Processing.

3.3 البيانات data.

1.3 واجهة المستخدم

توجد بهذه الطبقة البرامج التي تمكن المستخدم من التعامل مع التطبيقات. مثلا البرامج التي تظهر في الشاشات والواجهات الرسومية التي يتعامل معها بواسطة لوحة المفاتيح والماوس.

2.3 المعالجة

في هذا المستوى تعالج البيانات التي أدخلت بواسطة المستوى الأول ، فمثلاً إذا أدخلنا بيانات عبر نموذج في المستوى الأول فمعالجة البيانات المدخلة يتم هنا في هذا المستوى.

مثال

الواجهة التي يتعامل معها المستخدم في محرك البحث (search engine) و يدخل من خلالها كلماته البحثية وينقر على زر البحث فتظهر النتائج، تعتبر هذه الواجهة واجهة المستخدم. فهو يدخل المعلومات فيها وتظهر له النتائج من خلالها.

في مستوى المعالجة نجد أن محرك البحث عبارة عن برنامج يقوم بالبحث عن الكلمات المفتاحية التي أدخلها المستخدم باستخدام أوامر استفسارات (queries)، ثم يخرج نتائج الاستفسارات في شكل صفحة html.

يُمثّل مستوى البيانات في هذا المثال قاعدة البيانات المفهرسة والتي تحتوي كمية ضخمة من الصفحات التي يبحث فيها محرك البحث عن ضالته المنشودة.

3.3 البيانات

يحتوي هذا المستوى على البرامج التي تعالج البيانات الحقيقية التي يعمل عليها التطبيق.

من خواص هذا المستوى المهمة جعل البيانات معدة وجاهزة للاستخدام حتى ولو لم يكن هنالك تطبيق يعمل عليها الآن.

يتكون هذا المستوى في أبسط صوره من نظام ملفات.

توضع هذه الطبقة غالباً بجهاز الخادم، بينما تكون طبقة واجهة المستخدم بجهاز العميل، وتكون الطبقة الوسطى إما في الخادم أو في العميل.

من المهام التي تقوم بها هذه الطبقة توفير توافقية البيانات (consistency) لكل التطبيقات.

4.معمارية الخادم والعميل

عزيزي الدارس، إن تقسيم التطبيق إلى ثلاث طبقات (مستويات) قادنا إلى سؤال مهم هو:

كيف توزع هذه الطبقات الثلاث بين الخادم والعميل؟

هنالك طرق مختلفة لتوزيع الطبقات، منها:

a) وضع جزء من طبقة من واجهة المستخدم في جهاز العميل والبقية في جهاز الخادم.

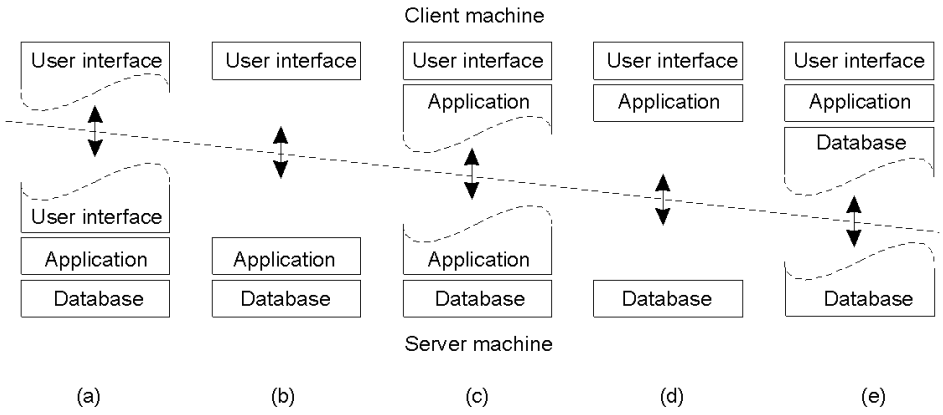
b) وضع طبقة واجهة المستخدم في جهاز العميل، ووضع الطبقتين الأخرتين في جهاز الخادم، نسمي هذا التوزيع المعمارية ثنائية الطبقات (two-tiered architecture).

c) وضع طبقة واجهة المستخدم وجزء من طبقة المعالجة بجهاز العميل والبقية بالخادم. مثلاً يمكن أن تكون واجهة المستخدم مكونة من نماذج إدخال البيانات ، الجزء الموضوع من مستوى المعالجة بالجهاز العميل يمكن أن يكون الجزء الذي يعالج البيانات المدخلة والتأكد من صحتها (validation).

d) وضع واجهات المستخدم والمعالجة جميعها في جهاز العميل، بينما توضع طبقة البيانات بالخادم. مثال لذلك إذا كان لدى حاسب شخصي مرتبط بشبكة مع خادم به نظام ملفات موزع أو قاعدة بيانات فتتم عمليات الإدخال والمعالجة بجهاز العميل بينما نستخدم نظام الملفات للتخزين في جهاز الخادم.

e) يمكن أن تكون الواجهة والمعالجة وجزء من طبقة البيانات بجهاز العميل، والجزء الآخر من طبقة البيانات بالخادم. مثال لذلك عملية تصفُّح الويب، حيث يمثل المتصفح الواجهة والمعالجة، بينما تخبأ الصفحات الواردة من الإنترنت بالقرص المحلي، هذه النسخ المخبأة بجهاز العميل تمثل جزءاً من طبقة البيانات.

إذا طلبت صفحة سيحضرها لك المتصفح من النسخ المخبأة بجهازك "إن وجدت"، وإذا لم توجد محلياً، سنتصل ببقية طبقة البيانات الموجودة بالخادم للحصول على المعلومة من مكانها الأصلي، انظر الشكل (5-5).

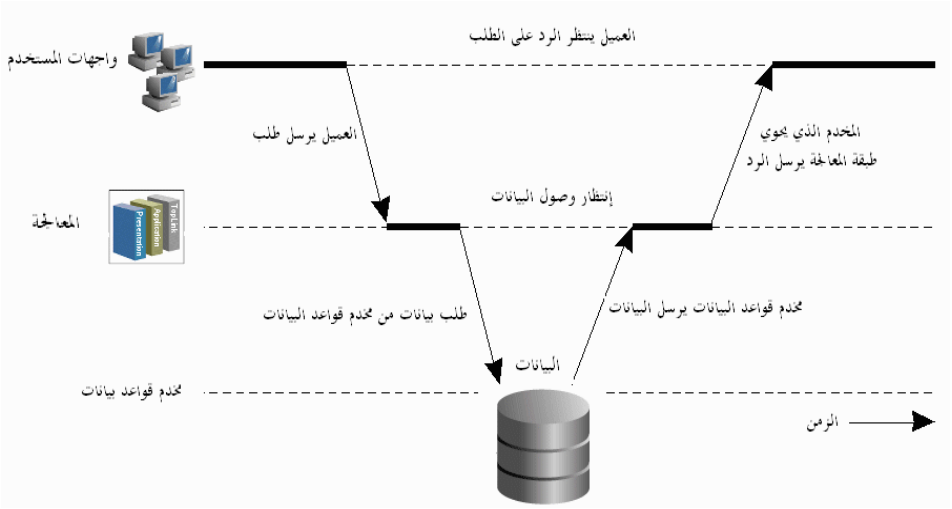


شكل رقم (5-5): توزيع المستويات بين الخادم والعميل.

5. المعمارية ثلاثية الطبقات (Three-tiers architecture)

عزيزي الدارس، حين يصبح الخادم عميلاً نسمَّي معمارية الخادم/العميل المعمارية ثلاثية الطبقات (tier).

في مثل هذه المعمارية تخزن طبقة المعالجة في خادمٍ لوحدها و طبقة البيانات في خادم آخر و طبقة واجهات المستخدم في جهاز ثالث (جهاز العميل)، بمعنى آخر إن المستويات الثلاث تكون موزعة في ثلاثة أجهزة.



شكل رقم (5-6): معمارية ثلاثية الطبقات.

من تطبيقات المعمارية ثلاثية الطبقات في مجال قواعد البيانات هي وضع نظام قواعد البيانات بمخدم، ووضع برامج المعالجة بمخدم ثانٍ، وواجهات المستخدم (الإدخال والنماذج) بجهاز آخر، أنظر الشكل (5-6).

أسئلة تقويم ذاتي

1. اذكر مثالاً لجهاز يعمل كخادم وعميل في نفس الوقت ؟
2. ما الفرق بين المعمارية ثنائية الطبقات والمعمارية ثلاثية الطبقات ؟
3. هنالك نوعان من البروتوكولات تستخدمان للاتصال بين الخادم والعميل، ما هما؟

6. التوزيع الرأسي والأفقي

عزيزي الدارس، معمارية تعدد الطبقات (multi-tier) للخادم/العميل ناتجة من تقسيم التطبيق إلى طبقات (ثلاث طبقات). وتدور المسألة حول أين نضع كل طبقة؟ في جهاز الخادم أو في جهاز العميل. تقسيم التطبيق وتوزيعه بهذه الطريقة يسمى التوزيع الرأسي (vertical distribution).

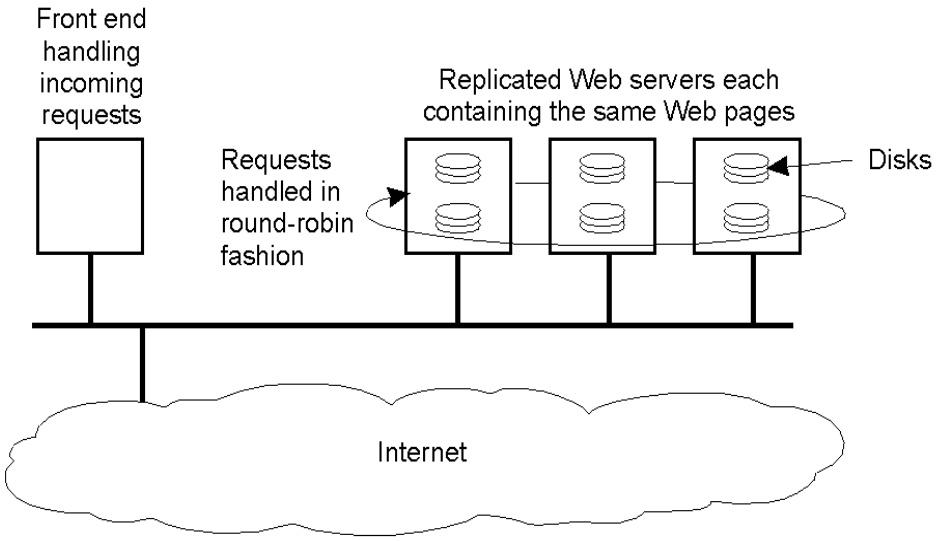
بعض المعماريات الحديثة تقوم بتوزيع تطبيقات الخادم أو العميل أفقيا (horizontal distribution) وذلك بعمل نسخٍ متعددة من التطبيق، و جعل كل نسخة تعمل على بيانات مختلفة، مما يساعد في توزيع الحمل بين الأجهزة (balancing the load)، فنقسم البيانات على النسخ بدلاً من أن تنفذ على نسخة واحدة مما يوفر الوقت.

مثال

لتوزيع جهد مخدم ويب وتقليل الحمل عليه ُضيف عدداً من مخدمات ويب بالشبكة، تحتوي هذه المخدمات على نفس الصفحات الموجودة بالمخدم الرئيسي (تعتبر نسخة مكررة من مخدم ويب الرئيسي (replication)).

إذا تم تعديل صفحة في أحد هذه المخدمات لابد من تعديلها في بقية المخدمات للحفاظ على التوافقية (consistency).

في حالة طلب عميل لصفحة ما، يرسل الطلب لأحد هذه المخدمات ويرد ذاك المخدم بإرسال الصفحة للعميل، إذا طلب عميل آخر أو نفس العميل صفحة أخرى، يرد مخدم ويب الثاني على الطلب، وإذا طلب عميل ثالث صفحة ثالثة يرد مخدم ويب الثالث (الشكل (5-7))، وهكذا كل طلب يرد عليه مخدم (بالتوالي)، ثم نعيد الكرة، فيبدأ المخدم الأول مرة ثانية بالرد على الطلب التالي، هذه الطريقة تسمى (round robin).



شكل رقم (5-7): نسخ مكررة من مخدم ويب.

فائدة تكرار مخدمات ويب :

- يقلل الحمل على مخدم ويب (هنالك مخدمات أخرى تساعده).
- تكون خدمة التصفح سريعة (يمكن الرد على الطلبات بالتوازي).
- إذا تعطل مخدم ويب هنالك مخدمات أخرى كبدائل (اعتمادية).
- توفير نطاق (bandwidth) كافٍ.

أسئلة تقويم ذاتي

1. ما الفرق بين التوزيع الأفقي والتوزيع الرأسي لنماذج الخادم والعميل ؟
2. اذكر مثالاً للتوزيع الأفقي ؟
3. ماهي فائدة تكرار مخدمات الويب

الخلاصة

عزيزي الدارس، في هذه الوحدة تناولنا مفهوم البروتكولات المستخدمة في الاتصال وكيفية تقسيم التطبيق إلى مستويات.

ووضحنا العلاقة بين الخادم والعميل، فأحيانا يكون الخادم عميلاً لخادم آخر فهو من جانب خادم ومن جانب آخر عميل. وأيضا بحثنا في تطبيقات الخادم والعميل والتي تربط المستخدمين بقواعد البيانات ومستوياتها الثلاثة وهي واجهة المستخدم user interface.

المعالجة Processing. البيانات data وفي القسم الرابع من هذه الوحدة تعرفنا على مفهوم معمارية الخادم والعميل وطرق توزيع الطبقات و القسم الخامس تناولنا المعمارية ثلاثية الطبقات أما القسم الأخير من هذه الوحدة بحثنا في الطبقات المتعددة (التوزيع الراسي والافقي ويبينا الفرق بين التوزيع الأفقي والتوزيع الرأسي لنماذج الخادم

لمحة مسبقة عن الوحدة القادمة

هذه الوحدة تناولنا فيها تعريف مفهوم الشفافية وكيفية تحقيق هذا المفهوم وكذلك البحث في إمكانية تطبيق الشفافية بأوجه مختلفة وذلك عن طريق إخفاء كيفية الوصول إلى المورد مكان المورد (شفافية الوصول تحول المورد من مكانه تحول المورد من مكانه وهو في حالة استخدام (relocation) وكذلك نسخ المورد بأماكن متعددة (تكرار المورد)

وأيضا سنتناول **مفهوم** درجة الشفافية (degree of transparency) ويتم الرد على السؤال هل الشفافية دوماً محمودة ؟

مسرد المصطلحات

برامج الخادم (sever):

تؤدي خدمة معينة، مثل خدمة نظام الملفات أو خدمة قواعد البيانات.

برنامج العميل (client)

يطلب خدمة من الخادم بإرسال طلب ثم انتظار الرد منه.

الاتصال

الاتصال بين العميل والخادم يتم عبر بروتوكول، وهنالك نوعين من البروتوكولات المستخدمة هي:البروتوكول غير المتصل connectionless.والبروتوكول موجهة الاتصال connection-oriented.

البروتوكول غير المتصل

إذا كانت الشبكة جيدة (معتمدة (reliable)) إلى حد ما (كما في الشبكات المحلية (LAN)) يمكننا استخدام البروتوكول غير المتصل،

معمارية الخادم والعميل

تقسيم التطبيق إلى ثلاث طبقات (مستويات) قادنا إلى سؤال مهم هو:كيف توزع هذه الطبقات الثلاث بين الخادم والعميل؟

المعمارية ثلاثية الطبقات (Three-tiers architecture)

حين يصبح الخادم عميلاً نسمى معمارية الخادم/العميل المعمارية ثلاثية الطبقات (tier).

في مثل هذه المعمارية تخزن طبقة المعالجة في خادم لوحدها و طبقة البيانات في خادم آخر، و طبقة واجهات المستخدم في جهاز ثالث (جهاز العميل)، بمعنى آخر ان المستويات الثلاثة تكون موزعة في ثلاثة أجهزة.

المراجع

1- 1.Coulouris, G. et al., (2004), Distributed Systems, Concepts and Design, Addison Wesley

2- Farley, J. (1998), Java Distributed Computing, O'reilly & Associates

3- Fedorov, A, (2002), A Programmer Guide to .NET, Addison Wesley

4- Ince, D. (2002), Developing Distributed and E-Commerce Applications, Pearson Education

5- Liu, M. (2004), Distributed Computing Principles and Applications, Pearson Education

6- Schonauer, W. (1998), Scientific Supercomputing: Architecture and use of Shared and Distributed Memory Parallel Computers, Willi Schonauer, Germany

7- Tanenbaum, A., (2002), The Amoeba Distributed Operating System, Vrije Universiteit Technical Report, URL ftp://ftp.cs.vu.nl/pub/amoeba/Intro.ps.Z

8- Tanenbaum , A. AND Van Steen, M. (2002), Distributed Systems Principles and Paradigms, Prentice Hall

9- Tanenbaum, A. (1995), Distributed Operating Systems, 1/e, Prentice Hall, USA

10- Osman, A., (may 2006), A Multi-level Web-based Parallel Processing System, TRANSACTIONS ON ENGINEERING,

COMPUTING AND TECHNOLOGY VOLUME 13 MAY 2006 ISSN 1305-5313

11-Forouzan, A. (2007), Data Communications and Networking, Mc Graw Hill, Fourth Edition

12-Tanenbaum, A., (2001), Modern Operating Systems, Prentice Hall, Second Edition

13-Tanenbaum, A., (1999), Structred Computer Organization, Prentice Hall, Fourth Edition


الوحدة السادسة
الشفافية
(Transparency)

محتويات الوحدة

الموضوع	الصفحة
المقدمة	**127**
تمهيد	**127**
أهداف الوحدة	**128**
1. الشفافية	**129**
2. تطبيق الشفافية بأوجه مختلفة	**129**
1.2 إخفاء الوصول (access transparency)	130
2.2 إخفاء الموقع (location transparency)	130
3.2 تحريك الموقع (migration)	131
4.2 (relocation).	131
5.2 التكرار (replication)	131
6.2 إخفاء تزامن الوصول (concurrent)	132
7.2 إخفاء حدوث الأعطال (failure & recovery)	132
8.2 **إخفاء الـ** persistence	133
3. درجة الشفافية (degree of transparency)	134
الخلاصة	**134**
لمحة مسبقة عن الوحدة القادمة	**136**
مسرد المصطلحات	**136**
المراجع	**137**

المقدمة

تمهيد

عزيزي الدارس،

مرحباً بك إلى الوحدة السادسة من مقرر "النظم الموزعة" والتي تأتيك "بعنوان الشفافية". نقسم هذه الوحدة إلى ثلاثة أقسام: القسم الأول تناولنا فيه تعريف مفهوم الشفافية وكيفية تحقيق هذا المفهوم. القسم الثاني بحثنا فيه إمكانية تطبيق الشفافية بأوجه مختلفة وذلك عن طريق إخفاء كيفية الوصول إلى المورد مكان المورد (شفافية الوصول تحول المورد من مكانه وهو في حالة استخدام relocation وكذلك نسخ المورد بأماكن متعددة (تكرار المورد) (replication)، وأيضا تشارك المستخدمين في المورد تزامنياً (concurrency)، ولم نغفل جانب حدوث أعطال ومعالجتها (failure & recovery)وأخيراً في هذا القسم بحثنا عن مكان تواجد البرامج داخل الحاسب (هل هي بالذاكرة أو بالقرص الصلب) (persistence).

في القسم الثالث من هذه الوحدة تناولنا **مفهوم** درجة الشفافية (degree of transparency) وتم الرد عن السؤال هل الشفافية دوماً محمودة ؟

أهداف الوحدة



عزيزي الدارس بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

1. تذكر ثلاثة من أوجه الشفافية مع ذكر مثال لكل نوع .
2. تعرف ما الفائدة من الوصول المتزامن للموارد .
3. تفهم كيف نحقق توافقية البيانات في الموارد المشتركية .
4. تستطيع الرد على سؤال هل الشفافية محمودة دوماً ؟ و لماذا؟

1. الشفافية

عزيـزي الـدارس، مـن الأهـداف التـي يسـعى النظـام المـوزع لتوفيرهـا لمسـتخدميه خاصـية الشـفافية (إخفـاء انتشـار برامجـه (عملياتـه processes) و مـوارده (resources) عبر العديد من الحاسبات).

فاذا جعلنا المستخدم و تطبيقاته تتعامل مع أجزاء النظام المنتشرة كأنها نظام واحد وليس مجموعة أجهزة مستقلة عن بعضها البعض ، نكون قد حققنا الشفافية (transparency).

2. تطبيق الشفافية بأوجه مختلفة

وذلك عن طريق إخفاء:

- كيفية الوصول إلى المورد (access transparency).
- مكان المورد (شفافية الوصول location transparency).
- تحول المورد من مكانه (migration).
- تحول المورد من مكانه وهو في حالة استخدام (relocation).
- نسخ المورد بأماكن متعددة (تكرار المورد) (replication).
- تشارك المستخدمين في المورد تزامنيا (concurrency).
- حدوث أعطال ومعالجتها (failure & recovery).
- مكـان تواجـد البـرامج داخـل الحاسـب (هـل هـي بالـذاكرة أو بـالقرص الصـلب) (persistence).

يمكننا التحدث عن هذه الأوجه بصورة أوسع قليلا وتفصيل ذلك فيما يأتي:

1.2 إخفاء الوصول (access transparency)

نحاول هنا إخفاء:

- التباين في طرق تمثيل البيانات بالحاسبات المختلفة.
- الطريقة التي يتعامل بها المستخدم مع الموارد.

مثال (1)

إذا كان لشخص حاسب كمبيوتر من شركة إنتل وأراد إرسال بيانات لشخص آخر يمتلك حاسباً من النوع صن اسبارك (Sun SPARC)، فعليه أن يضع في الحسبان أن طريقة ترتيب البايتات في أجهزة إنتل (little endian) (حيث البايت ذو الترتيب الأعلى يرسل أولاً)، بينما معالجات إسبارك (SPARC) تستخدم نظام (big endian) (حيث يكون البايت ذو الترتيب الأدنى يرسل أولاً).

مثال (2)

قد تكون الحاسبات تعمل بنظم تشغيل مختلفة، وعليه قد يكون كل حاسب لديه نظام تسمية ملفات مختلف، أيضا قد تختلف الطريقة التي يتعامل بها كل نظام مع الملفات.

كل هذا التباين يجب أن يُخفى عن المستخدم وتطبيقاته وأن يُتعامل مع الملفات والموارد كلها دون الانشغال بهذا التباين أو بذل جهد لمعالجته أو حتى العلم بوجوده.

فلا يعلم ولا يكترث المستخدم بأن الجهاز إنتل أو الجهاز إسبارك وأن كليهما لا يتوافقان في تمثيل البيانات. فالنظام الموزع هو الذي يهتم بالتباين ومعالجته دون تدخل المستخدم.

2.2 إخفاء الموقع (location transparency)

إذا أخفينا عن المستخدم موقع الموارد وأصبح يتعامل معها بطريقة واحدة دون الاهتمام بمكانها فهذا يضيف خاصية شفافية المكان (location transparency).

مثال

تستخدم التسمية naming لإخفاء مكان المورد. أوضح مثال لذلك هو استخدام الأسماء لإخفاء موقع الصفحات بالإنترنت، حيث نستخدم عناوين المواقع (URLs). يتعامل الشخص مع مواقع ويب بهذه العناوين دون معرفة مكان تواجد الموقع بالإنترنت.مثلاً www.**ous.edu.sd** هو موقع جامعة السودان للعلوم والتكنولوجيا ، ولكن من منا يعرف الجهاز الذي يخزن به هذا الموقع أو أين يوجد.

3.2 تحريك الموقع (migration)

إذا كان بالإمكان تحويل المورد من مكانه إلى مكان آخر دون أن يغير هذا في طريقة الوصول إليه نكون قد حققنا خاصية تحويل الموقع (migration).

4.2 (relocation).

أحياناً يتحول المورد من مكانه وهو في حالة استخدام. هنا تضاف خاصية أخرى هي (relocation).

مثال

كمثال للـ relocation، استخدامك لجهاز محمول (laptop) وأنت متصل بالإنترنت لاسلكياً، فتستطيع تصفح الإنترنت وأنت متحرك بسيارتك من مكان لآخر دون أن يؤثر هذا في تصفحك بشيء. كذلك تتنقل بهاتفك الجوال من مكان إلى آخر ومن مدينة إلى آخرى وهو في حالة اتصال دائم واستخدام دائم.

5.2 التكرار (**replication**)

يلعب تكرار نسخ الموارد دوراً هاماً في الأنظمة الموزعة، فقد ننسخ (نكرر البيانات أو الأجهزة) لتحسين الأداء بوضع نسخ قريبة من مستخدميها وإخفاء تواجد هذه النسخ، فلا يعلم المستخدم هل هو يتعامل مع جهاز واحد أو مع عدة نسخ من الجهاز.

إخفاء تكرار الموارد مرتبط بالآتي:

- إعطاء كل مورد مكرر، اسماً واحداً (متشابه) وإلا سينكشف التكرار.
- إخفاء تكرار الموارد مرتبط أيضاً بإخفاء الموقع (كيف ؟).

6.2 إخفاء تزامن الوصول (concurrent)

عزيزي الدارس، من أهداف الأنظمة الموزعة هي التشارك في الموارد من عتاد وبرامج وبيانات. وقد تكون البيانات المشتركة في مخدم مركزي تتعامل معه كل الأجهزة ، فتخزّن بياناتها فيه، وتتشارك عبره مع الأجهزة الأخرى.

لذلك كان لابد من ألا يعلم المستخدم بأن هنالك من يشاركه في الوصول للموارد التي يستخدمها الآن.

الوصول المتزامن (concurrent access) للموارد المشتركة يجعل هذه الموارد في حالة توافق (consistent state).

تتحقق التوافقية (consistency) عبر طريقة غلق المورد (locking) ، حيث يحتكر المستخدم الوصول للمصدر المطلوب (exclusive access) ثم بعد الانتهاء من استخدامه يقوم بفتحه للآخرين (unlock).

أسئلة تقويم ذاتي



1. اذكر ثلاثة من أوجه الشفافية مع ذكر مثال لكل نوع.
2. ما الفائدة من الوصول المتزامن للموارد؟

7.2 إخفاء حدوث الأعطال (failure & recovery)

إخفاء الأعطال يعني أن المستخدم لا يعلم و لا يظهر له لَن مورداً ما تعطل، وأن النظام قد قام بإصلاحه (recover).

عملية إخفاء الأعطال من أصعب المهام التي يقوم بها النظام الموزع. وقد تكون في بعض الأحيان مستحيلة، خاصة عندما لا يستطيع النظام تمييز عدم استجابة المورد هل يرجع لعطل به أو لأنه مزدحم وبطيء؟

مثال

عند الاتصال بمخدم ويب مزدحم قد يظهر للشخص الذي يطلب صفحة ما "أن الصفحة غير موجودة" في المتصفح. وقد يكون السبب ليس لعطل بالجهاز، أو لأن الصفحة فعلاً غير موجودة، وإنما لأن المخدم مزدحم، فينتظر المتصفح فترة زمنية محددة وبسبب تأخر الرد يظهر المتصفح عبارة "الصفحة غير موجودة".

8.2 إخفاء الـ persistence

هنا تتم تعمية المستخدم عن مكان المورد داخل الحاسب، هل هو بالذاكرة أم بالقرص الصلب ؟

مثال

في بعض قواعد البيانات كائنية التوجه (object oriented databases) توجد طرق تمكننا من استدعاء دوال (methods) موجودة بكائن مخزن (stored object) بالقرص الصلب (ليس بالذاكرة الرئيسية).

يتم ذلك بتحميل مخدم قاعدة البيانات لنسخة من حالة الكائن من القرص إلى الذاكرة الرئيسية ، ثم يتم تنفيذ الدوال التي تتبع له، ثم قد تحفظ حالة الكائن مرة أخرى بالقرص.

المستخدم لا يعنيه تحول حالة الكائن من القرص للذاكرة أوالعكس.

الـ persistence مهمة للأنظمة الموزعة ولغيرها.

3. درجة الشفافية (degree of transparency)

عزيزي الدارس، هل الشفافية دوماً محمودة ؟

رغم أهمية الشفافية للأنظمة الموزعة لكن توجد حالات يفضل فيها عدم وجودها.

مثال

قد يطلب شخص ما أن تصل صحيفته الإلكترونية اليومية في بريده الإلكتروني يومياً عند الساعة السابعة صباحاً . و بما أن الساعة السابعة صباحاً بالخرطوم غير التي بلندن، غير التي باليابان. فإن إخفاء تفاصيل الفروق بين الزمن هنا يسبب مشكلة.

1.3 درجة الشفافية مقابل الأداء

أحياناً تكون الكفاءة ضحية للشفافية، مثلاً :

- تطبيقات الإنترنت تحاول مراراً الاتصال بمخدم إنترنت، قبل أن تتوقف عن محاولاتها تكون قد أضاعت الكثير من الوقت. هنا إخفاء الأعطال يؤدي إلى إبطاء عمل النظام ككل، لأن محاولات الاتصال بالمخدم قد لا تتكرر إذا عُرف العطل بالمخدم، ولكن وبما أن العطل مخفي عنّا فإننا سنحاول الاتصال مرات ومرات بالمخدم دون أن ندري أن الجهاز الذي نكرر الاتصال به متوقف أصلاً ولا يستجيب.

لمعالجة هذه الحالة لابد من تقليل محاولات الاتصال بالمخدم أو إلغائها تماماً .

- عندما تكون لدينا نسخ مكررة من البيانات (replicated)، فإن متابعة توافق البيانات (هل هي متطابقة دوماً) يؤثر سلباً على كفاءة النظام، لأن هذه العملية تستغرق وقتاً لا يخفى عن المستخدم.

خلاصة القول أن الشفافية ضرورية ومهمة ومطلوب وجودها بالنظام الموزع عند تصميمه، لكن يجب ألا تكون على حساب كفاءة النظام.

أسئلة تقويم ذاتي



1. هل الشفافية محمودة دوماً ؟ و لماذا ؟
2. كيف نحقق توافقية البيانات في الموارد المشتركية ؟
3. ما فوائد التكرار؟

الخلاصة

عزيزي الدارس، نلخص هذه الوحدة ونقول:

إنَّ الأهداف التي يسعى النظام الموزع لتوفيرها لمستخدميه هي خاصية الشفافية (إخفاء انتشار برامجه (عملياته **processes**)، و موارده (**resources**)، عبر العديد من الحاسبات).

فإذا جعلنا المستخدم و تطبيقاته تتعامل مع أجزاء النظام المنتشرة كأنها نظام واحد وليس مجموعة أجهزة مستقلة عن بعضها البعض ، نكون قد حققنا الشفافية

أما إمكانية تطبيق الشفافية فتتم بأوجه مختلفة، وذلك عن طريق إخفاء كيفية الوصول إلى المورد مكان المورد (شفافية الوصول تحول المورد من مكانه وهو في حالة استخدام (relocation) وكذلك نسخ المورد بأماكن متعددة (تكرار المورد) (replication) وأيضا بتشارك المستخدمين في المورد تزامنيا (concurrency). ولم نغفل جانب حدوث أعطال ومعالجتها (failure & recovery).وأخيراً عن طريق مكان تواجد البرامج داخل الحاسب (هل هي بالذاكرة أو بالقرص الصلب)؟ (persistence).

لمحة مسبقة عن الوحدة التالية

سنلتقي بك عزيزي الدارس في الوحدة القادمة و نتحدث عن أهداف الأنظمة الموزعة الانفتاح على العالم الخارجي. والمقصود بالانفتاح في النظام الموزع وكيفية توفير واجهاتٍ للوصول إلى خدماته وإمكانية إضافة هذه الواجهات لها، وأيضاً سنعرف الصيغة (syntax) و كيفية استخدامها. وهنالك شرح يوضح معاني الخدمات الموفرة (semantics) .

كذلك سنتناول مفهوم التوسعية لإضافة مكونات جديدة إليه أو استبدال مكونات موجودة أو تحديثها دون أن يؤثر هذا على المكونات الأخرى بالنظام. وأيضا سنتعرف على مفهوم المرونة و على كيفية التسهيل لتهيئة النظام بمكونات مختلفة قد تكون من مطورين مختلفين.

مسرد المصطلحات

إخفاء الوصول (access transparency)

نحاول هنا إخفاء التباين في طرق تمثيل البيانات بالحاسبات المختلفة والطريقة التي يتعامل بها المستخدم مع الموارد.

إخفاء الموقع (location transparency)

إذا أخفينا عن المستخدم موقع الموارد وأصبح يتعامل معها بطريقة واحدة دون الاهتمام بمكانها فهذا يضيف خاصية شفافية المكان (location transparency).

تحريك الموقع (migration).

إذا كان بالإمكان تحويل المورد من مكانه إلى مكان آخر دون أن يغير هذا في طريقة الوصول إليه نكون قد حققنا خاصية تحويل الموقع (migration).

أحياناً يتحول المورد من مكانه وهو في حالة استخدام. هنا تضاف خاصية أخرى هي (relocation).

إخفاء تزامن الوصول (concurrent)

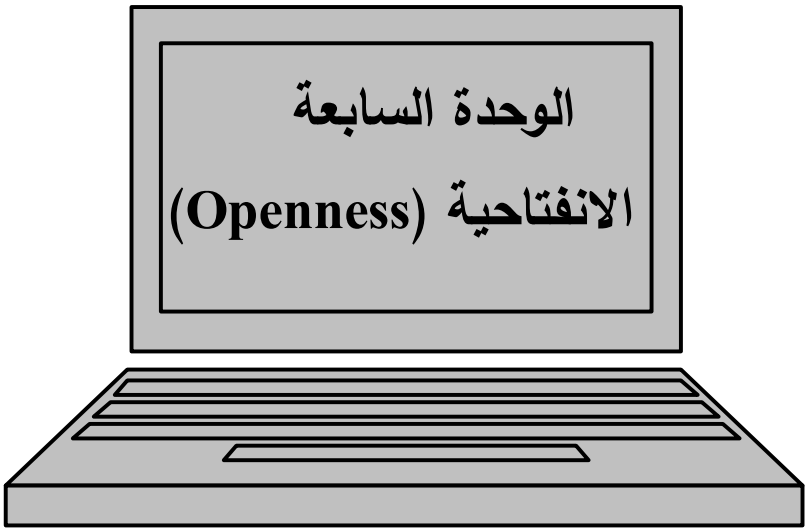
من أهداف الأنظمة الموزعة هي التشارك في الموارد من عتاد وبرامج وبيانات. وقد تكون البيانات المشتركة في مخدم مركزي تتعامل معه كل الأجهزة ، فتخزّن بياناتها فيه، وتتشارك عبره مع الأجهزة الأخرى.

إخفاء حدوث الأعطال (failure & recovery)

إخفاء الأعطال يعني أن المستخدم لا يعلم و لا يظهر له أن مورداً ما تعطل، وأن النظام قد قام recover بإصلاحه.

المراجع

1- 1.Coulouris, G. et al., (2004), Distributed Systems, Concepts and Design, Addison Wesley
2- Farley, J. (1998), Java Distributed Computing, O'reilly & Associates
3- Fedorov, A, (2002), A Programmer Guide to .NET, Addison Wesley
4- Ince, D. (2002), Developing Distributed and E-Commerce Applications, Pearson Education
5- Liu, M. (2004), Distributed Computing Principles and Applications, Pearson Education
6- Schonauer, W. (1998), Scientific Supercomputing: Architecture and use of Shared and Distributed Memory Parallel Computers, Willi Schonauer, Germany
7- Tanenbaum, A., (2002), The Amoeba Distributed Operating System, Vrije Universiteit Technical Report, URL ftp://ftp.cs.vu.nl/pub/amoeba/Intro.ps.Z
8- Tanenbaum , A. AND Van Steen, M. (2002), Distributed Systems Principles and Paradigms, Prentice Hall
9- Tanenbaum, A. (1995), Distributed Operating Systems, 1/e, Prentice Hall, USA
10- Osman, A., (may 2006), A Multi-level Web-based Parallel Processing System, TRANSACTIONS ON ENGINEERING, COMPUTING AND TECHNOLOGY VOLUME 13 MAY 2006 ISSN 1305-5313
11-Forouzan, A. (2007), Data Communications and Networking, Mc Graw Hill, Fourth Edition
12-Tanenbaum, A., (2001), Modern Operating Systems, Prentice Hall, Second Edition
13-Tanenbaum, A., (1999), Structred Computer Organization, Prentice Hall, Fourth Edition


الوحدة السابعة
الانفتاحية (Openness)

محتويات الوحدة

الموضوع	الصفحة
المقدمة	141
تمهيد	141
أهداف الوحدة	**141**
1. الانفتاحية	**142**
1.1 صيغة الخدمات	142
2.1 شروط الواجهات التي توفر خدمات لمستخدمين	144
2. التوسيعة	**145**
3. المرونة	**145**
1.3 الوصول للمرونة	145
3 .2 الأشياء التي لا يستطيع المستخدم التحكم بها	147
الخلاصة	**149**
لمحة مسبقة عن الوحدة القادمة	**150**
إجابات التدريبات	**151**
مسرد المصطلحات	**152**
المراجع	**141**

المقدمة

تمهيد

عزيزي الدارس،

مرحباً بك في الوحدة السابعة من هذا المقرر والتي تأتيك بعنوان "الانفتاحية"

تنقسم هذه الوحدة إلى ثلاثة أقسام في القسم الاول منها نتحدث عن أهداف الأنظمة موزعة الانفتاح على العالم الخارجي. والمقصود بالانفتاح في النظام الموزع وكيفية توفير واجهات للوصول إلى خدماته وإمكانية إضافة هذه الواجهات لها، وأيضاً سنعرف الصيغة (syntax) و كيفية استخدامها. وهنالك شرح يوضح معاني الخدمات الموفرة (semantics).

في القسم الثاني سنتاول مفهوم التوسيعة لإضافة مكونات جديدة إليه، أو استبدال مكونات موجودة أو تحديثها دون أن يؤثر هذا على المكونات الأخرى بالنظام. وفي القسم الثالث في مفهوم المرونة، وسنتعرف على كيفية التسهيل لتهيئة النظام بمكونات مختلفة قد تكون من مطورين مختلفين.

أهداف الوحدة

عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

1. تعرف القصد من مفهوم الانفتاحية في النظم الموزعة.
2. تفهم كيفية توفير خدمة الويب للانفتاحية.
3. تبين المقصود بأن تكون الواجهات كاملة ومحايدة.

1. الانفتاحية

من أهداف الأنظمة الموزعة الانفتاح على العالم الخارجي. والمقصود بالانفتاح هو مقدرة النظام الموزع على توفير واجهات للوصول إلى خدماته وإمكانية إضافة مكونات جديدة للنظام من خلالها. هذه الواجهات لها صيغة (syntax) توضح كيفية استخدامها وهنالك شرح يوضح معاني الخدمات الموفرة (semantics) .

لابد من نشر واجهات الخدمات للمستخدمين ليستطيعوا معرفة الخدمات الموجودة بالنظام ومعرفة كيفية استخدامها.

هذا يُمكّننا من إضافة مكونات جديدة للنظام تستطيع أن تتكامل مع مكونات النظام الموجودة. و هنا لابد من معالجة التباين بين المكونات الجديدة والموجودة أصلاً بالنظام.

مثال

في شبكات الحاسب توجد أسس و معايير تحدد شكل الرسالة و محتواها ومعناها، هذه المعايير جمعت فيما يسمى البروتوكول.

1.1صيغة الخدمات

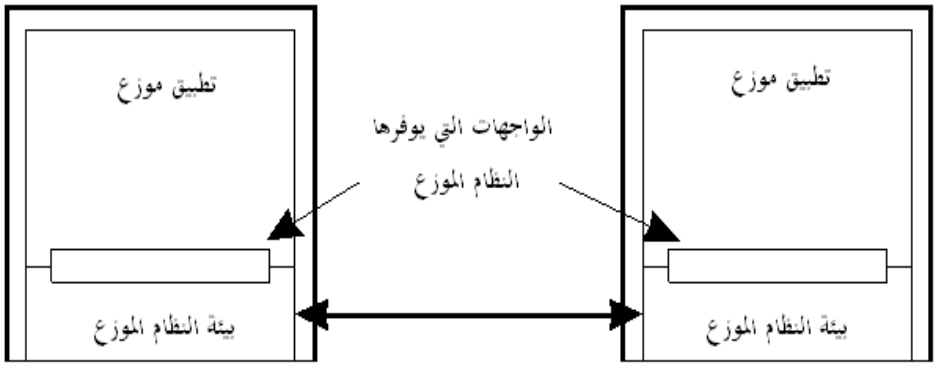
في الأنظمة الموزعة توصف الخدمات عبر واجهات (interfaces) هذه الواجهات تُعرف بصيغة الخدمات (syntax) مثل:

- أسماء الدوال المتاحة.
- نوع المعاملات (parameters).
- القيم الراجعة من الدوال (return values).
- الاستثناءات (exceptions) التي قد تحدث (raised).

تعريف المستخدم بالخدمات الموجودة بالنظام وكيف يتعامل معها يعتبر مشكلة (كيف نوصل له هذه التعريفات والمعلومات)، لذلك نمد المستخدم أحياناً بها عن طريق

لغتنا الطبيعية، فمثلاً نقول للمستخدم أن النظام به خدمة كذا وخدمة كذا، وإذا أردت الخدمة الفلانية يمكنك طلبها بالصيغة الفلانية.

يمكن لبرنامج بنظام موزع طلب خدمة من برنامج بنظام موزع آخر، في هذه الحالة نقول أن البرنامج الأول (العملية الأولى) خاطب البرنامج الثاني (العملية الثانية) من خلال الواجهة التي يوفرها الأخير.



شكل رقم (7-1): واجهات النظام الموزع.

مثال

تعتبر خدمات ويب (web services)مثالاً حّياً للخدمات المتاحة للمبرمجين عبر الويب، فيستطيع المستخدم بناء وحدة برمجية بها دوال تكون متاحة للكل كخدمات يمكن. أيّ شخص يستطيع الاستفادة من هذه الخدمات استخدامها إذا توافر لديه:

- عنوان الخدمة بالإنترنت (URL).
- طريقة استدعاء الطرق (أسماء الدوال والمعاملات (parameters) والقيم الراجعة من الدوال).
- عمل كل طريقة (معنى كل خدمة).
- توفر WDSL واجهات خدمة الويب للمستخدمين.

تدعم كل من لغة جافا ولغات فيجوال إستديو دوت نيت خدمات الويب (web services).

يمكن لأكثر من جهة بناء تطبيق باستخدام نفس الواجهات، ففي المثال السابق يستطيع أكثر من مستخدم بناء تطبيق على خدمة ويب معينة، فيكون لدينا العديد من التطبيقات تستخدم نفس خدمة ويب مما يوفر علينا إعادة كتابة الشفرة (code reuse).

إذا تم تعريف الواجهات بصورة جيدة؛ فقد تخدم هذه الواجهات العديد من التطبيقات، اما إذا لم تعرف الواجهات جيداً أو كانت غير كافية، (غير مكتملة) فقد يحتاج المستخدم إلى إضافة دوال (خدمات) من عنده لمعالجة النقص الموجود بالواجهات.

2.1 شروط الواجهات التي توفر خدمات لمستخدمين أن تكون:

1.2.1 كاملة (complete).

كاملة معناها أن كل ما يحتاجة المستخدم لبناء تطبيقه يكون قد تم تحديده، أم إذا كانت التعريفات ناقصة، ستجبر المطور على إضافة تفاصيل للتطبيق من عنده.

2.2.1 محايدة (**neutral**).

محايدة تعني أن على المواصفات التي وضعت بالواجهات ألا تكون محددة تماماً ، (تحدد كيف سيكون شكل التطبيق) وإنما يجب أن تكون فضفاضة وقابلة للتأقلم مع تطبيقات مختلفة.

خاصية الكمال، وخاصية الحياد، مهمتان لجعل التطبيقات المبنية على هذه الواجهات،

- متنقلة (portable) : يمكن نقلها من مكان لآخر دون تعديل.
- (interoperable) : يمكن لتطبيقين مختلفين العمل معاً .

متنقلة

إذا تم تطوير تطبيق موزع على نظام موزع معين، فيمكن تنفيذ هذا التطبيق في نظام موزع آخر دون إجراء أي تعديلات عليه، حيث يستخدم النظام الموزع الأول نفس الواجهات التي يستخدمها النظام الموزع الثاني.

(Interoperability)

مكونات مختلفة مصممة من قِبل شركات مختلفة يمكن أن توجد وتعمل معاً باستفادة كل مكون من الخدمات التي يوفرها المكون الآخر.

2. التوسعية

يجب أن يكون النظام المفتوح لديه القابلية لإضافة مكونات جديدة إليه، أو استبدال مكونات موجودة، أو تحديثها دون أن يؤثر هذا على المكونات الأخرى بالنظام. أي أن النظام الموزع يكون قابلاً للزيادة والتوسع (extendable).

3. المرونة

من أهداف النظام الموزع المفتوح المرونة، أي أنه من السهل تهيئة النظام بمكونات مختلفة قد تكون من مطورين مختلفين.

مثال

إذا كان النظام مرناً فيجب أن يكون لديه القدرة على إضافة أجزاء إليه تعمل على نظام تشغيل آخر مختلف (مكونات النظام تعمل على نظم تشغيل مختلفة)، أو يمكن استبدال نظام ملفات النظام بأكمله بنظام ملفات آخر.

1.3 الوصول للمرونة

للوصول إلى المرونة المنشودة، يجب على النظام الموزع المفتوح أن يقّسم إلى مكونات صغيرة سهلة الاستبدال وقابلة للتأقلم مع أي بيئة.

طريقة تقسيم النظام إلى مكونات صغيرة تعتبر طريقة حديثة، حيث كانت الطريقة القديمة هي مكون واحد كبير مقسم داخلياً (منطقياً) إلى مكونات صغيرة غير منفصلة عن بعضها البعض، لذلك تعديل جزء من هذه الأجزاء المنطقية يؤثر على بقية مكونات النظام ككل مما يؤدي إلى انغلاق النظام.

مثال

عملية وضع نسخ من الصفحات في جهاز العميل تسمى عملية التخبئة (caching) وهذه العملية تجعل استعراض الصفحات التي تم فتحها مسبقا سريعة جداً، ذلك لأن المتصفح:

- حين يفتح صفحة من الإنترنت يتم تحميلها في جهازك.
- يضع نسخة من الصفحة بالقرص المحلي بجهازك.
- عندما تكرر طلب نفس الصفحة بعد فترة وجيزة سيحضرها المتصفح من قرصك المحلي.
- يكون إحضارها سريعاً لأنه لا يتطلب اتصالاً بالإنترنت.
- يمكنك تصفحها وأنت "**offline**". غير متصل

يمكنك التحكم بسعة القرص الصلب الذي يستخدم في تخزين هذه الصفحات (سعة الكاش).

- هل على المستعرض مقارنة الصفحات التي بالقرص المحلي (بالكاش) مع التي بمخدم الويب (الصفحات الأصلية) وهل هي متطابقة (consistence)؟
- عدم الاهتمام بتطابق الصفحات في الجلسة الواحدة (session).

هذه العمليات يستطيع المستخدم التحكم بها عبر بعض الاعدادات بالمتصفح.

عملي

وضح كيف يمكننا التحكم بالنقاط أعلاه في متصفحك الذي تعمل عليه ؟

2.3 الأشياء التي لا يستطيع المستخدم التحكم بها مثل :

- إلى متى تظل الصفحات مخزّنة بالكاش؟
- أي صفحة يجب أن نحذف من الكاش في حال امتلائها؟
- كيف يمكننا تخزين بعض الصفحات التي تحتوي معلومات معينة وتجاهل بقية الصفحات؟
- كيف يمكننا عدم تخزين الصفحات التي لانريد؟

قد يحتاج المستخدم تخزين بعض الصفحات التي تحتوي دروس علمية بالكاش لأنها ثابتة، و غير متغيرة المحتوى (صفحة ساكنة (static))، فاستعراضها من الكاش مرات ومرات لا يؤثر بشيء فهذه يمكن تخزينها بالكاش لأطول مدة ممكنة، أي ليس هنالك مشكلة مع الصفحات الساكنة والتي لا يتغير محتواها.

المشكلة في استخدام الكاش تأتي من الصفحات التي تتغير باستمرار وأحياناً في جزء من الثانية، فهذه تخزينها بالكاش لا يفيد بشيء، فإذا استعرضتها الآن و طلبتها بعد رمشة عين ستجد محتواها غير متطابق مع محتوى الصفحة الأصلية بالإنترنت. مثال لهذه الصفحات، صفحات أسعار الأسهم، حركة السيارات في طريق سريع معين وغيرها.

من هنا نخلص للآتي:

- الصفحات الساكنة (غير متغيرة المحتوى) يمكن تخزينها بالكاش وإحضارها من الكاش دون أن يؤثر هذا في المعلومات المعروضة.
- الصفحات المتغيرة باستمرار لا يمكن تخزينها بالكاش واستعراضها من الكاش لا يعطي معلومات صحيحة.

إذن ما هو الحل؟

لوضع حل يناسب كل أنواع الصفحات في مثال الكاش أعلاه، تصميم النظام ليكون مرناً بما فيه الكفاية، حيث يمكننا حل مشكلة الكاش و ذلك بجعل المتصفح:

- يوفر عملية تخزين الصفحات (التخبيئة).

- يدع المستخدم يتحكم في الطريقة، مثل التحكم في نوعية الصفحات المراد تخزينها و مدة تخزين كل صفحة.

يمكن تطبيق هذه الفكرة عملياً على المتصفح بوضع معاملات (parameters) تسمح للمستخدم بتعبئتها ديناميكياً (و حسب رغبته).

لإضافة هذه الفكرة على المتصفح يجب على المطور إضافة مكونات تسمح للمستخدم بتطبيق خصوصياته، ولكي تعمل هذه المكونات لابد لها من واجهات لتمرر بيانات المستخدم عبرها للمكونات التي تقوم بتفيذ خصوصية المستخدم حسب المعاملات (parameters).

أسئلة تقويم ذاتي



1. ماذا نقصد بالانفتاحية ؟
2. كيف توفر خدمة الويب الانفتاحية ؟
3. ماذا نعني بأن الواجهات لابد من أن تكون كاملة ومحايدة ؟

الخلاصة

عزيزي الدارس تعرفنا في هذه الوحدة على أن المقصود بالانفتاح هو مقدرة النظام الموزع على توفير واجهات للوصول إلى خدماته وإمكانية إضافة مكونات جديدة للنظام من خلالها. هذه الواجهات لها صيغة (syntax) توضح كيفية استخدامها الخدمات الموفرة (semantics) ومن أمثلة هذه الواجهات: أسماء الدوال المتاحة. ونوع المعاملات (parameters).والقيم الراجعة من الدوال (return values).والاستثناءات (exceptions) التي قد تحدث (raised).

وتعرفنا على أنه يجب أن يكون النظام المفتوح لديه **القابلية لإضافة مكونات جديدة** إليه أو استبدال مكونات موجودة أو تحديثها دون أن يؤثر هذا على المكونات الأخرى بالنظام. أي أن النظام الموزع يكون قابلاً للزيادة والتوسع (extendable) وأيضا عرفنا أن من أهداف النظام الموزع المفتوح المرونة، أي أنه من **السهل تهيئة النظام بمكونات مختلفة** قد تكون من مطورين مختلفين.

إذا كان النظام مرناً فيجب أن يكون لديه القدرة على إضافة أجزاء إليه تعمل على نظام تشغيل آخر مختلف (مكونات النظام تعمل على نظم تشغيل مختلفة)، أو يمكن استبدال نظام ملفات النظام بأكمله بنظام ملفات آخر.

لمحة مسبقة عن الوحدة القادمة

عزيزي الدارس سنلتقي بك في الوحدة القادمة والتي ستتناول مفهوم التوسعية وعلاقة شبكة الإنترنت به والتي تعتبر أوسع وأكبر شبكة على ظهر البسيطة، وبالتالي هي تربط الداني بالقاصي والقريب بالبعيد، عبر شبكات متعددة وأجهزة متنوعة وبرامج متباينة.

وسنتعرف على أنواع التوسع في الحجم بإمكانية زيادة عدد المستخدمين للنظام وإضافة الموارد بكل سهولة ويسر وكذلك سنتعرف على التوسع جغرافياً وأمكانية أن يكون مستخدمو النظام وموارده متفرقة جغرافياً وفي أماكن متباعدة عن بعضها البعض و التوسع إدارياً بإمكانية إدارة أجزاء النظام المتباعدة والمتفرقة، حتى ولو كانت لها إدارات مستقلة.

وأيضا نتناول تكدس الخدمات كلها في جهاز واحد مركزي، بحيث يخدم هذا الجهاز كل عناصر النظام الموزع يولد إزدحاماً حول هذا الجهاز وقد يقود إلى اختناقه. ونتعرف كذلك على محدودية ومركزية الموارد في الأنظمة الموزعة هي السبب وراء مشاكل التوسع.

مسرد المصطلحات

المرونة flexibility

مـن أهـداف النظـام المـوزع المفتـوح المرونـة، أي أنــه مـن السـهل تهيئـة النظـام بمكونات مختلفة قد تكون من مطورين مختلفين.

التوسعية scalability

يجب أن يكون النظام المفتوح لديه القابلية لإضافة مكونات جديدة إليه أو استبدال مكونـات موجـودة أو تحـديثها دون أن يـؤثر هذا علـى المكونـات الأخـرى بالنظـام. أي أن النظام الموزع يكون قابلاً للزيادة والتوسع (extendable).

الصفحات الساكنة **Statstic page**

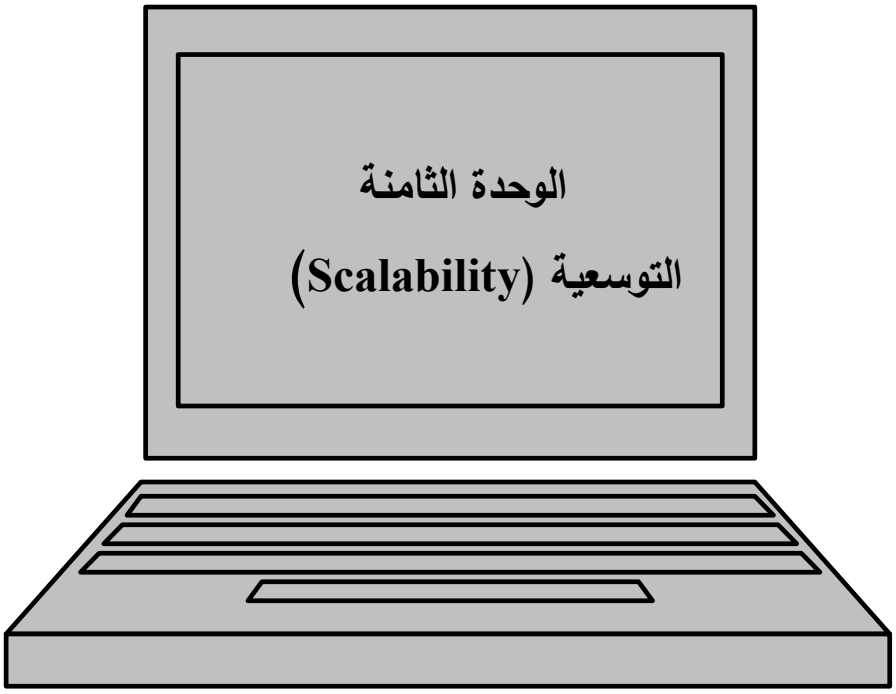
(غير متغيرة المحتوى) يمكن تخزينها بالكاش وإحضارها من الكاش دون أن يؤثر هذا في المعلومات المعروضة.

الصفحات المتغيرة **Dynamic page**

وهي الصفحات المتغيرة المحتوي ولا يمكن تخزينها بالكاش واستعراضـها من الكاش لا يعطي معلومات صحيحة.

المراجع

1- 1.Coulouris, G. et al., (2004), Distributed Systems, Concepts and Design, Addison Wesley
2- Farley, J. (1998), Java Distributed Computing, O'reilly & Associates
3- Fedorov, A, (2002), A Programmer Guide to .NET, Addison Wesley
4- Ince, D. (2002), Developing Distributed and E-Commerce Applications, Pearson Education
5- Liu, M. (2004), Distributed Computing Principles and Applications, Pearson Education
6- Schonauer, W. (1998), Scientific Supercomputing: Architecture and use of Shared and Distributed Memory Parallel Computers, Willi Schonauer, Germany
7- Tanenbaum, A., (2002), The Amoeba Distributed Operating System, Vrije Universiteit Technical Report, URL ftp://ftp.cs.vu.nl/pub/amoeba/Intro.ps.Z
8- Tanenbaum , A. AND Van Steen, M. (2002), Distributed Systems Principles and Paradigms, Prentice Hall
9- Tanenbaum, A. (1995), Distributed Operating Systems, 1/e, Prentice Hall, USA
10- Osman, A., (may 2006), A Multi-level Web-based Parallel Processing System, TRANSACTIONS ON ENGINEERING, COMPUTING AND TECHNOLOGY VOLUME 13 MAY 2006 ISSN 1305-5313
11-Forouzan, A. (2007), Data Communications and Networking, Mc Graw Hill, Fourth Edition
12-Tanenbaum, A., (2001), Modern Operating Systems, Prentice Hall, Second Edition
13-Tanenbaum, A., (1999), Structred Computer Organization, Prentice Hall, Fourth Edition.


الوحدة الثامنة
التوسعية (Scalability)

محتويات الوحدة

الموضوع	الصفحة
المقدمة	**155**
تمهيد	**155**
أهداف الوحدة	**156**
1. التوسعية	**157**
2. أنواع التوسع	157
3. مشاكل التوسع (scalability problems)	158
1.3 المشاكل الناجمة عن التمدد في الحجم	158
2.3 مشاكل التمدد الجغرافي	162
3.3 مشاكل إدارية	164
4. التخبئة (cashing)	**167**
1.4 عيوب التكرار (replication) والتخبئة (caching)	167
الخلاصة	169
لمحة مسبقة عن الوحدة القادمة	170
مسرد المصطلحات	171
المراجع	172

المقدمة

تمهيد

عزيزي الدارس،

مرحباً بك في الوحدة الثامنة من هذا المقرر والتي تأتيك بعنوان" التوسعية ". تنقسم هذه الوحدة إلى ستة أقسام وتعتبر شبكة الإنترنت تعتبر أوسع وأكبر شبكة على ظهر البسيطة، وبالتالي هي تربط الداني بالقاصي والقريب بالبعيد، عبر شبكات متعددة وأجهزة متنوعة وبرامج متباينة.

وسنتعرف معاً على**أنواع التوسع** في الحجم بإمكانية زيادة عدد المستخدمين للنظام وإضافة الموارد بكل سهولة ويسر، وكذلك سنتعرف على التوسع جغرافياً وأمكانية أن يكون مستخدمو النظام وموارده متفرقة جغرافيا وفي أماكن متباعدة عن بعضها البعض و التوسع إدارياً بإمكانية إدارة أجزاء النظام المتباعدة والمتفرقة، حتى ولو كانت لها إدارات مستقلة.

وفي القسم الرابع نتناول تكدس الخدمات كلها في جهاز واحد مركزي ، وخدمة الجهاز لكل عناصر النظام الموزع يولد ازدحاماً حول هذا الجهاز، وقد يقود إلى اختناقه.

وفي القسم الخامس نتعرف أنَّ محدودية ومركزية الموارد في الأنظمة الموزعة هي السبب وراء مشاكل التوسع. وفي القسم الأخير نتاول التخبئة كنوع خاص من التكرار.

أهداف الوحدة



عزيزي الدارس بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

- تشرح أنواع التوسع الثلاث.
- تعرف مشاكل التمدد الجغرافي .
- تفهم المشاكل الناتجة عن التوسع في الحجم.
- تبين الخوارثيمات المركزية عن اللامركزية.
- تعرف عيب التخبئة (caching).

1. التوسعية

عزيزي الدارس، شبكة الإنترنت تعتبر أوسع وأكبر شبكة على ظهر البسيطة، وبالتالي هي تربط الداني بالقاصي والقريب بالبعيد، عبر شبكات متعددة وأجهزة متنوعة وبرامج متباينة. فلا ريب إذن من أن تُسمَّى شبكة الشبكات.

هذا التوسع الذي بلغت به الإنترنت الآفاق، واستمرارها في التوسع بسرعة البراق جعلت مصممي الأنظمة الموزعة لطريقتها في استراق، ولتوسعها في التحاق.

سر نجاح الإنترنت **هو قابليتها للتوسع** وبالتالي لتنجح النظم الموزعة لابد **من السير في نفس الطريق، طريق التوسعية.**

2. أنواع التوسع

التوسع يمكن أن يكون في الحجم أو في المكان أو في الإدارة كما موضح أدناه:

- **التوسع حجمياً:** إمكانية زيادة عدد المستخدمين للنظام وإضافة الموارد بكل سهولة ويسر.
- **التوسع جغرافياً:** يمكن أن يكون مستخدمو النظام وموارده متفرقة جغرافيا وفي أماكن متباعدة عن بعضها البعض.
- **التوسع إدارياً:** إمكانية إدارة أجزاء النظام المتباعدة والمتفرقة، حتى ولو كانت لها إدارات مستقلة.

توسع النظام أحياناً يكون على حساب الكفاءة.

3. مشاكل التوسع (scalability problems)

توسع النظام قد يتسبب في مشاكل لابد من معالجتها نذكر منها:

1.3 المشاكل الناجمة عن التمدد في الحجم

توسع الموارد والمستخدمين قد يجابه بمعضلة هي المركزية:

1. مركزية الخدمات. 2. مركزية البيانات. 3.مركزية الخوارزميات.

1.1.3 مركزية الخدمات

تكدس الخدمات كلها في جهاز واحد مركزي ، وبخدمة هذا الجهاز لكل عناصر النظام الموزع يولد ازدحاماً حول هذا الجهاز وقد يقود إلى اختناقه.

إذا جعلنا هذا الجهاز عالي الكفاءة وسريعاً وبه ذاكرة تخزين ضخمة ليستطيع خدمة الكل؛ فسنجد أن هنالك مشكلة في الوصول إليه (اختناق في منافذه التي من خلالها نصل إليه).

الحل هو عدم استخدام مخدم واحد بالنظام الموزع، وعمل نسخ مكررة من الخادمات (replication) . ولكن قد تجد بعض النظم الموزعة أنها لا تستطيع تفادي استخدام الخادم الواحد المركزي، مثل الخادمات التي تخزن أرقام حسابات عملاء بنك معين أو التي تحفظ بها السجلات الطبية للمرضى بمستشفى معين، هذه المعلومات تعتبر مهمة ولابد من توافر البرامج التي تعالج هذه المعلومات بمخدم واحد مركزي عالي التأمين، معزول في غرفة منفصلة مثلاً ، محمي عن بقية أجزاء النظام الموزع بعتاد شبكات خاص.

إذا أردنا عمل نسخ متعددة من هذا المخدم (replicate the server) في مواقع مختلفة لتحسين الأداء وفك الاختناق سنحتاج إلى تأمين لكل هذه النسخ من المخدمات، قد تصبح هذه المخدمات عرضة للاختراقات.

2.1.3 مركزية البيانات

مشكلة مركزية البيانات تشبه إلى حد كبير مشكلة مركزية الخدمات. مثلاً إذا كان لدينا ملف يحتوي سجلات عناوين وهواتف 50 مليون شخص ، وإذا افترضنا أن كل سجل سيحتاج إلى 50 حرفاً لتخزين بياناته بالقرص، سيكون لدينا ملف حجمه 2.5 غيغابايت في القرص.

إذا كانت البيانات متمركزة بملف واحد (قاعدة بيانات واحدة) فكيف سيتم التعامل معها؟

مثال **خدمة نطاق الأسماء**

Domain Name Services (DNS)

إذا كنت مستخدماً للإنترنت وتتصفح المواقع وتفتح رسائلك الإلكترونية فأنت زبون دائم لمخدمات نطاق الأسماء **(DNS)**، دون أن تعرف ذلك. فهذه المخدمات مهمة جداً لنا لكنها مخفية عنَّا.

نظام **DNS** هو نظام يتكون من أضخم قاعدة بيانات "موزعة" عرفتها المجرة. وبدون خدمة **DNS** سيصبح استخدام الإنترنت عسيراً وتذكر أسماء المواقع وعناوين البريد الإلكتروني صعباً .

ما رأيك لو كان هاتفك النقال ليس به خدمة الأسماء، كيف ستيمز بين رقم ورقم وكيف ستتعامل مع مئات الأرقام ؟

عندما تستخدم الإنترنت فإنك تستخدم أسماء سهلة الحفظ وذات مدلول معنوي، مثلا إذا كتبت **www.ous.edu.sd** فهذا عنوان موقع سهل الحفظ.

يحتوي داخل هذا العنوان "اسم نطاق **domain name**"، هذا العنوان لا يعرفه جهاز حاسوبي ولا برنامج المتصفح لدي ولا الخادم الذي يوجد به الموقع. فهذا الاسم له عنوان مرادف له (عنوان **IP**) وهو في شكل أرقام مثل **216.180.110.110**، وكلما كتبنا عنوان موقع نصي قام **DNS** بتحويله للعنوان الرقمي المقابل له (عنوان **IP**).

فمثلاً أثناء فتحك لبريدك الإلكتروني يمكن أن تكون تعاملت مع **DNS** مئات المرات.
فمهمة **DNS** الأساسية هي تحويل نطاقات الأسماء إلى عناوين **IP**. ولكن ...:

- هنالك بلايين من عناوين **IP**مستخدمة حالياً .
- هنالك بلايين الطلبات الواردة لخدمة **DNS**يومياً .
- هنالك يومياً مئات الملايين من الأشخاص الذين يستخدمون الإنترنت.
- كل شخص يمكنه إرسال مئات الطلبات لخدمة **DNS** (بصورة غير مباشرة).
- تتغير العديد من نطاقات الأسماء وعناوين **IP**يومياً .
- تنشأ نطاقات أسماء جديدة يومياً .

1. ملايين من البشر يقومون بأعمال تغير وتضيف نطاقات أسماء وعناوين **IP**يومياً .

من هذا نجد أن نظام **DNS** هو قاعدة بيانات لا تشبهها قاعدة بيانات في الكون (فريدة) من حيث:

- كثرة الطلبات التي ترد عليها.
- ملايين التعديلات اليومية التي تحدث بها.

كل عناوين الإنترنت التي نستخدم مخزنة في قواعد بيانات وتوجد عناوين **IP** مقابلة لها. إذا كانت كل هذه العناوين مخزنة بجدول واحد، أو قاعدة بيانات واحدة، أو في جهاز واحد ،أو البرنامج الذي يقوم بمعالجتها واحد ، فهذا يعني أنني إذا كتبت عنوان موقع فقد يحتاج DNS إلى مدة طويلة جداً ليعطيني عنوان IP المقابل له. وإذا تمركزت العناوين في مكان واحد فلن يستخدم الانترنت أحد بعد اليوم.

3.1.3 مركزية الخوارزميات

تمركز الخوارزميات مشكلة تضاف إلى المشاكل السابقة. والمقصود بتمركز الخوارزمياتهو تعاملها واحتياجها لكل المعلومات الموجودة بالنظام لتعمل عملاً معيناً . هذه المعلومات التي تحتاجها الخوارزمية لابد من تجميعها من أجزاء النظام عبر الشبكة،

فتجمع كلها بالجهاز الذي تعمل فيه الخوارزمية، هذا يسبب ازدحاماً بالشبكة لأن هذه المعلومات المطلوبة من قبل الخوارزمية سترسل للخوارثمية في شكل رسائل.

مثال

إذا أردت معرفة أسرع طريق لإرسال رسالة عبره، يمكنني استخدام خوارثمية معينة لمعرفة ذلك، مثلاً يمكنني استخدام الـ graph theory لحساب أفضل الطرق التي يمكن توجيه الرسائل عبرها.

المشكلة هنا أنه لتعمل الخوارزمية تحتاج إلى أن تجتمع لديها بيانات كل الرسائل الواردة والصادرة في الشبكة. ولتتجمع هذه البيانات في مكان واحد لتعالجها الخوارزمية نحتاج إرسالها عبر الشبكة لمكان واحد ، هنا نكون قد تسببنا في:

- ازدحام الشبكة بالرسائل (ارسال البيانات التي تحتاجها الخوارزمية لإكمال عملها).
- تكدس المعلومات في جهاز (مكان) واحد بالشبكة (تجمع البيانات في الجهاز الذي توجد به الخوارزمية).

لذلك يجب علينا تجنب استخدام خوارزميات مركزية ما أمكن.

ما هو الذي يميز الخوارثيمات المركزية عن اللامركزية ؟ هنالك بعض الصفات التي إذا طبقناها على الخوارزم أصبح لا مركزي منها:

- لا يحتاج جهاز معلومات كاملة عن حالة النظام ككل.
- تتخذ الأجهزة قراراتها بناًء على المعلومات المحلية.
- لا يؤثر عطل في جهاز على عمل الخوارزم.
- لا يفترض وجود توقيت زمني مشترك (موحد) (global clock).

2.3 مشاكل التمدد الجغرافي

1.2.3 عمل النظام الموزع في الشبكات المحلية (LANs) سريع وسهل ذلك لأن الشبكات المحليه سريعة. الاتصال بين أجزاء النظام الموزع تتم تزامنيا (synchronous communication)، أي إذا طلب جزء من النظام الموزع أمراً من جزء آخر بجهاز اخر سينتظر هذا الجزء الرد على طلبه، هذا الانتظار لا يتجاوز "جزء من ميكروثانية"، هذا في الشبكة المحلية.

ولكن كم من الزمن نحتاج لانتظار ردّ من جهاز بشبكة واسعة؟ نحتاج "عدة ميكروثانيات". لذلك فإن بناء تطبيقات تفاعلية باستخدام الاتصال المتزامن (synchronous) يحتاج عناية خاصة.

2.2.3 سبب آخر يجعل بناء النظام الموزع على الشبكات المحلية أيسر وأسهل من بنائه في الشبكات الواسعة؛ هو اعتمادية الاتصال في الشبكات المحلية. (reliable communication) ، فترسل الرسائل عن طريق البث (broadcasting) ، أما في الشبكة الواسعة فالارسال يكون نقطة لنقطة مما يعقد المسألة.

مثال

مثلا لايجاد "خدمة" في النظام، يرسل الجهاز الذي يبحث عن هذه الخدمة رسالة (يبثها)، فتصل كل الأجهزة، هذه الرسالة مفادها "من يمتلك الخدمة الفلانية"، ستصل الرسالة للكل، فيرد من يمتلك الخدمة برسالة مفادها "أنا، وهذا هو عنواني" ، أي سيرسل الجهاز الذي يمتلك الخدمة رداً للجهاز الذي طلب الخدمة به عنوانه.

في الشبكات الواسعة الطريقة أعقد من هذه؛ حيث يتم وضع كل الخدمات في موقع خاص بحيث يصل إليها الكل.

3.2.3 التمدد الجغرافي يقف عاجزا أمام مركزية المكونات من خدمات وغيرها، فيصبع تمديد النظام جغرافياً إذا كانت المكونات مركزية. لأن الكفاءة والاعتمادية يتأثران بشكل

مباشر بالتمدد الجغرافي، فستكون لدينا مشكلة الاختناق والاتصال بالموكونات المركزية. أيضاً تهدر عملية تمركز المكوناتجزءاً كبيراً من موارد الشبكة.

مثال

تخيل وجود مخدم بريد الكتروني (mail server) واحد بالدولة كلها، هذا يعني أن أي رسالة ترسل ستصل في البداية إلى المخدم أولاً . فاذا أرسلت رسالة الكترونية إلى جارك القريب، ستذهب أولاً إلى مخدم البريد المركزي البعيد (تقطع الفيافي والقفار)، لتقطع نفس المسافة مرة ثانية لتصل جارك القريب.

4.2.3 من مشاكل التوسع الجغرافي هي كيفية تمديده عبر مناطق مستقلة عن بعضها إداريا بها نزاعات سياسية أو مشاكل حول حدود إقليمية أوحروبات أهلية.

فقد لا تتفق هذه المناطق مع بعضها في الصرف المادي أو في الإدارة أو في طريق تأمين المعلومات.

مثال

إذاكانت مكونات النظام الموزع في **نطاق واحد** (single domain) فيمكن للمستخدمين من **التعامل داخل هذا المجال بموثوقية** (trusted)، في هذه الحال يمكن لمدير النظام أن يختبر ويستوثق من التطبيقات وفق معايير معينة ليتأكد من ان مكونات النظام لا يمكن التلاعب بها. فالمستخدمين يمكنهم الوثوق في مدير النظام. لكن هذه الثقة لا تمتد خارج هذا النطاق.

فإذا **توسع النظام ليمتد خارج هذا النطاق** فلا بد من أن نتخذ **مقاييس حماية أخرى** منها:

- على النظام الموزع أن يحمي نفسه من الهجمات التخريبية التي قد تأتي من المنطقة الأخرى (من خارج النطاق)، خاصة إذا كان لمستخدمي النظام الآخر صلاحيات معينة مفتوحة.
- يجب على المنطقة الأخرى حماية نفسها أيضاً ضد الهجمات التخريبية المحتملة من النظام الموزع، مثلاً قد يحمل النظام الموزع بريمجات جافا بواسطة المستعرض في أجهزة المنطقة الأخرى وينفذها على أجهزتهم.فغالباً لاتعرف ما يحدث من تنفيذ مثل هذه

الشفرات (الأكواد) والبريمجات الدخيلة عن طريق نظام الموزع، لذلك عادة تتخذ إجراءات تمنع هذه الشفرات والبرامج من الوصول إلى بعض الموارد الهامة مثل القرص الصلب، وتجعل الوصول لبعض الموارد محدوداً جداً وفي نطاق ضيق.

3.3 مشاكل إدارية

محدودية ومركزية الموارد في الأنظمة الموزعة هي السبب وراء مشاكل التوسع.

هنالك ثلاث طرق للتوسع هي:

- إخفاء تأخر الرد (hiding communication latencies).
- **التوزيع** (distribution).
- المماثلة (replication).

1.3.3 إخفاء تأخر الرد

المقصود بإخفاء تأخر الرد واضح. فالعميل قد يطلب أمراً ما من مخدم بعيد، ويكون منتظراً للرد، الذي قد يتأخر بسبب زحمة الاتصالات أو بطئها. لنخفي هذا التأخير سنجعل العميل ينشغل بأعمال أخرى يؤديها حتى يصله الرد (busy waiting). عند وصول الرد سيتم قطع عمل العميل وتسليمه الرد. أي أن الاتصال هنا هو اتصال لامتزامن (asynchronous communication).

حاول تجنب انتظار الردود (responses) على طلبات الخدمات البعيدة (remote requests service) بقدر ما يمكن.

هنالك بعض التطبيقات التي لا تريد تطبيق الاتصال غير المتزامن لأن لا شيء لديها أهم من وصول الرد على طلبها، في هذه الحالة ليس لدينا شئ نفعله سوى محاولة تقليل زمن الاتصال الكلي بقدر ما نستطيع.

مثال لتقليل الزمن الكلي للاتصال

إذا أراد عميل تعبئة نموذج مرتبط بقاعدة بيانات في الخادم ، فسيتم ذلك كالآتي:

- تعبئة حقل من حقول النموذج.
- إرسال الحقل للمخدم.
- يقوم المخدم باختبار الحقل وتصحيح الأخطاء إن وجدت.
- الرد للعميل بقبول الحقل.
- تكرار الخطوات من (1−4) لبقية الحقول.

لتقليل هذا الزمن نقوم بترحيل الكود الذي يقوم باختبار الحقول وتصحيحها من الخادم إلى جهاز العميل حيث يوجد النموذج، فتتم عملية اختبار الحقول وتصحيحها (validation) بجهاز العميل و الحاجة للاتصال بالمخدم. ثم ترسل كل الحقول دفعة واحدة للمخدم لمعالجة تخزينها بقاعدة البيانات. بهذه الطريقة نكون قد قللنا زمن الاتصال بتقليل الرسائل المتبادلة بين العميل والخادم.

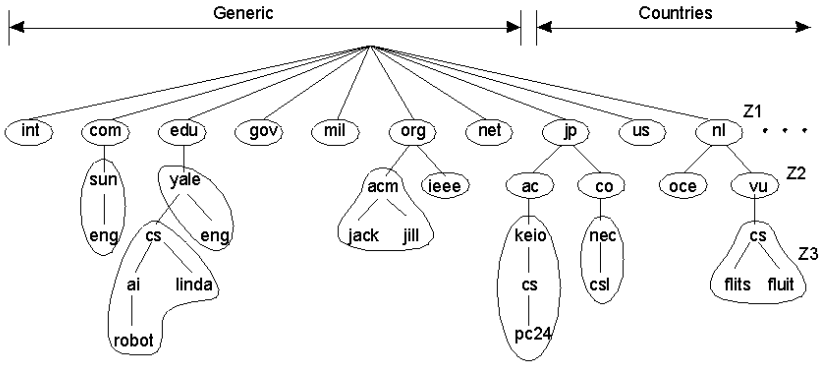
هذه الطريقة مستخدمة الآن وبكثرة في الويب، حيث يمكن كتابة كود اختبار حقول النموذج المدخلة وتصحيحها (validation) بواسطة لغات برمجة الإنترنت التي تعمل في جانب العميل، مثل بريمجات الجافا (java applets). فيحمل كود الاختبار والتصحيح من المخدم إلى جهاز العميل وينفذ فيه بواسطة المتصفح.

2.3.3 التوزيع (distribution)

تقسيم المكونات الكبيرة الى أجزاء صغيرة ثم توزيعها على أجهزة النظام.

مثال

مثلاً DNS يتم تنظيمه في شكلٍ هرمّي حيث يقسم النطاق حسب المناطق (zones) ، فتعالج كل أسماء منطقة لوحدها بواسطة مخدم أسماء منفرد (single name server).



مثال

تعتبر الويب مخزناً ضخماً للوثائق (الصفحات)، وكل وثيقة لها عنوان مفرد غير متكرر. هل كل هذه الوثائق مخزّنة في مكان واحد (جهاز واحد) ؟ طبعاً لا ، وإنما موزعة في أجهزة عديدة. تخزن هذه الوثائق في كم هائل من الأجهزة (مخدمات الويب)، وتنتشر هذه المخدمات في أماكن متفرقة وبلاد متعددة.

رغم اختلاف أماكن الوثائق ، يتم استدعاؤها بطريقة واحدة من أي مكان (شفافية المكان).

تخيل لو لم يتم "توزيع" هذه الوثائق عبر العديد من مخدمات الويب وكانت متمركزة في جهاز واحد، كيف سيكون التعامل معها ؟ الحمد لله على نعمة التوزيع وفوائده العديدة ومنها التسريع.

3.3.3 التكرار (replication)

قد تؤثر التوسعية (scalability) على الكفاءة. أحد الحلول لتحسين الأداء هو تكرار الموارد (عمل نسخ عديدة منها).**هذا يفيد في الآتي:**

- زيادة توافر الموارد.
- توزيع الحمل والأعباء بين النسخ المكررة (balancing the load).
- وجود نسخ قريبة من المستخدم (خاصة في النظم الواسعة والممتدة جغرافياً).

4. التخبئة (cashing)

التخبئة هي نوع خاص من التكرار (replication) وليس هنالك فرق جلي بينهما، حيث تقوم التخبئة مثل التكرار بعمل نسخ متعددة من المورد.

الغرض الرئيسي من التخبئة هو وصول العميل للمورد المخبأ بسرعة.

الفرق بين التخبئة والتكرار أن التخبئة تُحدَّد بواسطة المستخدم، فهو الذي يحدد هل يريد أن تكرر البيانات مثلاً أم لا، بينما التكرار يُحدَّد بواسطة مالك المورد. أما في التخبئة فليس للمالك فيها دور.

1.4 عيوب التكرار (replication) والتخبئة (caching)

أكبر عيب للتكرار والتخبئة (وهو مرتبط دوماً بهما) هو مشكلة تطابق البيانات (consistency). فإذا ما عدلت نسخة من النسخ المكررة، أصبحت هذه النسخة مختلفة عن بقية النسخ، وبالتالي لابد من مراعاة هذا الأمر ومحاولة جعل كل النسخ متشابهة في المحتوى. مشكلة عدم التطابق أحياناً يمكن تجاهلها وفي معظم الأحايين لا يمكن تجاهلها. مثلاً يمكن تجاهل حالة التطابق إذا كنت متصفحاً لمواقع ويب غير متغيرة المحتوى (ساكنة)، ففي هذه الحالة إذا وجدت نسخاً من الصفحات بجهاز العميل (مخبأة فيه) يمكن

تصفحها وكأنها أُحضِرت من الإنترنت الآن، ذلك لأن المحتوى غير متغير وبالتالي ليس هنالك فرق بين الصفحات المخبأة بجهاز العميل والصفحات التي بالإنترنت.

أما إذا كنت تتصفح موقعاً عن أسعار الأسهم أو حركة السير في طريقٍ ما، فلا يمكنك تجاهل التطابق، لأن الصفحات الأصلية (بالإنترنت) قد تتغير كل ثانية وبالتالي فإنّ استعراض الصفحات المخزنة بجهاز العميل قد يمدك بمعلومات غير صحيحة، وبالتالي الصفحات المخزنة من هذا النوع تكون دائما غير مطابقة للصفحات الأصلية بالإنترنت، هنا لا تفيد التخبئة في شيء.في هذه الحالة نحتاج درجةً عاليةً من التطابقية.

أسئلة تقويم ذاتي



1. اذكر أنواع التوسع الثلاث، مع توضيح كل نوع.
2. ما مشاكل التمدد الجغرافي ؟
3. ما المشاكل الناتجة عن التوسع في الحجم ؟
4. ما الذي يميز الخوارزميات المركزية عن غير المركزية ؟
5. ما عيب التخبيئة (caching) ؟

الخلاصة

عزيزي الدارس، تتلخص هذه الوحدة في عدة نقاط أهمها:

سر نجاح الإنترنت هو **قابليتها للتوسع**، وبالتالي لتنجح النظم الموزعة لابد من السير في نفس الطريق، طريق التوسعية.و**أنواع التوسع** في **الحجم** بإمكانية زيادة **عدد المستخدمين** للنظام **وإضافة الموارد** بكل سهولة ويسر. وكذلك تعرفنا على التوسع **جغرافياً**، وأمكانية أن يكون مستخدمو النظام وموارده متفرقة جغرافيا وفي أماكن متباعدة عن بعضها البعض، و التوسع **إدارياً**بإمكانية إدارة أجزاء النظام المتباعدة والمتفرقة، حتى ولو كانت لها إدارات مستقلة.

وأن **تكدس الخدمات** كلها في جهاز واحد مركزي **(**بحيث يخدم هذا الجهاز كل عناصر النظام الموزع**)** يولد إزدحاماً حول هذا الجهاز وقد **يقود إلى اختناقه.**

كما عرفنا أن محدودية ومركزية الموارد في الأنظمة الموزعة هي السبب وراء مشاكل التوسع. وأخيرا تناولنا التخبئة كنوع خاص من التكرار.

لمحة مسبقة عن الوحدة القادمة

عزيزي الدارس، في الوحدة القادمة سنتعرف إنْ شاء الله

على فكرة الكائنات الموزعة التي تتبني على نداء الإجراء البعيد (RPC)، ثم نرى كيف يطبق مفهومه على الكائنات الموزعة. واستعمال نداء الإجراء البعيد في تقنية الخادم/العميل، فهنالك برنامج بجهاز يطلب تنفيذ إجراء بجهاز آخر، حيث يعتبر البرنامج الذي يطلب الخدمة عميلاً، والبرنامج الذي ينفذ الخدمة خادماً. وكذلك سنتعرف على النداء البعيد غير المتزامن عند استدعاء الإجراء البعيد المتزامن، و سنبحث أيضا في مفهوم الكائنات الموزعة. عزيزي الدارس نتمنى لك التوفيق والسداد.

مسرد المصطلحات

مشاكل التوسع (scalability problems)

توسع النظام قد يتسبب في مشاكل لابد من معالجتها نذكر منها:

مركزية الخدمات

تكدس الخدمات كلها في جهاز واحد مركزي، بحيث يخدم هذا الجهاز كل عناصر النظام الموزع يولد ازدحاما حول هذا الجهاز، وقد يقود إلى اختناقه.

التكرار (replication)

قد تؤثر التوسعية (scalability) على الكفاءة. أحد الحلول لتحسين الأداء هو تكرار الموارد (عمل نسخ عديدة منها).

التخبئة (cashing)

التخبئة هي نوع خاص من التكرار (replication) وليس هنالك فرق جلي بينهما، حيث تقوم التخبئة مثل التكرار بعمل نسخ متعددة من المورد.

المراجع

1- 1.Coulouris, G. et al., (2004), Distributed Systems, Concepts and Design, Addison Wesley
2- Farley, J. (1998), Java Distributed Computing, O'reilly & Associates
3- Fedorov, A, (2002), A Programmer Guide to .NET, Addison Wesley
4- Ince, D. (2002), Developing Distributed and E-Commerce Applications, Pearson Education
5- Liu, M. (2004), Distributed Computing Principles and Applications, Pearson Education
6- Schonauer, W. (1998), Scientific Supercomputing: Architecture and use of Shared and Distributed Memory Parallel Computers, Willi Schonauer, Germany
7- Tanenbaum, A., (2002), The Amoeba Distributed Operating System, Vrije Universiteit Technical Report, URL ftp://ftp.cs.vu.nl/pub/amoeba/Intro.ps.Z
8- Tanenbaum , A. AND Van Steen, M. (2002), Distributed Systems Principles and Paradigms, Prentice Hall
9- Tanenbaum, A. (1995), Distributed Operating Systems, 1/e, Prentice Hall, USA
10- Osman, A., (may 2006), A Multi-level Web-based Parallel Processing System, TRANSACTIONS ON ENGINEERING, COMPUTING AND TECHNOLOGY VOLUME 13 MAY 2006 ISSN 1305-5313
11-Forouzan, A. (2007), Data Communications and Networking, Mc Graw Hill, Fourth Edition
12-Tanenbaum, A., (2001), Modern Operating Systems, Prentice Hall, Second Edition
13-Tanenbaum, A., (1999), Structred Computer Organization, Prentice Hall, Fourth Edition


الوحدة التاسعة
الكائنات الموزعة
Distributed Objects

محتويات الوحدة

الموضوع	الصفحة
المقدمة	175
تمهيد	175
أهداف الوحدة	175
1. نداء الإجراء البعيد	176
1.1 خطوات نداء الإجراء البعيد	176
2.1 وكيل العميل (client stub).	178
2. النداء البعيد غير المتزامن (synchronous)	180
3. الكائنات الموزعة (distributed objects)	181
1.3 عندما يريد كائن "أ" مهمة من كائن آخر "ب":	182
2.3 الكائن الموزع يختلف عن الكائن العادي في الآتي	183
3.3 أمثلة للكائنات الموزعة	184
الخلاصة	185
لمحة مسبقة عن الوحدة القادمة	185
إجابات التدريبات	186
مسرد المصطلحات	187
المراجع	175

المقدمة

تمهيد

عزيزي الدارس،مرحباً بك في الوحدة التاسعة من هذا المقرر والتي تأتيك بعنوان "الكائنات الموزعة" تنقسم هذه الوحدة إلى ثلاثة أقسام. في القسم الأول منها نتناول فكرة الكائنات الموزعة مبنية على نداء الإجراء البعيد (RPC)، ثم نرى كيف يطبق مفهومه على الكائنات الموزعة. واستعمال نداء الإجراء البعيد في تقنية الخادم/العميل، فهنالك برنامج بجهاز يطلب تنفيذ إجراء بجهاز آخر، حيث يعتبر البرنامج الذي يطلب الخدمة عميلاً ، بينما البرنامج الذي ينفذ الخدمة خادم. وفي القسم الثاني منها سنتعرف على النداء البعيد اللامتزامن عند استدعاء الإجراء البعيد المتزامن في القسم الثالث سنبحث في الكائنات الموزعة.

أهداف الوحدة

عزيزي الدارس بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

1. تفرق بين نداء الإجراء البعيد (RPC) ونداء الطريقة البعيدة (RMI).
2. تعرف الغرض الرئيسي من وجود الوكيل (stub).
3. تعدد خمسة أمثلة لطبقات تطبق مفهوم الكائنات الموزعة.
4. تبين الفروقات بين الكائن العادي والكائن الموزع.

عزيزي الدارس، إن فكرة الكائنات الموزعة مبنية على نداء الإجراء البعيد (RPC)، لذلك سنبدأ به أولاً ثم نرى كيف يطبق مفهومه على الكائنات الموزعة.

1. نداء الإجراء البعيد Remote Procedure Call (RPC)

هي طريقة تستخدم لطلب خدمة من برنامج في حاسوب بعيد على الشبكة دون الإلمام بفهم تفاصيل طريقة عمل الشبكة وأين يوجد البرنامج واسم الجهاز الذي يوجد به البرنامج والبروتوكول المستخدم في الاتصال ، وغيرها من تعقيدات الشبكات ومشاكل الاتصالات.

يستعمل نداء الإجراء البعيد في تقنية الخادم/العميل، فهنالك برنامج بجهاز يطلب تنفيذ إجراء بجهاز آخر، حيث يعتبر البرنامج الذي يطلب الخدمة عميل، بينما البرنامج الذي ينفذ الخدمة خادم.

يعتبر الإجراء البعيد من التقنيات القوّية التي تبنى عليها الأنظمة الموزعة، والتطبيقات المستندة على نموذج الخادم/العميل.

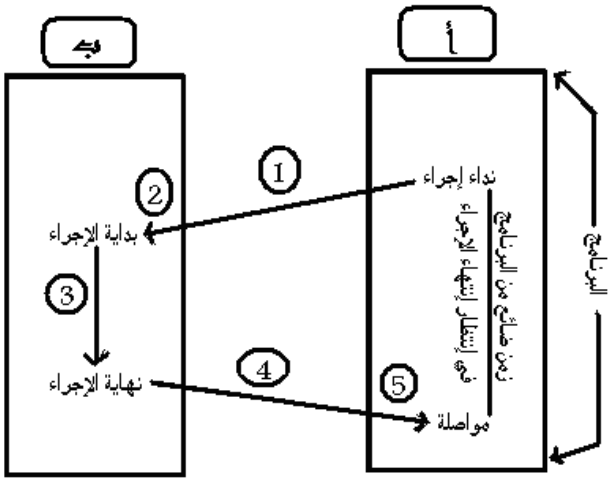
1.1 خطوات نداء الإجراء البعيد

افترض أن برنامج (عملية) "أ" تريد استدعاء إجراء بعيد في برنامج (عملية) أخرى "ب". قد يكون البرنامجان في جهازٍ واحدٍ أو في جهازين مختلفين، هذا لا يُفْرِقُ كثيراً، فبما أن العمليتين مختلفتان فالطريقة ستكون واحدة.

إليك **ما يحدث بالتفصيل بين البرنامجين (العمليتين):**

- يرسل البرنامج "أ" اسم الدالة المراد تنفيذها بالبرنامج "ب" وبياناتها (المعطيات) الضرورية لتنفيذ البرنامج "ب" ويتوقف "أ" في حالة انتظار على رده.
- يقوم "ب" باستلام الطلب مع المعطيات.
- ينفذ "ب" الطلب.

- يرسل "ب" النتائج للجهاز "أ".
- يستلم "أ" الرد الوارد من "ب" ثم يواصل عمله.



شكل رقم (9-1): نداء الإجراء البعيد

أثناء عملية نداء الإجراء البعيد قد تواجهنا مشاكل لابد من مراعاتها، مثل تباين الأجهزة واختلافها سواء أكان ذلك في العتاد وتمثيل البيانات أو في البرامج ونظم التشغيل.أيضاً توقع حدوث عطل في الشبكة أو في أحد الجهازين. توجد طرق عديدة لحل مثل هذه المشاكل، تفاصيل هذه المشاكل وطرق معالجاتها ليس موضوعنا الآن.

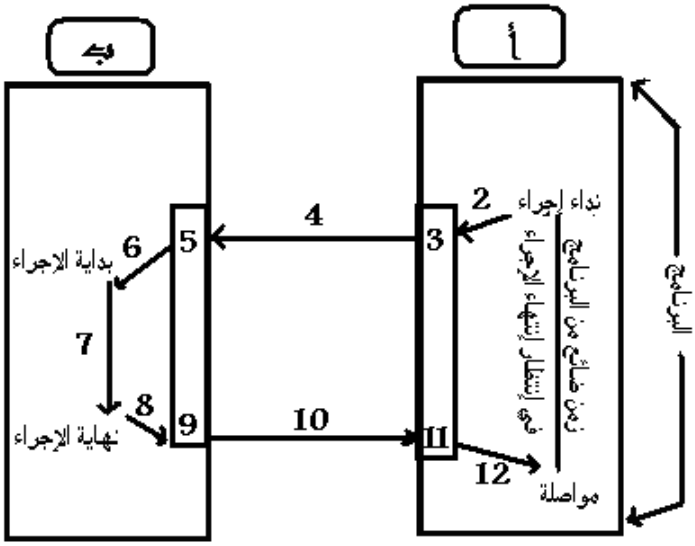
الهدف من وراء RPC هو جعلها تشبه النداء المحلي (local procedure call) وذلك بجعل البرامج لا تميز بين نداء الإجراء المحلي ونداء الإجراء البعيد (تحقيق الشفافية في نداء الإجراء).

2.1 وكيل العميل (client stub)

هل يتعامل برنامج العميل مباشرة مع برنامج الخادم أم هنالك وسيط ؟ نعم هنالك وسيط بين برنامج الخادم وبرنامج العميل، هذا الوسيط يسمى الوكيل (stub). هنا سنقوم بتعديل الخطوات أعلاه فيما يأتي من حديث:

1.2.1 كيف يتم نداء الإجراء البعيد في حضور الوكيل.

- توضع نسخة من الإجراء البعيد بجهاز العميل نسميها وكيل العميل (client stub).
- يقوم برنامج العميل باستدعاء هذه النسخة المحلية.
- يستلم وكيل العميل طلب النداء من برنامج العميل.
- لا ينفذ الوكيل النداء وإنما يحزمه في شكل رسالة ويرسلها للجهاز البعيد عن طريق نظام التشغيل، ثم ينتظر الرد.
- عندما تصل الرسالة للخادم يستلمها نظام التشغيل ويسلمها لوكيل الخادم (server stub)، الذي يمثل الواجهة المقابلة لوكيل العميل (client stub).
- يقوم وكيل الخادم بتحويل الطلب القادم من وكيل العميل إلى برنامج الخادم الذي يحتوي على الإجراء البعيد.
- ينفذ برنامج الخادم (الذي يحتوي فعلاً على الإجراء البعيد)الإجراء محلياً .
- يرجع برنامج الخادم النتائج لوكيله (server stub).
- يحزم وكيل الخادم النتائج في شكل رسالة.
- و يرسلها لوكيل العميل بالجهاز الآخر عن طريق نظام التشغيل.
- يستلم نظام التشغيل بجهاز العميل الرسالة ويمررها لوكيل العميل.
- يمرر وكيل العميل بدوره النتيجة لبرنامج العميل.



شكل رقم (9-2): تفاصيل نداء الإجراء البعيد

2.2.1 ملخص خطوات نداء الإجراء البعيد

في جهاز العميل:

- البرنامج مع الوكيل.
- الوكيل مع نظام التشغيل.
- نظام التشغيل مع نظام تشغيل جهاز الخادم.

في جهاز الخادم:

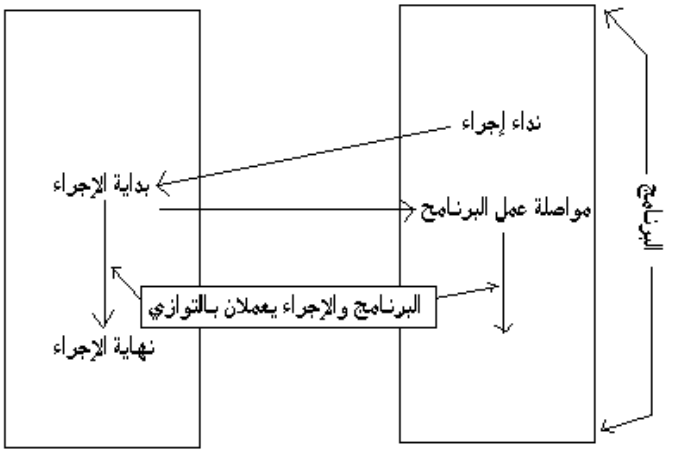
1. نظام التشغيل بالخادم مع الوكيل.
2. الوكيل مع البرنامج (الذي يحتوي النداء البعيد)، العكس صحيح.

2. النداء البعيد غير المتزامن (synchronous)

عزيزي الدارس، عند إستدعاء الإجراء البعيد المتزامن هذا يعني أن البرنامج الذي يستدعي الإجراء سيكون في حالة انتظار حتي يتم تنفيذ الإجراء البعيد وتصل النتائج.

الإجراء المتزامن أحياناً يكون ضرورياً ومهماً، مثلاً إذا كان الإجراء هو معرفة رصيد شخص ما، لنقرر كم نسحب من المبالغ ، يصبح هنا الانتظار ضروري، فلابد من النداء المتزامن. فليس هنالك أهم من هذه المعلومة ليكمل البرنامج عمله.

أما إذا كان الإجراء هو تحويل مبلغ من حساب شخص لحساب شخص آخر وليس مرتبطاً بعملية بعده، فيمكن جعل البرنامج الذي قام بالنداء تنفيذ أوامر أخرى لحين اكتمال تنفيذ الإجراء، في هذه الحالة يمكن إستدعاء الإجراء ثم مواصلة تنفيذ البرنامج وعندما يكتمل عمل الإجراء سيعطينا خبراً بذلك. أي يواصل البرنامج في عمله بينما يعمل الإجراء الذي تم استدعاؤه في نفس الوقت. هذه الخاصية الهامة تمكنا من تنفيذ العديد من البرامج في وقت واحد فيما يسمى المعالجة المتوازية(parallel processing)، ويسمى هذا النوع من النداء النداء غيرالمتزامن (asynchronous call).



شكل رقم (9-3): النداء غير المتزامن

أسئلة تقويم ذاتي

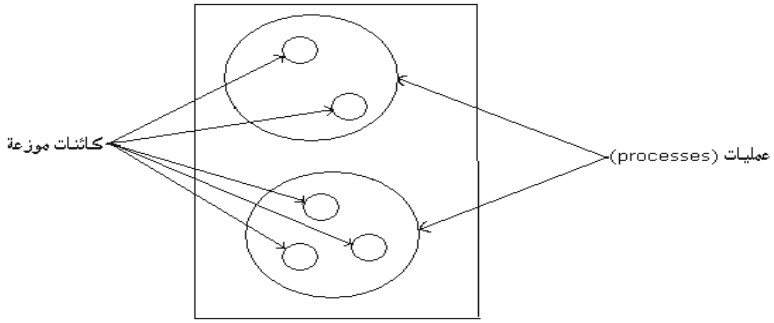


1. ما الفرق بين نداء الإجراء البعيد (RPC) ونداء الطريقة البعيدة (RMI) ؟
2. ما الغرض الرئيس من وجود الوكيل (stub) ؟

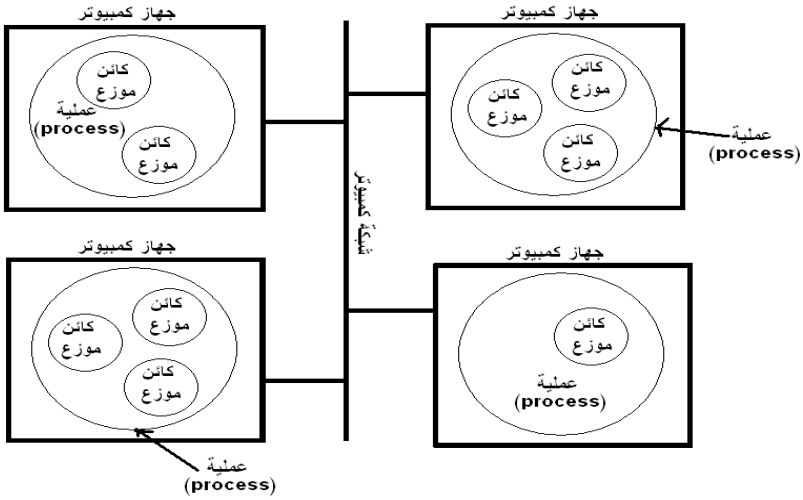
3. الكائنات الموزعة (distributed objects)

الكائنات الموزعة هي عبارة عن برامج مصممة في شكل كائنات وتعمل معاً، لكنها موزعة أما في حاسبات عدة متصلة بشبكة (شكل رقم 9–5) أو قد تكون في حاسب واحد (شكل رقم 9–4) لكنها في عمليات مختلفة (different processes). تتعامل الكائنات الموزعة مع بعضها بنفس طريقة نداء الإجراء البعيد (RPC)، فكل كائن يريد خدمة من كائن آخر ينادي دالة بذاك الكائن لتنفذ هناك بالجهاز البعيد ثم يأتي رد الخدمة عبر الشبكة.

نداء الإجراء البعيد (Remote Procedure Call (RPC)) عندما يستخدم لكائنات يسمى نداء الطريقة البعيدة (Remote Method Invocation (RMI)). هذه التسمية جاءت من مفاهيم اللغة كائنية التوجه، حيث نسمي الإجراء الموجود داخل الكائن طريقة (Method).



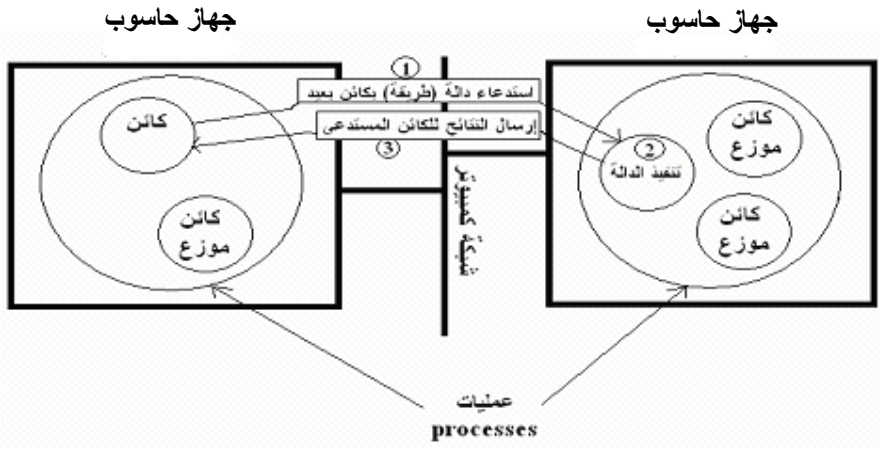
شكل رقم (9–4): كائنات موزعة في جهاز واحد



شكل رقم (9–5): كائنات موزعة عدة أجهزة

1.3 عندما يريد كائن "أ" مهمة من كائن آخر "ب":

- يرسل الكائن "أ" طلب (رسالة) للكائن "ب".
- ينفذ الكائن "ب" المهمة المطلوبة.
- يرسل الكائن "ب" الرد (النتائج) إلى الكائن "أ".



شكل (9-6): كائن في (برنامج) يستدعي طريقة (دالة) في كائن آخر بعملية (برنامج) بعيد

2.3 الكائن الموزع يختلف عن الكائن العادي في الآتي:

- طريقة إنشاء وحذف الكائن.
- إنشاء مرجع للوصول إلى الكائن عملية معقدة كثيراً في الكائنات الموزعة.
- تنفيذ طرق (دوال) في كائن موزع يعتبر أبطأ من تنفيذها بالكائن المحلي.
- الكائن الموزع قد لا يكون موجوداً طول الوقت لتلبية الطلبات.
- يمكن تنفيذ الكائنات الموزعة بالتوازي (عدد من الكائنات ينفذ في وقت واحد).
- هنالك طرق عديدة للاتصال مع الكائن الموزع.
- هنالك نقاط عديدة بالكائن الموزع يمكن أن يحدث فيها خلل.
- الكائن الموزع عرضة للخروقات الأمنية (ذلك لأنه يعمل في شبكة).

3.3 أمثلة للكائنات الموزعة

عزيزي الدارس، هنالك العديد من الطبقات الوسيطة (middleware) التي تطبق مفهوم الكائنات الموزعة مثل:

- RMI في لغة جافا.
- CORBA في العديد من لغات البرمجة مثل لغة ديلفي (Delphi).
- DCOM في بيئات ميكروسوفت
- .NET Remoting في بيئة دوت نت.
- Web services في بيئة دوت نيت و في جافا.

لماذا نحتاج الكائنات الموزعة

عزيزي الدارس، الهدف من أغلب النظم الموزعة هو جعل الكائنات متوفرة في أي مكان في الشبكة والسماح للتطبيقات بالتفاعل معها وكأنها موجودة محلياً . بعض النظم الموزعة تمكننا من إنشاء كائن في جهاز وإرساله إلى جهاز آخر بعيد و التحكم فيه من بعد واستخدامه كما لو كان في جهازنا.خاصية توزيع الكائنات في النُظم توفر علينا الكثير من الجهد ومتاعب التصميم وتجعل من السهل بناء نظم موزعة كبيرة.

أسئلة تقويم ذاتي



1. هل يستدعي البرنامج المحلي حقيقة الإجراء البعيد ؟ علل.
2. اذكر خمسة أمثلة لطبقات تطبق مفهوم الكائنات الموزعة.
3. ما الفروقات بين الكائن العادي والكائن الموزع ؟

الخلاصة

عزيزي الدارس، عرفنا في هذه الوحدة أنَّ فكرة الكائنات الموزعة مبنية على نداء الإجراء البعيد (RPC)،والتي هي طريقة تستخدم لطلب خدمة من برنامج في حاسوب بعيد على الشبكة دون الإلمام بفهم تفاصيل طريقة عمل الشبكة وأين يوجد البرنامج واسم الجهاز الذي يوجد به البرنامج والبروتوكول المستخدم في الاتصال ، وغيرها من تعقيدات الشبكات ومشاكل الاتصالات. ويستعمل نداء الإجراء البعيد في تقنية الخادم/العميل، فهنالك برنامج بجهاز يطلب تنفيذ إجراء بجهاز آخر، حيث يعتبر البرنامج الذي يطلب الخدمة عميلاً ، بينما البرنامج الذي ينفذ الخدمة خادم. وأيضا وضحنا أن النداء البعيد غير المتزامن يتم عند استدعاء الإجراء البعيد المتزامن، مما يعني أن البرنامج الذي يستدعي الإجراء سيكون في حالة انتظار حتى يتم تنفيذ الإجراء البعيد وتصل النتائج.

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس، سنتناول في الوحدة القادمة بإذن الله مفهوم وخصائص الطبقة الوسيطة RMI ومسئوليتها عن معالجة المسائل المتعلقة بالاتصال بين مكونات برمجية موزعة في أجهزة مختلفة. وسنتعرف أيضاً على كيفية إنشاء تطبيق موزع بسيط باستخدام RMI وكيف تبني تطبيق موزع ببرنامج eclipse، وسنتعرف على كيفية التعامل مع واجهات تطبيق java

مسرد المصطلحات

نداء الإجراء البعيد Remote Procedure Call (RPC)

هي طريقة تستخدم لطلب خدمة من برنامج في حاسوب بعيد على الشبكة دون الإلمام بفهم تفاصيل طريقة عمل الشبكة وأين يوجد البرنامج واسم الجهاز الذي يوجد به البرنامج والبروتوكول المستخدم في الاتصال ، وغيرها من تعقيدات الشبكات ومشاكل الاتصالات.

النداء البعيد غير المتزامن (synchronous)

عند استدعاء الإجراء البعيد المتزامن هذا يعني أن البرنامج الذي يستدعي الإجراء سيكون في حالة انتظار حتى يتم تنفيذ الإجراء البعيد وتصل النتائج. الإجراء المتزامن أحياناً يكون ضرورياً ومهماً

الكائنات الموزعة (distributed objects)

الكائنات الموزعة هي عبارة عن برامج مصممة في شكل كائنات وتعمل معاً، لكنها موزعة، أما في حاسبات عدة متصلة بشبكة (شكل رقم 9–5) أو قد تكون في حاسب واحد (شكل رقم 9–4) لكنها في عمليات مختلفة (different processes). تتعامل الكائنات الموزعة مع بعضها بنفس طريقة نداء الإجراء البعيد (RPC)، فكل كائن يريد خدمة من كائن آخر ينادي دالة بذاك الكائن لتنفذ هناك بالجهاز البعيد ثم يأتي رد الخدمة عبر الشبكة.

الكائنات الموزعة (distributed objects)

الكائنات الموزعة هي عبارة عن برامج مصممة في شكل كائنات وتعمل معاً، لكنها موزعة أما في حاسبات عدة متصلة بشبكة أو قد تكون في حاسب واحد لكنها في عمليات مختلفة (different processes) تتعامل الكائنات الموزعة مع بعضها بنفس طريقة نداء الإجراء البعيد (RPC)، فكل كائن يريد خدمة من كائن آخر ينادي دالة بذاك الكائن لتنفذ هناك بالجهاز البعيد ثم يأتي رد الخدمة عبر الشبكة.

المراجع

1- 1.Coulouris, G. et al., (2004), Distributed Systems, Concepts and Design, Addison Wesley
2- Farley, J. (1998), Java Distributed Computing, O'reilly & Associates
3- Fedorov, A, (2002), A Programmer Guide to .NET, Addison Wesley
4- Ince, D. (2002), Developing Distributed and E-Commerce Applications, Pearson Education
5- Liu, M. (2004), Distributed Computing Principles and Applications, Pearson Education
6- Schonauer, W. (1998), Scientific Supercomputing: Architecture and use of Shared and Distributed Memory Parallel Computers, Willi Schonauer, Germany
7- Tanenbaum, A., (2002), The Amoeba Distributed Operating System, Vrije Universiteit Technical Report, URL ftp://ftp.cs.vu.nl/pub/amoeba/Intro.ps.Z
8- Tanenbaum , A. AND Van Steen, M. (2002), Distributed Systems Principles and Paradigms, Prentice Hall
9- Tanenbaum, A. (1995), Distributed Operating Systems, 1/e, Prentice Hall, USA
10- Osman, A., (may 2006), A Multi-level Web-based Parallel Processing System, TRANSACTIONS ON ENGINEERING, COMPUTING AND TECHNOLOGY VOLUME 13 MAY 2006 ISSN 1305-5313
11-Forouzan, A. (2007), Data Communications and Networking, Mc Graw Hill, Fourth Edition
12-Tanenbaum, A., (2001), Modern Operating Systems, Prentice Hall, Second Edition
13-Tanenbaum, A., (1999), Structred Computer Organization, Prentice Hall, Fourth Edition


الوحدة العاشرة
طبقة RMI الوسيطة
Java RMI Middleware

محتويات الوحدة

الموضوع	الصفحة
المقدمة	191
تمهيد	191
أهداف الوحدة	**192**
1. الطبقة الوسيطة	**193**
1.1 دعم جافا للكائنات الموزعة	193
2.1 نداء الطريقة البعيدة(RMI)	193
3.1 RMI مبنية على تقنية الخادم/العميل	194
4.1 مفاهيم متقدمة في RMI	195
5.1 تطبيقات RMI الموزعة	196
2. إنشاء تطبيق موزع بسيط باستخدام RMI	198
1.2 واجهة الكائن البعيد (Salam. java)	199
2.2 تصميم برنامج العميل (Client.java)	203
3.2 زيادة تنظير داخل (RMI)	207
4.2 كيف يتعرف الوكيل على الجهاز الذي يوجد به برنامج الخادم والمنفذ (port) ؟	209
3. التطبيق الموزع الثاني	210
4. التطبيق الموزع الثالث (بواسطة Eclipse)	214
الخلاصة	246
لمحة مسبقة عن الوحدة القادمة	247
إجابات التدريبات	248
مسرد المصطلحات	250
المراجع	151

المقدمة

تمهيد

عزيزي الدارس مرحبا بك في الوحدة العاشرة من هذا المقرر والتي تأتيك بعنوان "الطبقة الوسيطة".

في تعريفنا للنظام الموزع ذكرنا أنه عبارة عن مكونات برمجية (software components) تعمل في أجهزة مختلفة. ولتكون هذه المكونات نظاماً واحداً متحداً لابد من أن تتعامل مع بعضها. ولتعامل مع بعضها لابد من اتصال، ولتتصل مع بعضها لابد من بروتوكولات لتبادل البيانات والأوامر وغيرها.

الاتصال بين الأجهزة يحتاج خطوات كثيرة، مثلاً الاتصال المعتمد (reliable) يحتاج الخطوات التالية: إعداد الاتصال، موافقة الطرف الآخر على الاتصال، إرسال الطلب، رد من الجهاز الآخر على الطلب، إغلاق الاتصال هنا تتبادر إلى أذهاننا الكثير من الأسئلة، مثل:كيف يجد جهازٌ جهازاً آخر بالشبكة؟ وكيف يصل إلى ما يريد داخل هذا الجهاز؟ ماذا يحصل لو تغير الجهاز الذي نتعامل معه من مكانه بالشبكة ؟كيف يتعامل جهاز واحد مع أجهزة متعددة بالشبكة، والعكس (أجهزة متعددة مع جهاز واحد)؟ ما هي لغة الحوار والبروتوكولات المستخدمة بين الأجهزة ؟كيف تسلسل البيانات (serialize) عند الإرسال، وكيف تعاد لشكلها الأصلي بعد الإرسال ؟كيف يستدعي برنامج دالةً ببرنامج آخر بعيد (في جهاز آخر)؟ كل هذه الأسئلة أين الإجابة عليها ؟ في هذا الوحدة سنتعرف على ذلك.

أهداف الوحدة



عزيزي الدارس بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً علي أن:

1. تشرح مفهوم وخصائص الطبقة الوسيطة **RMI**.
2. تنشي تطبيقاً موزعاً بسيطاً باستخدام **RMI**.
3. تبني تطبيقاً موزعاً ببرنامج **eclipse**.
4. تتعامل مع واجهات تطبيق **java**.

1. الطبقة الوسيطة

عزيزي الدراس، يمكن اعتبارها برنامجاً يربط تطبيقات منفصلة مع بعضها البعض. فالطبقة الوسيطة في النظام الموزع مسؤولة عن معالجة المسائل المتعلقة بالاتصال بين مكونات برمجية موزعة في أجهزة مختلفة.ومن مهامها أيضاً توفير طريقة لتسجيل الخدمات الجديدة واكتشاف الخدمات المتاحة للعملاء.

تمثل RMIطبقةً وسيطةً توفرها لغة جافا، حيث تسمح لكائن يعمل في آلة جافا افتراضية (JVM) من التعامل والاتصال بكائن آخر يعمل في آلة جافا افتراضية أخرى (JVM).فهي توفر اتصالاً بعيداً بين برامج مكتوبة بلغة جافا. الهدف الرئيسي من RMI توفير بيئة سهلة للمبرمجين لتطوير برامج موزعة بنفس الطريقة البسيطة التي يطورون بها البرامج العادية (غير الموزعة). فلا يحتاج المبرمج إلى التعامل مع الشبكات والبروتوكولات وغيرها من العقبات أعلاه. فهو يستطيع بناء تطبيق موزع كامل يعمل على شبكة محلية أو واسعة دون أن يكون لديه خبرة في برمجة الشبكات، ولا كيف تعمل الشبكة.

1.1 دعم جافا للكائنات الموزعة

تدعم لغة جافا مفهوم الكائنات الموزعة بصورة واسعة. ومن الأدوات المستخدمة في هذه اللغة لإنشاء الكائنات الموزعة والتعامل معها طبقة RMI الوسيطة (RMI middleware). سندرس هنا كيفية الاستفادة من هذه الطبقة في بناء كائنات موزعة في الشبكة وكيفية التعامل معها واستدعائها من بعيد. ثم سنقوم ببناء تطبيق موزع باستخدام هذه التقنية.

2.1 نداء الطريقة البعيدة(RMI)

طبقة جافا الوسيطة RMI هي طبقة تمكننا من استدعاء طرق (دوال) بعيدة موجودة بكائنات منتشرة في الشبكة.

1.2.1 حيث يمكن استخدامها للآتي:

- تمكين التطبيقات من نداء طرق بكائن بعيد.
- إتاحة واجهات الكائن البعيد (stub) للعميل (يعرف أيضاً المرجع البعيد (Remote reference)).
- تسجيل الكائنات البعيدة في مسجل (registry) يستخدمه العميل في العثور على هذه الكائنات عبر الشبكة.
- الاتصال بين الكائنات البعيدة وبين العملاء (المستفيدين منها).
- إنشاء كائن بعيد ديناميكاً عند الحاجة إليه (activation).

آلة جافا الافتراطية (JVM)

1. لايمكن تنفيذ أي كائن مكتوب بلغة جافا (java) ما لم يتم ترجمته.
2. عند ترجمة الكائن تنتج منه نسخة تُسمَّى JavaByteCode هذه النسخة هي برنامج بلغة آلة جافا الافتراضية جاهز للتنفيذ (.class).
3. لتنفيذ برنامج (.class) نحتاج آلة جافا الافتراضية (Java Virtual Machine (JVM))، لمزيد من المعلومات أنظر مرجع جافا المختصر بملحق الكتاب

إذا قمنا بتوزيع كائنات بعيدة في أجهزة متفرقة فلابد من أن يكون في كل جهاز من هذه الأجهزة آلة جافا الافتراطية (JVM). وإذا استدعينا دالة بكائن بعيد فستفذ بالجهاز البعيد وبالتالي على آلة جافا الافتراطية بذاك الجهاز.

3.1 RMI مبنية على تقنية الخادم/العميل

تعمل RMI بطريقة الخادم/العميل حيث يكون هنالك برنامج يمثل الخادم وبرنامج آخر يمثل العميل وتفاصيلهما كالآتي:

1.3.1 خادم RMI

- هو برنامج مكتوب بلغة جافا يقوم بإنشاء الكائنات التي نريد استدعاءها أو استدعاء دوال (طرق) منها.
- يقوم هذا البرنامج بربط (bind) أو تسجيل تلك الكائنات المنشأة بمسجل RMI Registry، حيث تصبح بهذا معروفة للآخرين.

2.3.1 عميل RMI

- برنامج يستفيد من الكائنات البعيدة الموجودة في الخادم.
- يطلب العميل من الخادم السابق البحث عن كائن ما بالمسجل (Registry).
- يبحث الخادم عن الكائن المطلوب.
- عند ما يجد الخادم الكائن المطلوب يعيد وكيل (Stub) هذا الكائن إلى العميل.
- يستدعي العميل الطرق (الدوال) المطلوبة.
- تنفذ الطريقة (الدالة) المستدعاة عند الخادم و ليس عند العميل.
- ترجع النتيجة (return) للعميل.

- استدعاء RMI يعتبر استدعاء لامتزامن (asynchronous).
- أحيانا نطلق على وكيل الكائن البعيد (stub)، المرجع البعيد (Remote reference).

4.1 مفاهيم متقدمة في RMI

باستخدام RMI يمكنني تصدير كائن لجهاز بعيد والتعامل معه كأنه كائن محلي .

تقوم RMI بكل التفاصيل الدقيقة التي نحتاجها للتعامل مع الشبكة من اتصالات وإرسال البيانات.

الكائنات الموزعة والمستخدمة عن طريق RMI لابد من أن تكون كائنات جافا، ذلك لأن RMI تقنية معتمدة كلية على الجافا.

هنالك بعض الطبقات الوسيطة غير معتمدة على لغة معينة مثل CORBA، هذا يعني أنه يمكن أن يكون لديك كائنات موزعة بأي لغة لديها رابط مع هذه التقنية، مثل C، C++، Java، Smalltalk، و Ada .

بما أن RMI معتمدة كلياً على الجافا فمسألة تكامل كائناتها الموزعة مع تطبيقات جافا يعتبر عادياً وسهلاً . وبما أن RMI مبنية على فرضية أن كائنات الخادم وكائنات العميل جميعها كائنات جافا فهذا يجعل من السهل تمديد طرق جامع النفايات المحلية (garbage collection) لآلة جافا الافتراطية (JVM) لتكوين جامع نفايات موزع يستطيع التخلص من الكائنات البعيدة والتي يتم تصديرها لأجهزة بعيدة (عند الانتهاء من استخدامها).

5.1 تطبيقات RMI الموزعة

يتكون عادة تطبيق جافا الموزع من شقين :

1.5.1 شق الخادم حيث يقوم بالآتي:

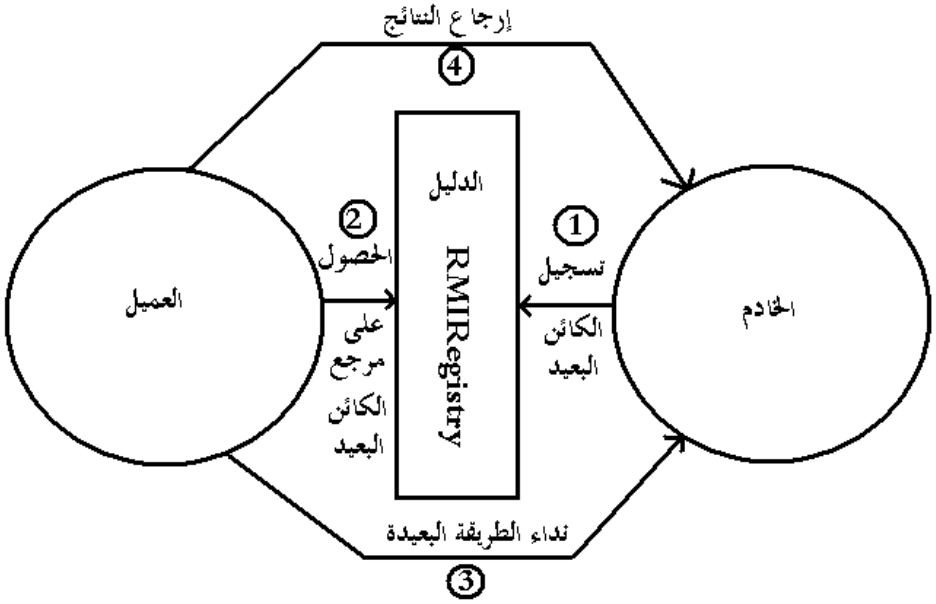
- إنشاء كائنات تسمى الكائنات البعيدة.
- يجعلها متاحة للوصول إليها بتوفير مراجع بعيدة (remote references) لها (يسمى أحيانا الوكيل (proxy or stub)).
- ينتظر العملاء لإرسال نداءاتهم للطرق الموجودة بهذه الكائنات.

2.5.1 شق العميل حيث يقوم بالآتي:

1. الحصول على مراجع بعيدة (وكلاء) للكائنات التي يريد التعامل معها.
2. يستدعي طرق الكائنات البعيدة.
3. يتم تنفيذ الطرق في جهاز الخادم وليس بجهاز العميل.

يوفر RMI طريقة تسمح للخادم والعميل بالاتصال وتبادل المعلومات. أيضا يوفر RMI طريقة بسيطة تسمى rmiregistry تشبه القاموس أو الدليل في عملها حيث:

1. يسجل الخادم كائناته البعيدة بها مع اسم تعريفي لكل كائن (مجرد اسم).
2. يمكن للعميل الحصول على مرجع الكائن (الوكيل) بالبحث عنه داخل هذا الدليل rmiregistry.



عزيزي الدارس سنتناول في الأقسام الآتية من هذه الوحدة الجزء العملي

دائماً في كل برامجنا التي نريدها أن تدعم الكائنات الموزعة نقوم باستيراد الحزمة java.rmi بواسطة الأمر import. ذلك لأنها تحتوي على كل الفئات التي تمكننا من بناء كائنات بعيدة والتعامل معها بواسطة العميل وكل ما نحتاجه لبناء تطبيق موزع يعمل على أجهزة متعددة عبر شبكة كمبيوتر، وأهم فئة بهذه الحزمة هي الفئة Remote والتي من خلالها (بوراثة صفاتها) نستطيع إنشاء كائن بعيد.

2. التطبيق الأول إنشاء تطبيق موزع بسيط باستخدام RMI

عزيزي الدارس، سنكتب هنا برنامجاً موزعاً بسيطاً جداً لتوضيح مفهوم الكائنات الموزعة وكيفية بناء تطبيقات موزعة على طبقة RMI.

في شق الخادم من التطبيق سنقوم بالآتي:

- إنشاء واجهة للكائن البعيد (salam.java).
- إنشاء كائن يطبق الواجهة أعلاه (Rserver.java).

في شق العميل من التطبيق نقوم بعمل برنامج يستدعي طرق الكائن البعيد في شق الخادم (Rclient.java).

إذن تطبيقنا الموزع الذي نحن بصدده يتكون من ثلاثة ملفات هي: Rclient،salam، Rserver. وتفاصيلها كالآتي:

- ملف salam.java: هذا ملف الواجهة interfaceويعتبر فئةً فارغةً أو هيكلاً مهمته تعريف العميل بالطرق التي بالكائن البعيد (واجهة للكائن البعيد). في مثالنا هذا توجد طريقة واحدة بهذه الفئة الواجهة هي ()SaySalam.

الواجهة في جافا (interfaces) هي فئات فارغة لا يوجد بها أي شفرات فقط تعريفات الطرق (الدوال) مثل اسم الطريقة ، القيمة التي ترجعها ، المعطيات (parameters) التي تمررها. ثم نستخدم فئة أخرى تطبق هذه الواجهة (implement)، بطريقة تشبه مفهوم الوراثة إلى حدٍ ما.

- ملف Rserver.java: يحتوي على الكائن البعيد (يطبق الواجهة salam.java)، حيث يحتوي على نفس الطريقة الموجودة بالواجهة (لكن هنا الطريقة بها شفرات (جسم)).
- ملف Rclient.java: البرنامج الذي يستدعي طريقة الكائن البعيد.

أدناه شفرات البرامج الثلاث مع شروحاتها.

1.2 واجهة الكائن البعيد (Salam.java)

البرنامج التالي يمثل واجهة الكائن البعيد حيث يحتوي على طريقة واحدة هي SaySalam() التي نريد استدعاءها من بعيد بواسطة برنامج العميل.

```
package example.salam;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Salam extends Remote {
   String saySalam() throws RemoteException;
}
```

أولاً نستورد الحزمة java.rmi.Remote والحزمة java.rmi.RemoteException ، حيث نستخدم الحزمة الأولى لإنشاء الكائن البعيد بوراثة صفاته التي توجد بالكائن Remote ، ويتم ذلك بالأمر extends Remote. أما الحزمة الثانية فهي للاستثناءات (throws RemoteException)، حيث لابد من أن نرمي استثناءات لكل الطرق البعيدة بالواجهة.

وبما أننا نريد إنشاء واجهة فلن نفعل سوى كتابة اسم الطريقة المراد وضعها بالكائن البعيد هي SaySalam() (تعريف الطريقة (رأسها)).

تعريفات الطرق بالواجهة لابد من أن تتطابق مع تعريفاتها بالكائن البعيد، فمثلا إذا عرفنا الطريقة بالواجهة على أن اسمها SaySalam وليس بها معطيات وترجع قيمة نصية لابد من أن تكون كذلك في الكائن البعيد. فإذا غيرنا ولو فقط في نوع القيمة الراجعة من نصية إلى رقمية تحدث مشكلة.

2.1.2 برنامج الخادم (Rserver.java)

في هذه المرحلة نصمم برنامج الخادم الذي يحتوي تطبيق الكائن البعيد (object implementation).

في الدالة الرئيسية main بهذا البرنامج نقوم بالآتي:

- إنشاء الكائن (البعيد) عن طريق تطبيق الواجهة salam.java.
- تصدير الكائن البعيد إلى java RMI runtime ليتلقى (يستلم) النداءات الواردة.
- تسجيل الكائن بالدليل RMI Registry ليصبح معروفاً للعميل.

```
package example.salam;

import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server implements Salam {

   public Server() {}

   public String saySalam() {
         return "Alslamu Alikum...";
   }

public static void main(String args[]) {

try {
   Server obj = new Server();
   Salam rst = (Salam) UnicastRemoteObject.exportObject(obj, 0);

            // Bind the remote object's stub in the registry
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("justName", rst);
```

```
            System.err.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

إنشاء الكائن البعيد يتم بالأمر:

Server obj = new Server();

حيث يمثل المتغير obj الكائن البعيد.

تصدير الكائن البعيد لتلقي الطلبات الواردة:

salam rst = (salam) UnicastRemoteObject.exportObject(obj, 0);

حيث يمثل المتغير rst وكيل الكائن البعيد (مرجع الكائن البعيد).

حيث تقوم الطريقة UnicastRemoteObject.exportObject بالآتي:

1. تصدير الكائن البعيد (obj) إلى Java RMI runtime ليستلم النداءات الواردة.
2. إنشاء وكيل للكائن البعيد (rst) يرسل للعملاء (لأنه هو الوحيد الذي يعرف كيف يمكن التعامل مع الكائن البعيد وبواسطته يصل العميل للكائن البعيد).

وكيل الكائن البعيد الذي تنشئه هذه الطريقة (الدالة) له نفس واجهات الكائن البعيد ويحتوي على اسم الجهاز البعيد (host name) والمنفذ الذي يستخدمه الكائن البعيد بالجهاز.

تنبيه

في جافا 5.0 توقف استخدام مترجم RMI الذي يسمى rmic والذي يستخدم لإنشاء وكيل (stub) للكائن البعيد. إذا كنت تستخدم نسخة جافا أقدم من النسخة 5.0 يمكنك الحصول على معلومات مفصلة عن كيفية استخدام rmic لإنشاء وكيل وتصديره) من موقع java.sun.com.

في نسخ جافا الجديدة (من 5.0 وأعلى)يتم إنشاء ذلك تلقائياً . في هذا الكتاب نستخدم نسخة جافا السادسة التعديل الثالث (jdk1.6.0_03)) وبالتالي لن نقوم باستخدام rmic.

تسجيل الكائن البعيد في مسجل RMI registry

ليستطيع العميل استدعاء طريقة بالكائن البعيد لابد أن يحصل على وكيل للكائن البعيد أولاً . لذلك يقوم العميل بالبحث عن وكيل الكائن البعيد في دليل RMIRegistry.

دليل RMI registry هو خدمة تسمح للعملاء بالعثور على مرجع (وكيل) للكائن البعيد. كل الكائنات البعيدة المسجلة بالدليل RMI registry، يستطيع العميل العثور عليها إذا استخدم الاسم التعريفي لها.

يستدعي العميل الطرق الموجودة بالوكيل، لكن الوكيل لاينفذ الاستدعاء وإنما يتصل بالكائن البعيد ليطلب منه تنفيذ الاستدعاء، ثم يتنظر ليستلم نتائج التنفيذ من الكائن البعيد، ثم يمرر النتائج للعميل، فيشعر العميل أن الذي قام بتنفيذ الاستدعاء هو الوكيل حيث استدعى طريقة به وجاءته النتيجة منه. ولكن الحقيقة غير ذلك كما بينا هذا.

الكود أدناه (بالخادم) يستخدم لتسجيل وكيل الكائن البعيد (rst) بالدليل (RMIregistry) مع الاسم "justName". حيث سيستخدم العميل فيما بعد هذا الاسم "justName" للبحث عن وكيل الكائن البعيد بالدليل.

```
Registry registry = LocateRegistry.getRegistry();
registry.bind("justName", rst);
```

2.2 تصميم برنامج العميل (Client.java)

عزيزي الدارس، مهام برنامج العميل هي:

1. البحث عن الكائن البعيد.
2. استدعاء الطرق الموجودة به.

```
package example.salam;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {

   private Client() {}

   public static void main(String[] args) {

         String host = (args.length < 1) ? null : args[0];
         try {
            Registry registry = LocateRegistry.getRegistry(host);
            Salam crst = (Salam) registry.lookup("justName");
            String response = crst.saySalam();
            System.out.println("response: " + response);
         } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
         }
   }
```

الأمر التالي:

```
String host = (args.length < 1) ? null : args[0];
```

سنختبر بهذا الأمر **هل تم تحديد اسم الجهاز** (أو عنوان IP) الذي يحتوي على برنامج الخادم في سطر الأوامر (command line) أم لا؟ (يختبر معطيات الدالة main). إذا تم ذلك فسنضع اسم الجهاز داخل المتغير host.

اذا لم نحدد اسم الجهاز سنستخدم null كاسم للجهاز (بالمتغير host) وهذا يعني أن عنوان الجهاز المحلي (أي أن برنامج العميل وبرنامج الخادم في جهاز واحد).

أما الأمر التالي:

```
Salam crst = (Salam) registry.lookup("justName");
```

يبحث العميل عن وكيل الكائن البعيد بالاسم (justName) في الدليل RMI registryثم يخزنه محلياً في المتغير crst.

الآن سيتعامل العميل مع crst وهو الوكيل كأنه الكائن البعيد. فيتم استدعاء الطريقة البعيدة من crst كالتالي:

```
String response = crst.saySalam();
```

ترجمة البرنامج

تتم ترجمة البرنامج بالصيغة التالية:

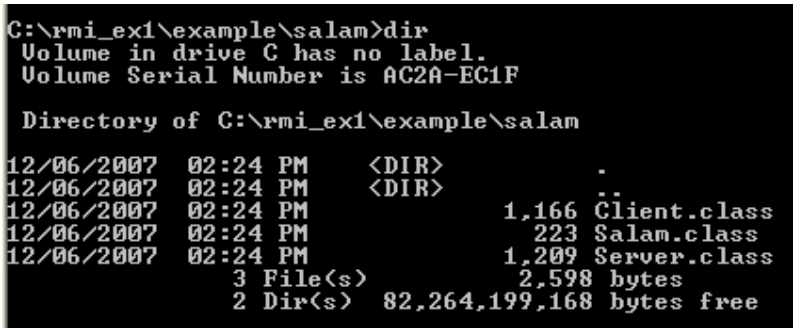
```
javac -d destDir Salam.java Server.java Client.java
```

حيث يمثل destDir الدليل الذي توجد به الملفات الثلاثة أعلاه.

بافتراض أن الملفات المصدرية (Salam.java, Rserver.java, Client.java) موجودة بالدليل c:\rmi_ex1 سأكتب الأمر كالتالي:

```
C:\rmi_ex1>javac -d c:\rmi_ex1 Salam.java Server.java Client.java
C:\rmi_ex1>
```

إذا نظرت للدليل c:\rmi_ex1 ستجد أن ملفات جافابايت كود (.class) قد تم إنشاؤها بالدليل c:\rmi_ex1\example\salam، كما في الشكل.



ذلك لأني استخدمت الأمر salam.example package في كل الملفات (يعني هذا وضعها في حزمة واحدة هي salam.example).

التنفيذ

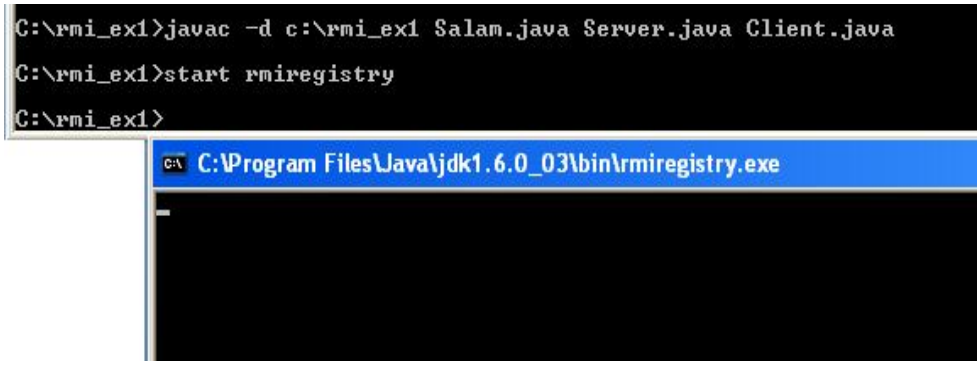
بعد ترجمة الملفات يمكنك تنفيذ التطبيق بالخطوات التالية:

- تشغيل RMIRegistry.
- تشغيل برنامج الخادم.
- تشغيل برنامج العميل.

وأوامر التشغيل كالآتي:

تشغيل برنامج المسجل RMI Registry بالأمر التالي.

```
start rmiregistry
```



نشغل برنامج الخادم server بالصيغة:

```
start java -classpath c:\rmi_ex1 -
Djava.rmi.server.codebase=file:classDir/ example.salam.Server
```

فتظهر الشاشة التالية.

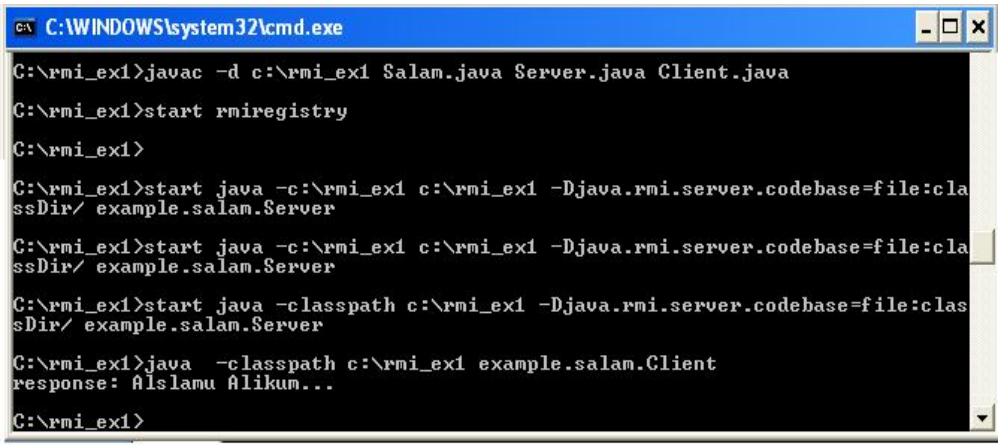


تشغيل العميل بالأمر:

```
java  -classpath classDir example.hello.Client
```

مثلاً classDir سيكون هنا c:\rmi_ex1

فتظهر الشاشة أدناه حيث يتم تنفيذ البرنامج ويتصل مع الخادم ويستدعي الدالة SaySalam فتظهر عبارة Aslamu Alikum في شاشة الخادم.



بهذا نكون قد أكملنا تطبيقنا الأول بنجاحٍ بحمد الله. فقد جعلنا العميل يطلب من الخادم أمراً وهو أن يقول السلام عليكم وقد فعل.

3.2 زيادة توضيح (داخل RMI)

إذا أردت أن تعرف تفاصيل أكثر عن RMI وطريقة عملها الداخلية فلك ذلك...

تعمل RMI في الأساس بطريقة الخادم/العميل ، فهنالك كائن بعيد في خادم (جهاز) يريد تنفيذ دواله (طرقه) برنامج قريب في عميل (جهاز آخر).

كيف يتم ذلك هو ما تقوم به RMI . فالعمل حقيقة هو تبادل رسائل عبر الشبكة وبرمجة الشبكة والتعامل مع الـ sockets، لذلك لابد من بناء بيئة تسمح للعميلألاً يهتم بتفاصيل الشبكة وبمكان الدوال، وإنما يتعامل معها كأنها محلية بالجهاز نفسه. لذلك أوجد مطوري RMI فكرة وكيل العميل (stub) ووكيل الخادم (skeleton).

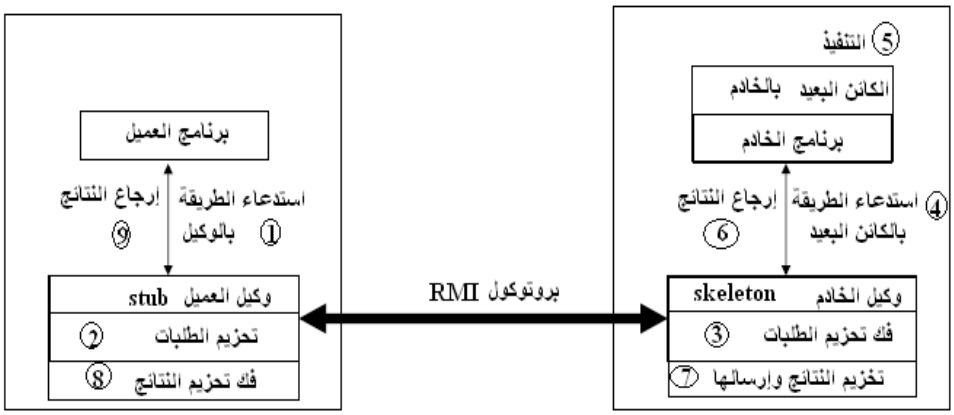
الفكرة هي وضع كل ما يتعلق ببرمجة الشبكة في الوكيل (stub و skeleton). فلا يحتاج الخادم ولا العميل إلى التعامل مع الشبكة والـ sockets داخل برامجهم.

يطبق وكيل العميل (stub) واجهة الكائن البعيد. بالتالي يستدعي العميل الطرق الموجودة بالوكيل (stub).

ولكن الوكيل (stub) يحتوي على أكوادٍ متعلقة بالشبكة مثل:

1. كيفية الاتصال بالجهاز البعيد.
2. كيفية إرسال تفاصيل الدوال المستدعاة.
3. كيفية إرسال معطيات الدوال.
4. كيفية استلام النتائج من الجهاز البعيد.

ولا يوجد به أكواد الدوال المستدعاة هذه.



Client<--->stub<--->[NETWORK]<--->skeleton<--->Server

1. لذلك سيرسل العميل الطلب للوكيل (stub).
2. يمرر الوكيل (stub) الطلب لوكيل الخادم (skeleton) عبر الشبكة.
3. يمرر الوكيل (skeleton) بدوره الطلب إلى الخادم.
4. ينفذ الخادم الطلب.
5. ترجع نتيجة التنفيذ بنفس الطريقة فمن الخادم إلى وكيله (skeleton) ومنه إلى وكيل العميل (stub) عبر الشبكة ثم إلى العميل.

الآن أصبح وكيل الخادم (skeleton) مضمناً في الخادم فلا حاجة إلى وكيل (skeleton) منفصل.

تفاصيل التفاصيل (إذا أردت المزيد)

على مستوى الـ socket يتم الاتصال كالآتي (مخفي عنا (يقوم به RMI)):

- يتصنت الخادم على منفذ معين بجهاز الخادم.
- لا يعلم العميل في اي جهاز وعلى اي منفذ يصطنتُ الخادم ولكن وكيله يعرف كل هذا.

- لذلك يتعامل العميل مع وكيله.
- يقوم وكيل العميل بالاتصال بالخادم عبر المنفذ الذي يصطنتُ فيه ويرسل له بيانات الدالة من معطيات وغيرها، وتفاصيل هذا الاتصال تتم كالآتي:

أ. يتصل العميل بمنفذ الخادم.

ب. يقبل الخادم الاتصال وينشئ socket جديدة لهذا الاتصال.

ت. يواصل الخادم الاصطنات (ليستقبل طلبات أخرى).

ث. يتبادل الخادم والعميل البيانات (المعطيات والنتائج) عبر الـ socket.

ج. البروتوكول المستخدم هنا قد يكون JRMP(Java Remote Method Protocol) أو RMI-IIOP (Internet Inter-ORB Protocol)

1. تنفذ الدالة في الخادم وترجع النتائج إلى الوكيل.
2. يرجع الوكيل النتائج للعميل.

4.2 كيف يتعرف الوكيل على الجهاز الذي يوجد به برنامج الخادم والمنفذ (port) ؟

تتعبر هذه مشكلة كبيرة (كما ذكرنا هذا في مفاهيم النظم الموزعة في خاصية الانفتاح). فكيف نُعرِّف العميل بمكان الخادم لكلي يتصل به ؟

هنالك حلول كثيرة، حل RMI لهذه المشكلة كان بواسطة الدليل RMI registry . حيث يتم تسجيل كل ((كائن بعيد) مع (اسم تعريفي له)) بهذا الدليل RMIRegistry في شكل أزواج (كأنه قاموس). فيستخدم العميل هذا القاموس للبحث عن الكائن البعيد.

3. التطبيق الموزع الثاني

عزيزي الدارس، الفرق بين التطبيق الأول وهذا التطبيق هو تغيير طريقة (دالة) الكائن البعيد لتجمع عددين بدلاً من إظهار رسالة على الشاشة. الغرض من هذا المثال توضيح الفروقات بين التطبيقين، حيث ستلاحظ فعلاً أنه ليس هنالك فروقات كبيرة بين أكواد هذا التطبيق والسابق.

1.3 الواجهة (Clac.java):

```
package ex2;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Calc extends Remote {
   int add(int x, int y) throws RemoteException;
}
```

2.3 برنامج الخادم (Server.java):

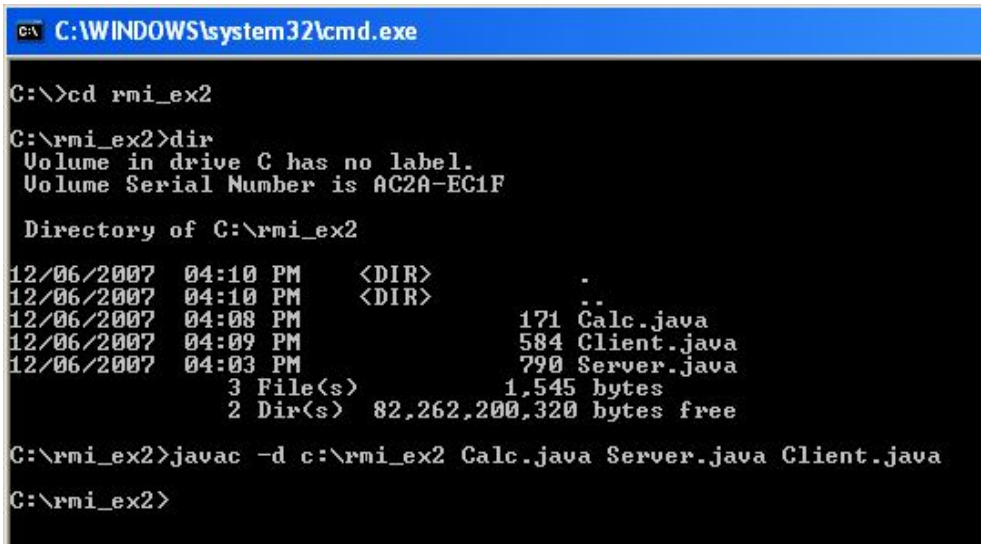
```
package ex2;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class Server implements Calc {
   public Server() {}
   public int add(int x, int y) {
        return x+y;
   }
   public static void main(String args[]) {
        try {
           Server obj = new Server();
```

```
            Calc stub = (Calc) UnicastRemoteObject.exportObject(obj,
0);
            // Bind the remote object's stub in the registry
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("aname", stub);
            System.err.println("Server waiting...");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

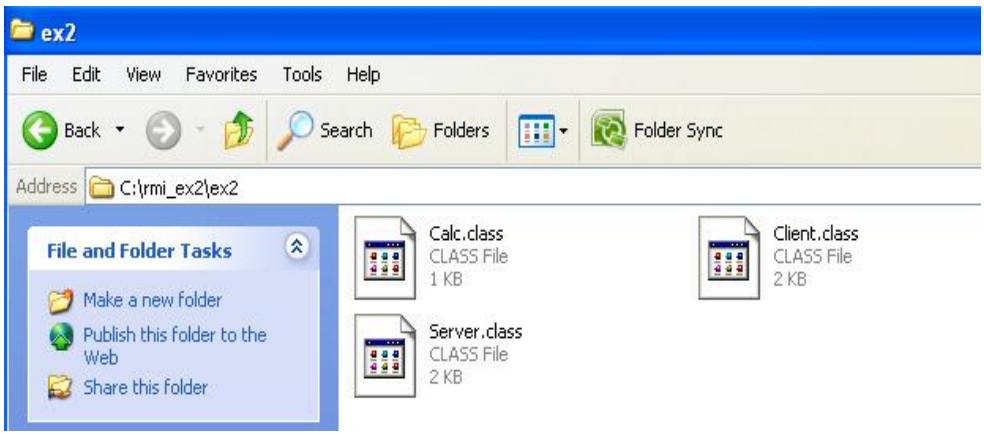
3.3 برنامج العميل (Client.java):

```
package ex2;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class Client {
    private Client() {}
    public static void main(String[] args) {
        String host = (args.length < 1) ? null : args[0];
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            Calc stub = (Calc) registry.lookup("aname");
            int response = stub.add(5,7);
            System.out.println("response: " + response);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

الترجمة



الملفات المترجمة بالدليل c:\rmi_ex2\ex2 ذلك لأن الأمر الآن هو package ex2; في ملفات التطبيق، حيث سينشأ دليل ex2 وتخزن فيه ملفات جافابايت كود.



التنفيذ

الأوامر التالية هي :

1. تشغيل RMIRegistry بالأمر start rmiregistry

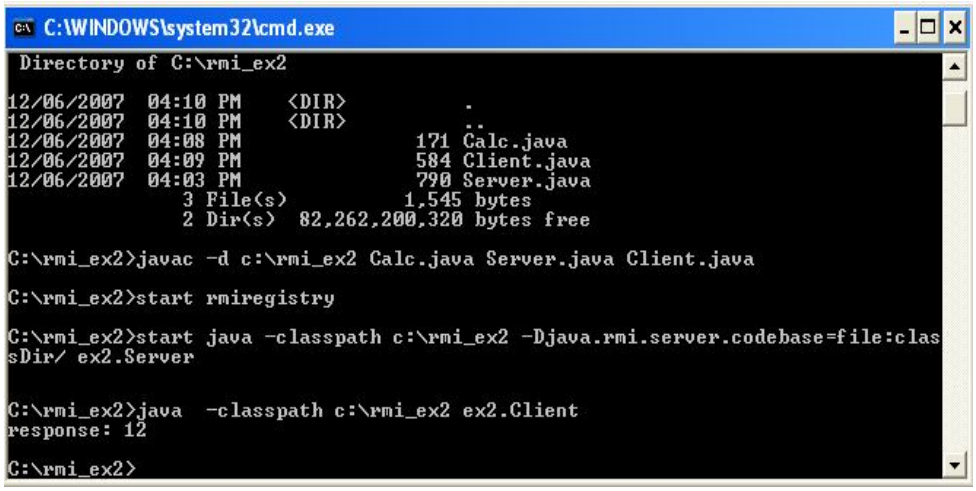
2. تشغيل برنامج الخادم (server) بالأمر :

```
start java -classpath c:\rmi_ex2 -
Djava.rmi.server.codebase=file:classDir/ ex2.Server
```

3. تشغيل برنامج العميل (client) بالأمر :

```
java  -classpath c:\rmi_ex2 ex2.Client
```

فينتج التالي:



أسئلة تقويم ذاتي

1. ما الفرق بين النداء المتزامن والنداء غير المتزامن؟
2. من الذي ينشئ الوكيل في الحقيقة؟ هل هو الخادم، الدليل Registry، أم العميل؟ ينشأ بواسطة الفئة Naming بالخادم.
3. ما المنفذ الذي يتصنت عليه الخادم؟ المنفذ الذي يستخدمه الخادم للتصنت يختار عشوائياً في زمن التنفيذ ويحدد بواسطة JVM أو نظام التشغيل.

4. التطبيق الموزع الثالث (بواسطة Eclipse)

لتحميل eclipse التعامل معه ارجع لملاحق الكتاب.

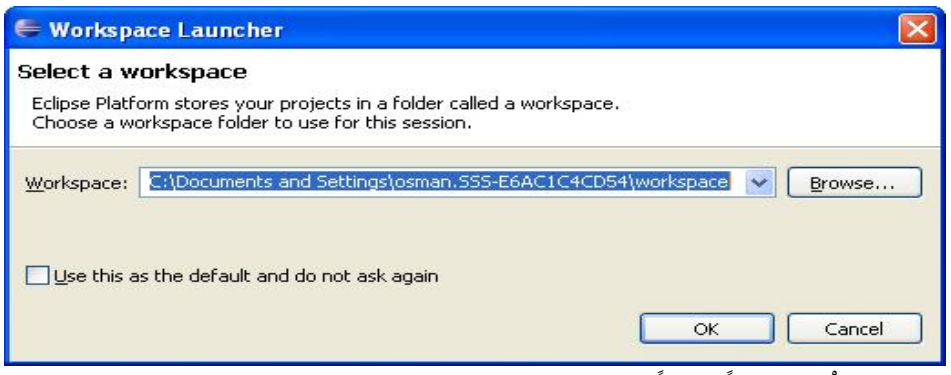
التطبيق الثالث يوضح كيفية بناء تطبيق موزع ببرنامج eclipse.

الخطوة الأولى هي أن نتأكد أن برنامج eclipse يدعم RMI ، إذا لم يكن يحتوي على طبقة RMI يمكنك متابعة خطوات تثبيت RMI على eclipse بملحق الكتاب.

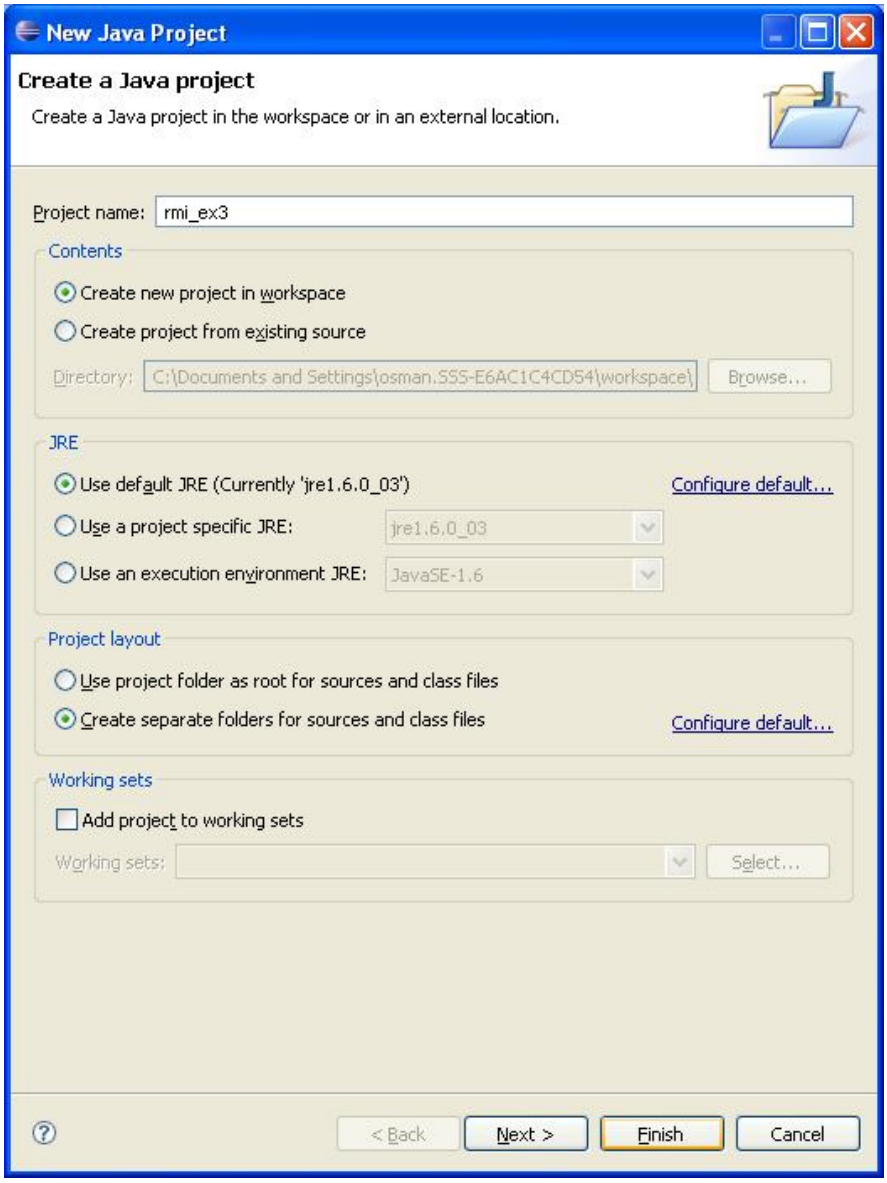
فالنبدأ ...

1. افتح eclipse 

2. اختار المكان الذي تريد تخزين مشروعك به (يمكنك النقر على موافق إذا أردت الدليل الافتراضي) وهذا ما سنفعله هنا.



3. أنشئ مشروعاً جديداً بالأمر: File – new – project.



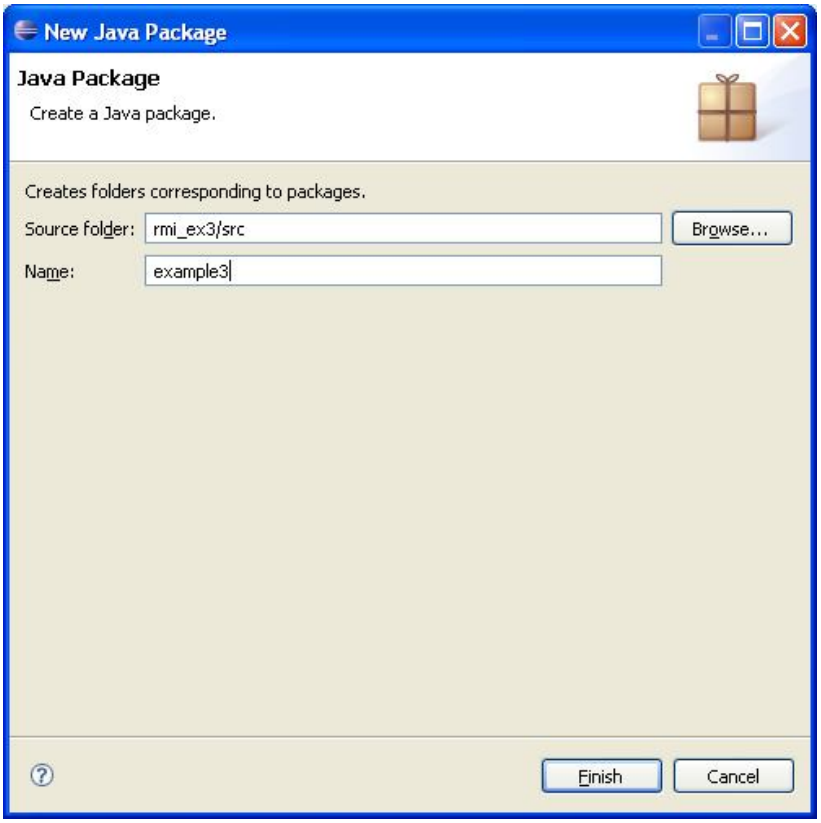
اكتب اسم المشروع rmi_ex3

ثم انقر Finish

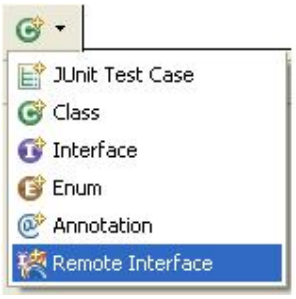
اختر file-new-package



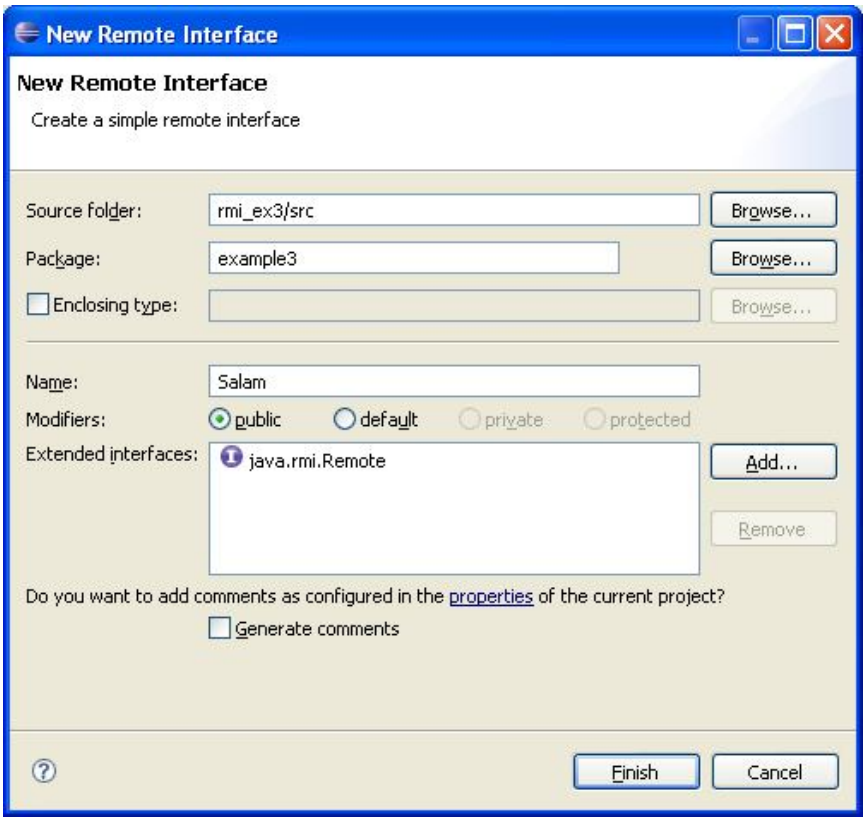
لإنشاء حزمة داخل المشروع ، ولنسميها example3



ثم ننشئ الواجهة :



نسمي الواجهة Salam



Salam.java

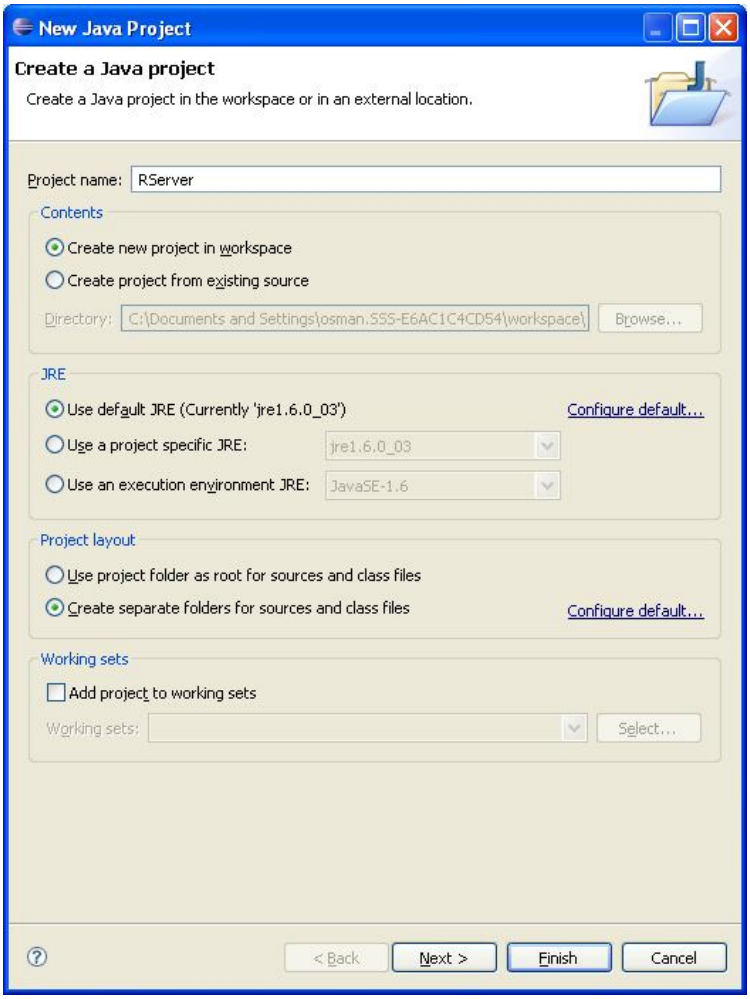
```
package example3;

import java.rmi.Remote;

public interface Salam extends Remote {

    /** This is a sample method. Delete it and add one of your own. */
    public String simpleRemoteMethod(String arg) throws RemoteException;

}
```

هذه الواجهة للكائن البعيد ، نعدل الواجهة بوضع الدالة SaySalam بها فتصبح الواجهة كما يلي:

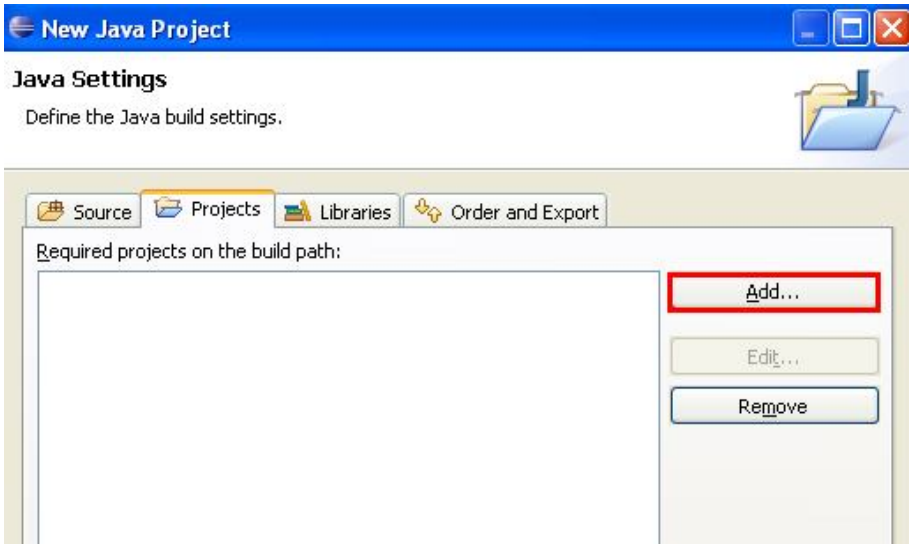
*Salam.java

```
package example3;

import java.rmi.Remote;

public interface Salam extends Remote {

    /** This is a sample method. Delete it and add one of your own. */
    public String SaySalam(String arg) throws RemoteException;

}
```

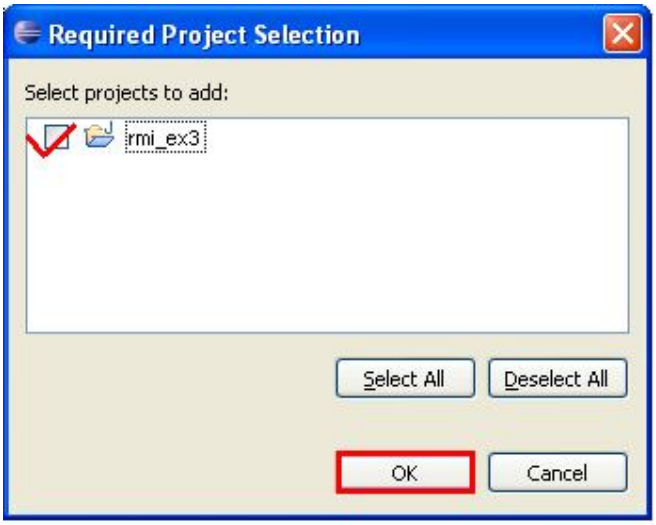
ننشئ مشروعاً آخر جديداً نسميه RServer :


New Java Project
Create a Java project
Create a Java project in the workspace or in an external location.
Project name: RServer
Contents
Create new project in workspace
Create project from existing source
Directory: C:\Documents and Settings\osman.SSS-E6AC1C4CD54\workspace\
Browse...
JRE
Use default JRE (Currently 'jre1.6.0_03')
Configure default...
Use a project specific JRE:
jre1.6.0_03
Use an execution environment JRE:
JavaSE-1.6
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
Working sets:
Select...
< Back
Next >
Finish
Cancel

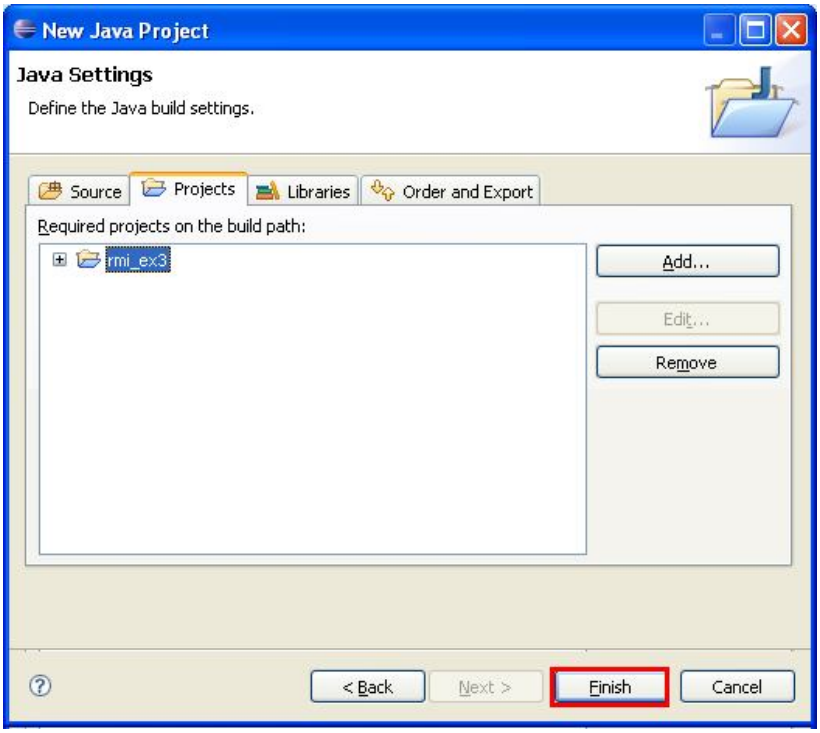
نختار next



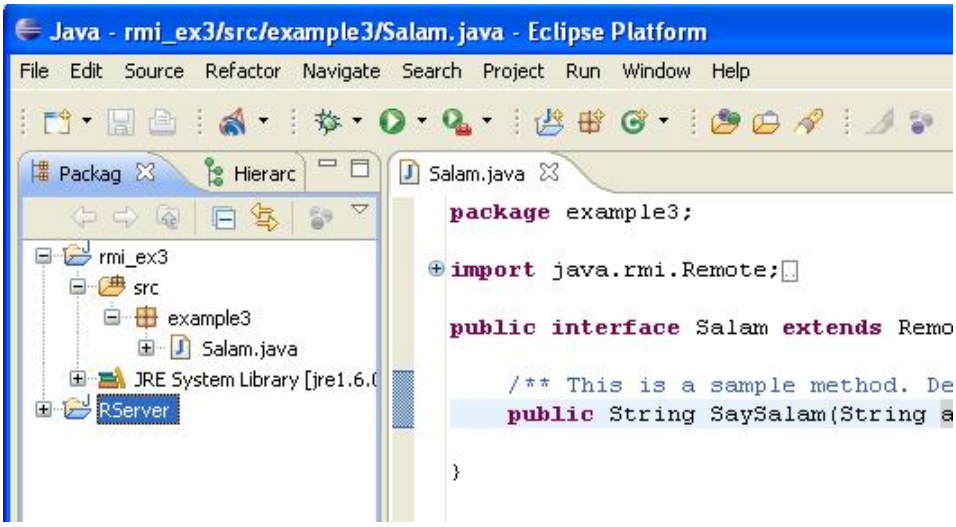
ننقر على add



نختار rmi_ex3 ثم ننقر OK

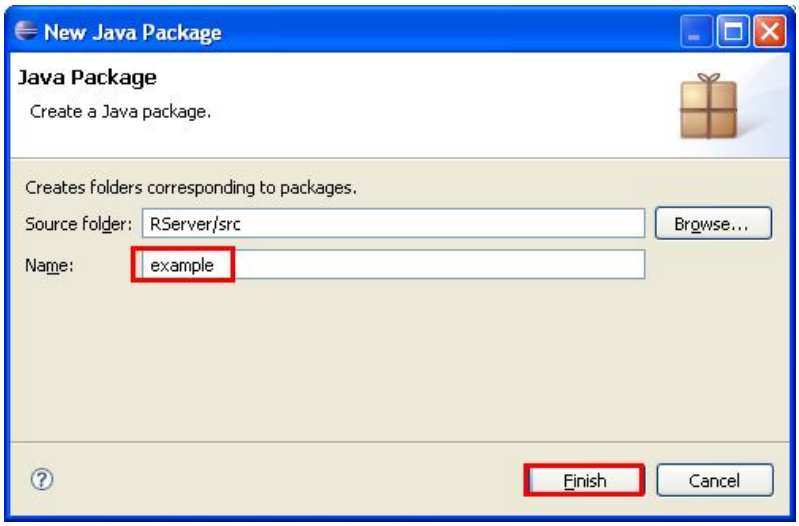


ثم finish

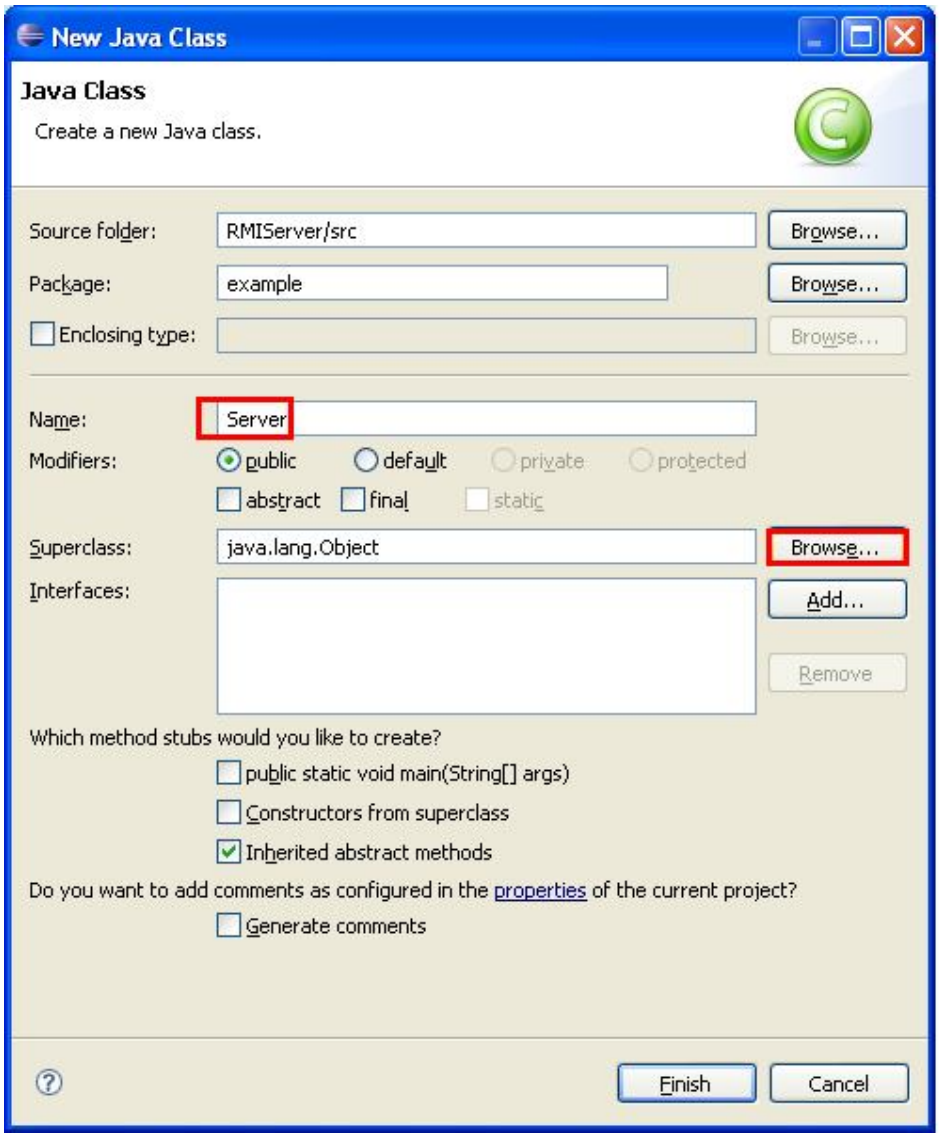


ننقر يمين الماوس على RServer ثم نختار new – package

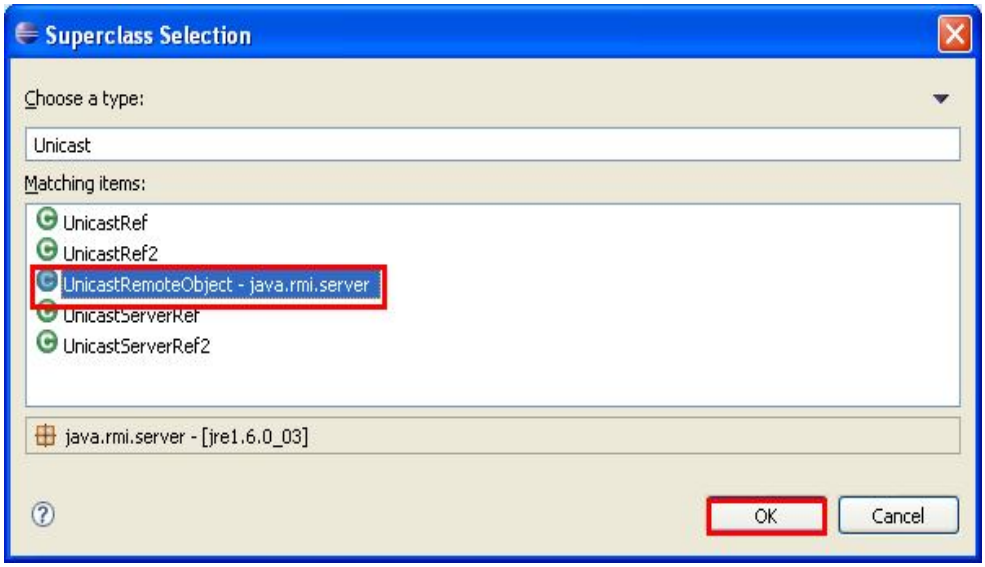
نسمي الحزمة example "نفس اسم الحزمة بالواجهة"



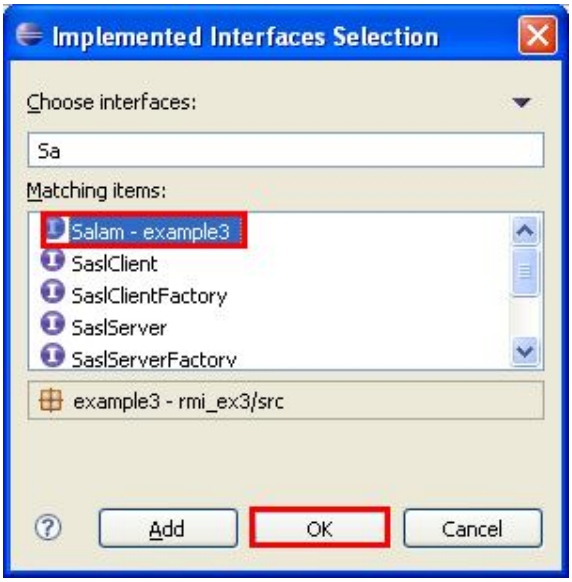
ثم ننشئ فئةً نسميها Server وبها دالة رئيسية main

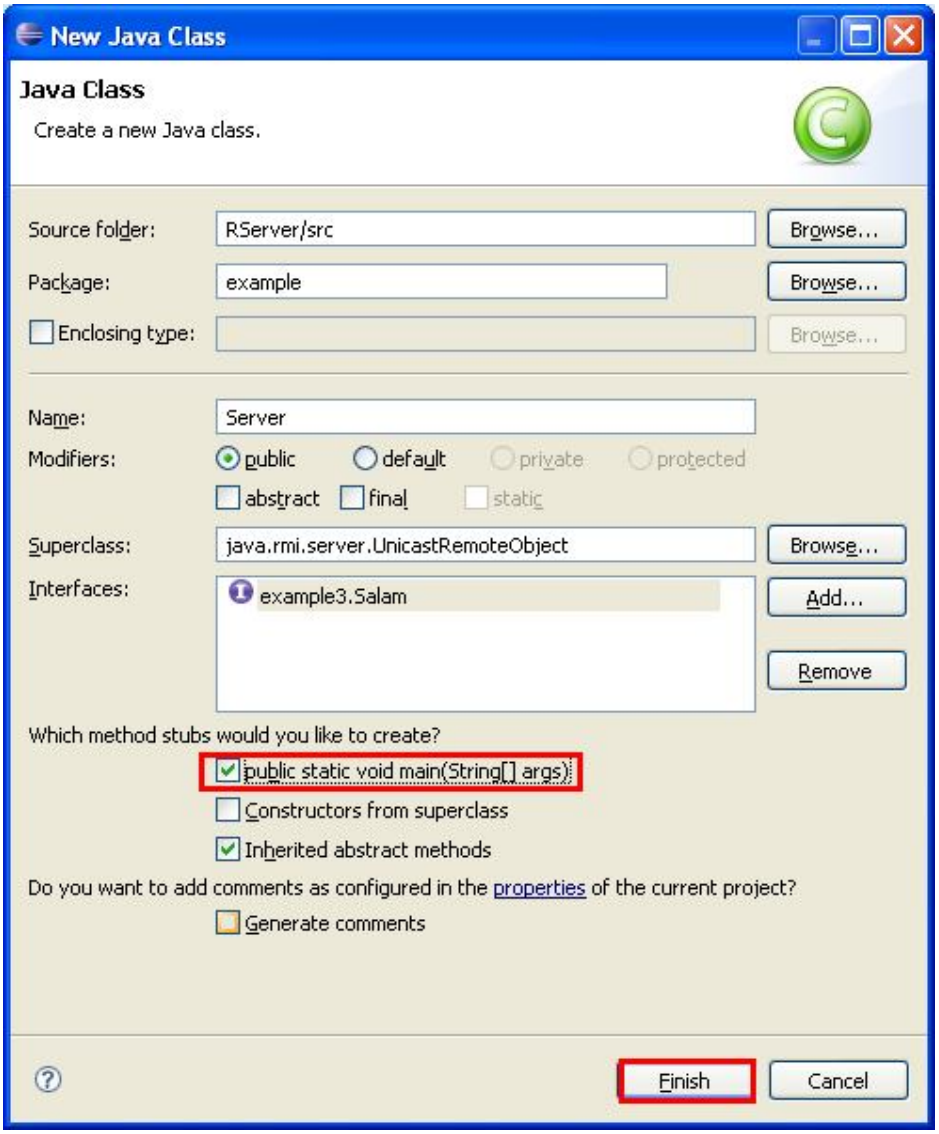


بالضغط على browse استطيع إضافة UnicastRemoteObject كما يلي:



ثم بالنقر على add لإضافة الواجهة Salam كما يلي:

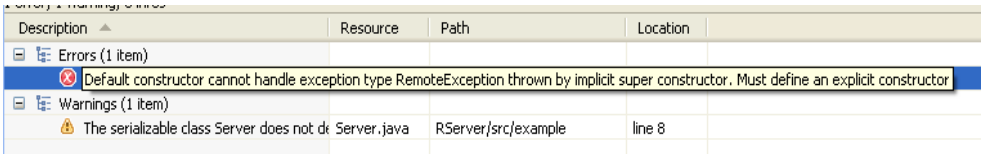




لابد من تعريف دالة مشيدة لهذه الفئة كما يلي

```
public class Server extends UnicastRemoteObject imple

    public Server(){
    }

    }
    @Override
```

لأننا أن لم نفعل سينتج هذا الخطأ:



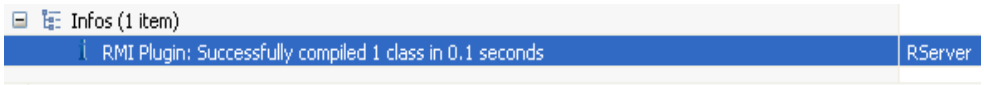
نكتب في الدالة main الاوامر التالية، والتي تنشئ Registry وتربط الكائن البعيد بالاسم "myname" ، لكي يستطيع العميل فيما بعد البحث عن الكائن البعيد بهذا الاسم .

```
try {
        Registry r = LocateRegistry.getRegistry();
        r.bind("myname", new Server());
} catch (Exception e){
e.printStackTrace();}
```

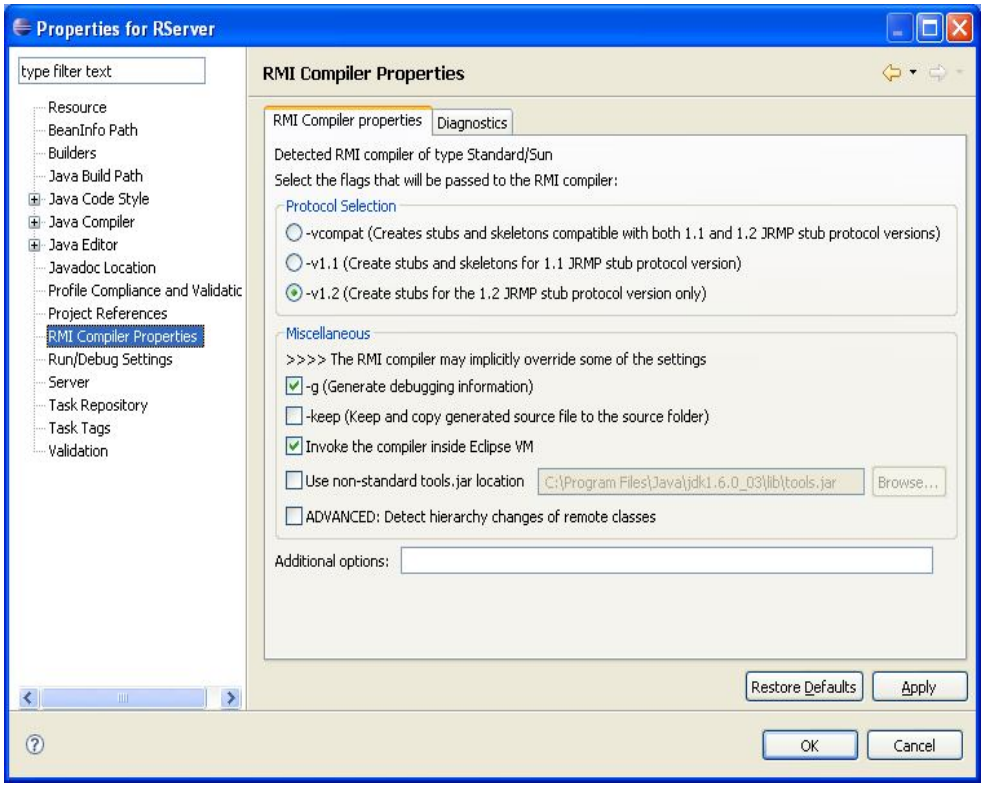
ننقر يمين الماوس على اسم المشروع (RServer) ثم نختار RMI ومنها enable stubs generator، وهذه الخطوة يتم فيها إنشاء وكيل الكائن البعيد.



فيظهر رسالة تخبرك بأن الوكيل قد تم إنشاؤه:



إذا أردت ان ترى الوكيل انقر يمين الماوس على اسم المشروع ثم اختر properties، فتظهر لك شاشة اختر منها RMI compiler Properties لتظهر الشاشة التالية:



انقر على keep (keep and copy generated source file to the source folder)

فيظهر الوكيل Server_stub.java في ملفات المشروع يمكنك بالنقر المزدوج عليه أن ترى أكواده:

Salam.java | *Server.java | Server_Stub.java

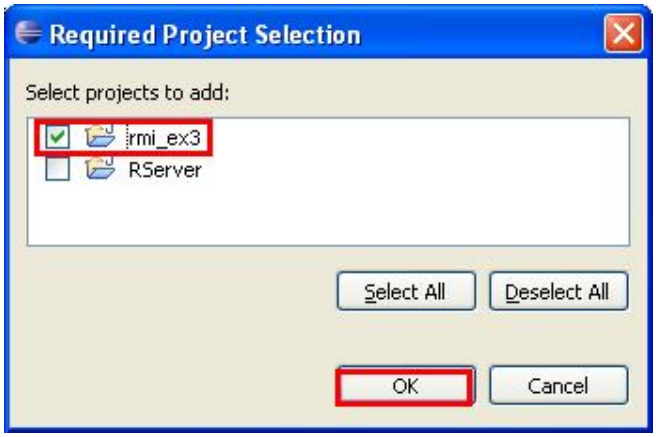
```
// Stub class generated by rmic, do not edit.

package example;

public final class Server_Stub
    extends java.rmi.server.RemoteStub
    implements example3.Salam, java.rmi.Remote
{
    private static final long serialVersionUID = 2;

    private static java.lang.reflect.Method $method_SaySalam_0;

    static {
    try {
        $method_SaySalam_0 = example3.Salam.class.getMethod("SaySalam",
    } catch (java.lang.NoSuchMethodException e) {
        throw new java.lang.NoSuchMethodError(
        "stub class initialization failed");
    }
    }

    // constructors
    public Server_Stub(java.rmi.server.RemoteRef ref) {
    super(ref);
    }
```

برنامج العميل

ننشئ مشروعاً جديداً من file-new-project ونسميه RClient ثم ننقر على next

ونختار project tab

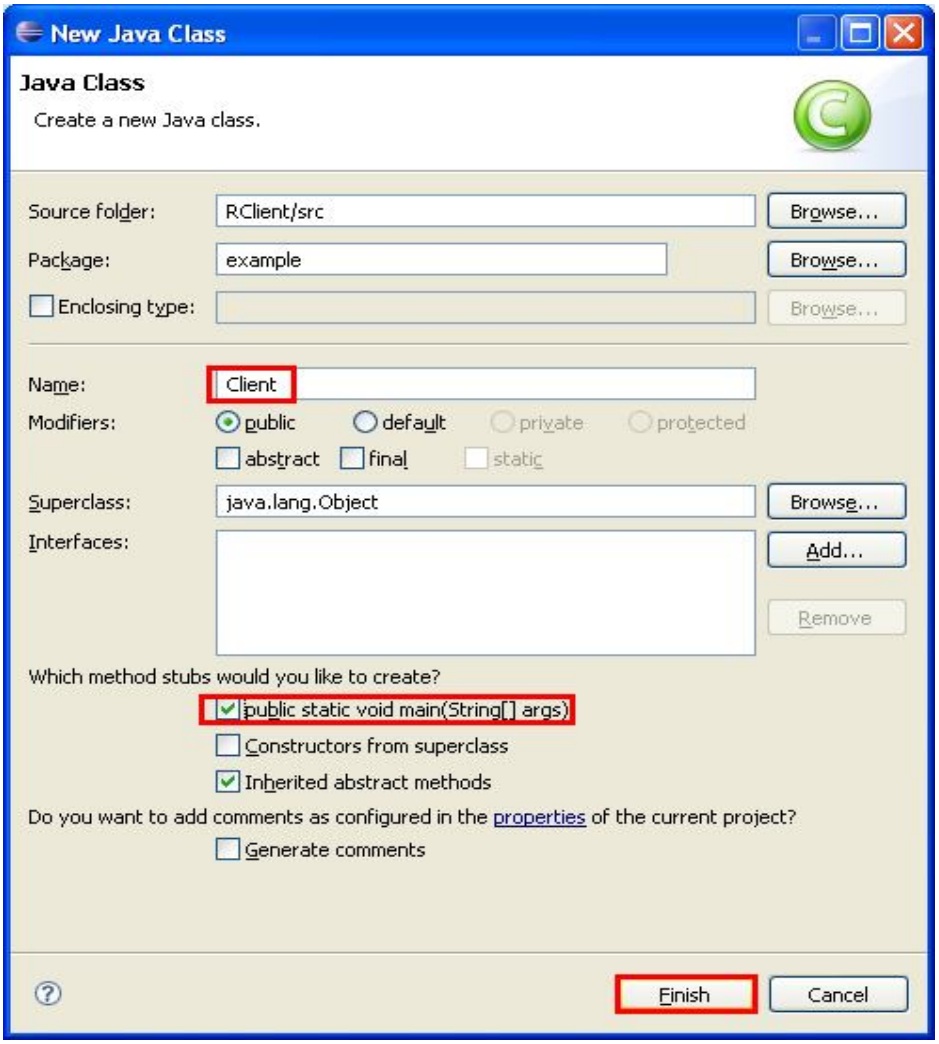
ثم نختار rmi_ex3 ثم ok



ثم finish

نضيف للمشروع حزمة جديدة نسميها example.

ثم نضيف فئة جديدة نسميها Client ونختار فيها دالة main كما يلي:



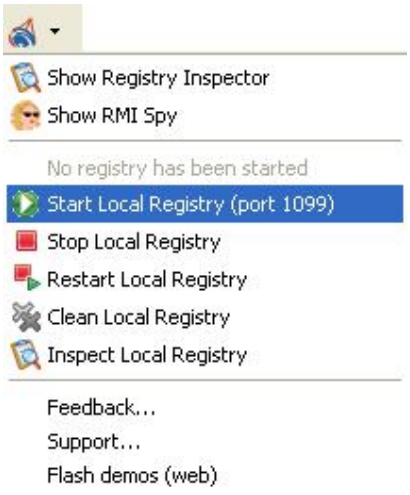
نكتب في الدالة main الآتي:

```
try{
System.setSecurityManager(new RMISecurityManager());
Salam sa = (Salam) Naming.lookup("rmi://localhost/example");
String s = sa.SaySalam("Alslam alikum");
System.out.print("client call remote method");
}catch (Exception e){
        e.printStackTrace();
}
```

وبهذا يكون برنامج العميل قد اكتمل والحمد لله.

كيف ننفذ برنامج الخادم والعميل في جهاز واحد

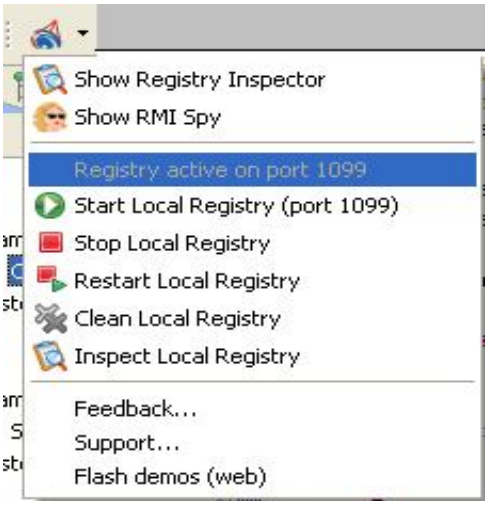
نبدأ بتشغيل RMI Registry بالنقر على RMI control Center Icon



تظهر الشاشة التالية:



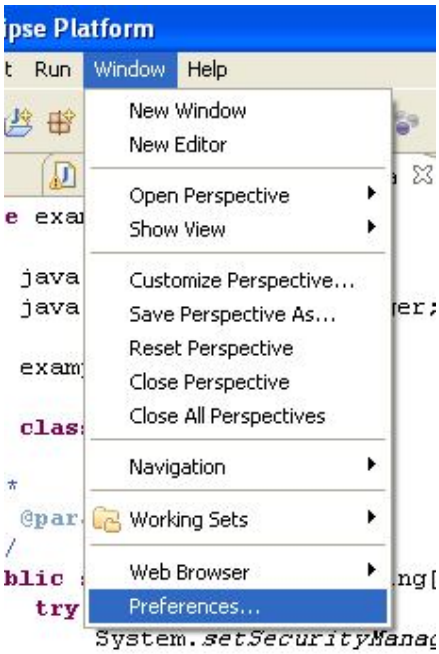
انقر No ثم مرة أخرى على RMI Control Center لترى أن



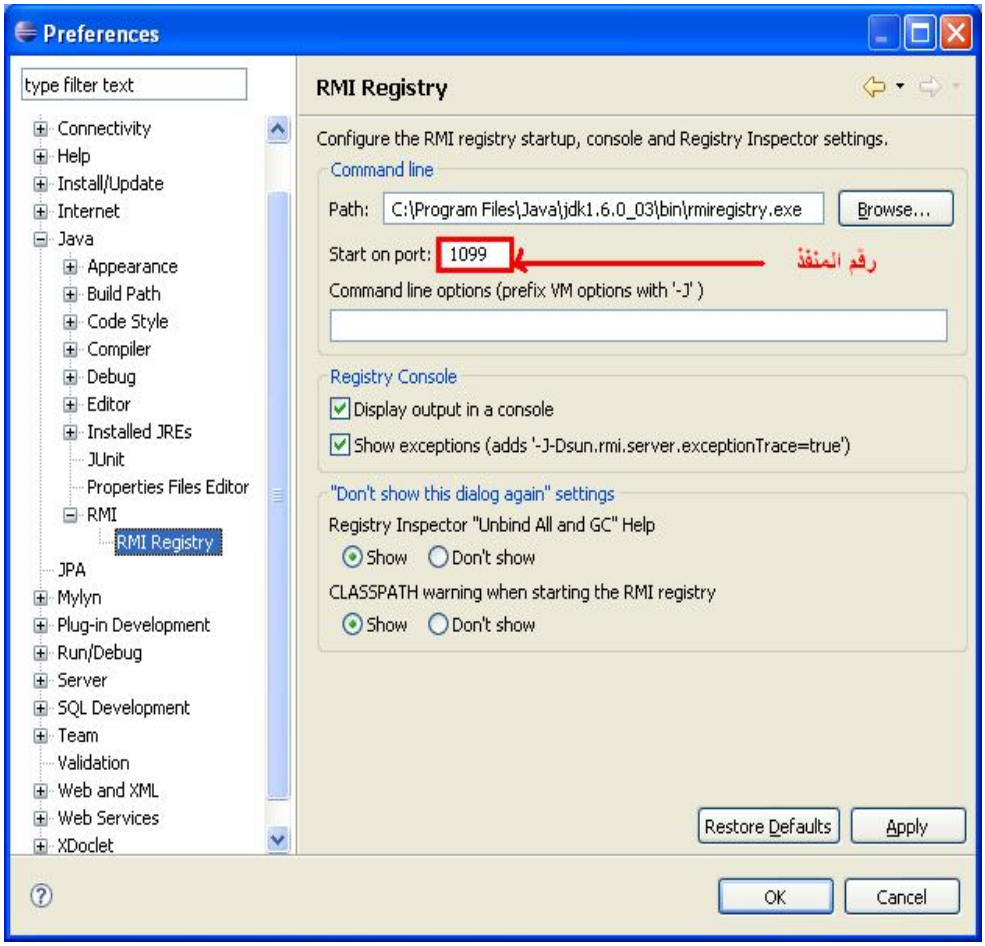
مما يدل على أن الـ Registry يعمل.

أيضاً يمكنك معرفة المنفذ الذي يعمل عليه :

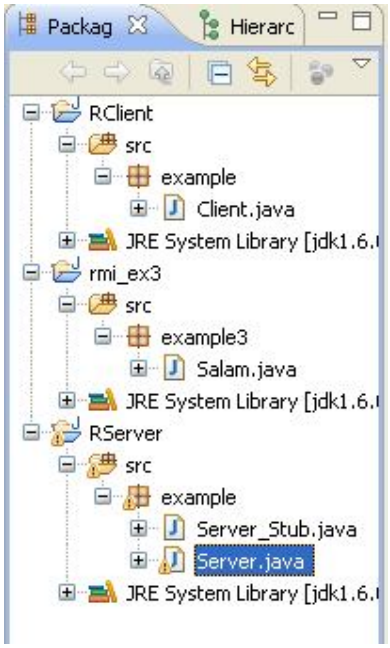
من Windows –references



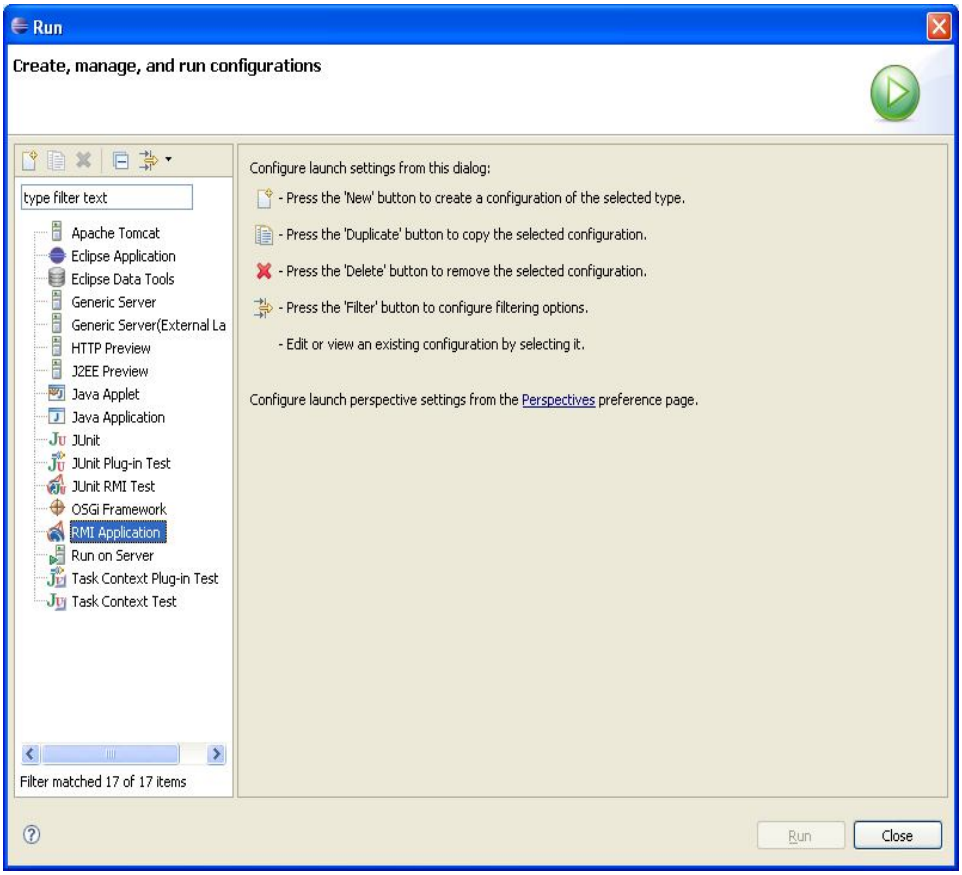
ثم باختيار java/RMI ثم local registry ورقم المنفذ للتأكد أنه يعمل.

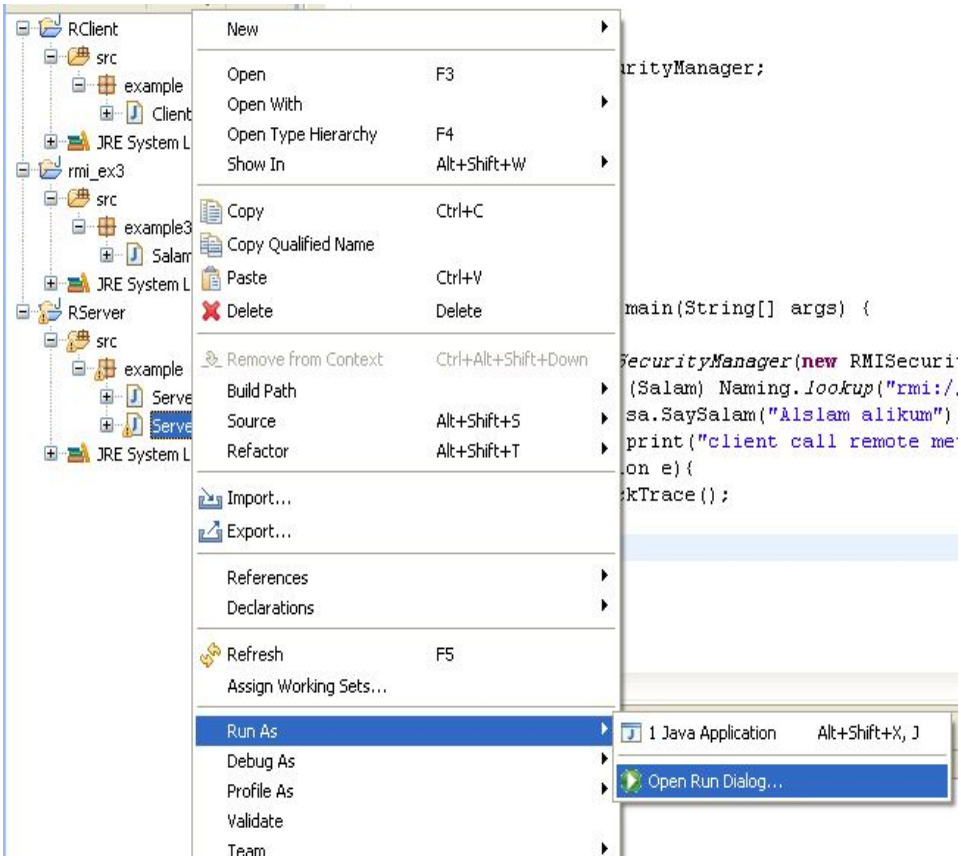


بعـدها نقـوم بتشـغيل الخـادم بـالنقر علـى Server بيمـين المـاوس الموجـود بالمشـروع RServer:

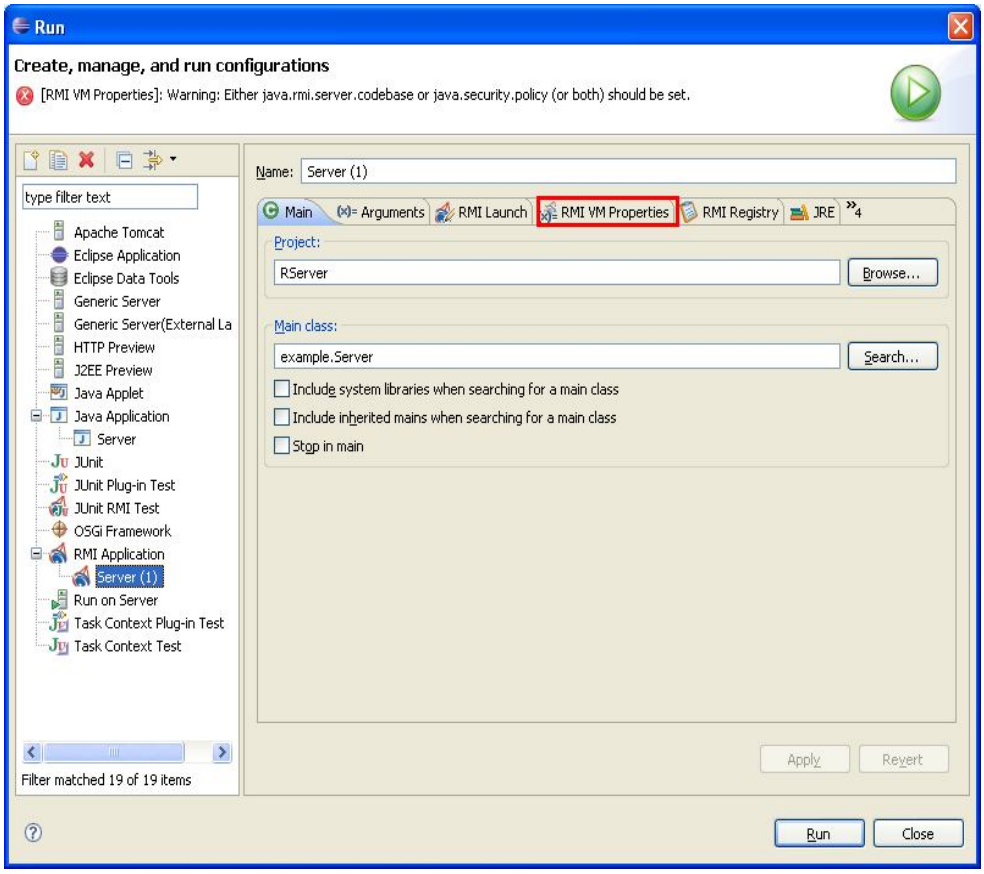

Packag
Hierarc
RClient
src
example
Client.java
JRE System Library [jdk1.6.
rmi_ex3
src
example3
Salam.java
JRE System Library [jdk1.6.
RServer
src
example
Server_Stub.java
Server.java
JRE System Library [jdk1.6.

ثم نختار :

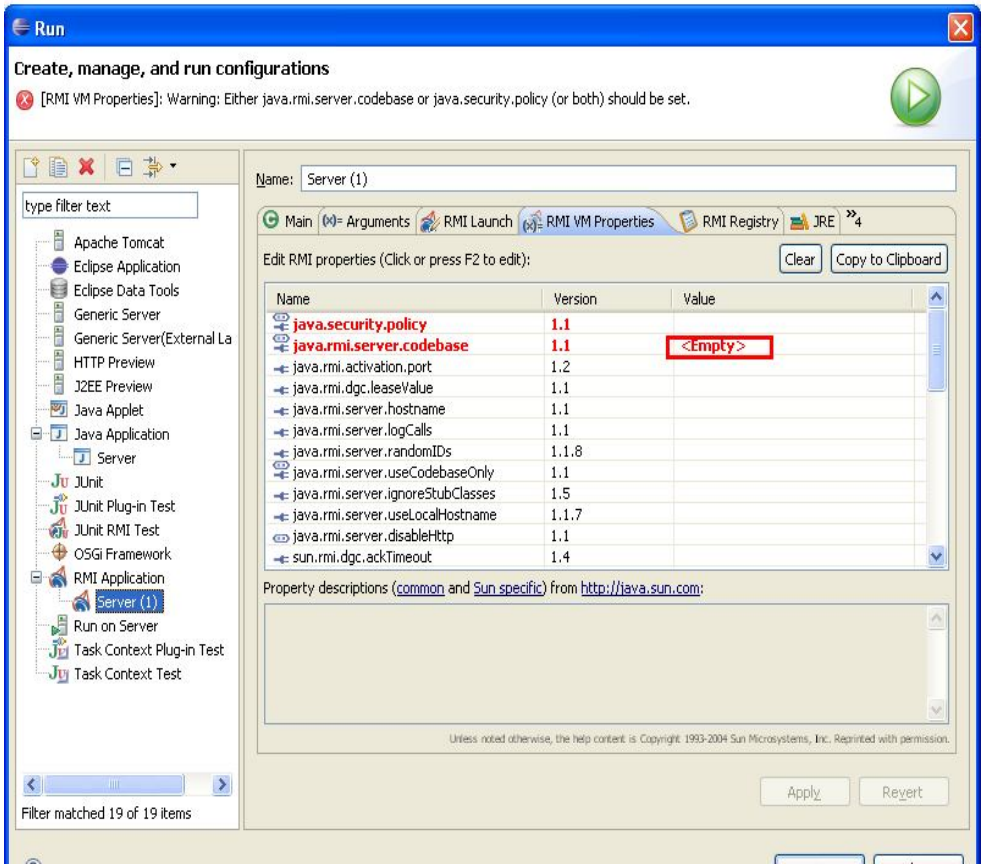

Run
Create, manage, and run configurations
type filter text
Apache Tomcat
Eclipse Application
Eclipse Data Tools
Generic Server
Generic Server(External La
HTTP Preview
J2EE Preview
Java Applet
Java Application
JUnit
JUnit Plug-in Test
JUnit RMI Test
OSGi Framework
RMI Application
Run on Server
Task Context Plug-in Test
Task Context Test
Filter matched 17 of 17 items
Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.
Configure launch perspective settings from the Perspectives preference page.
Run
Close



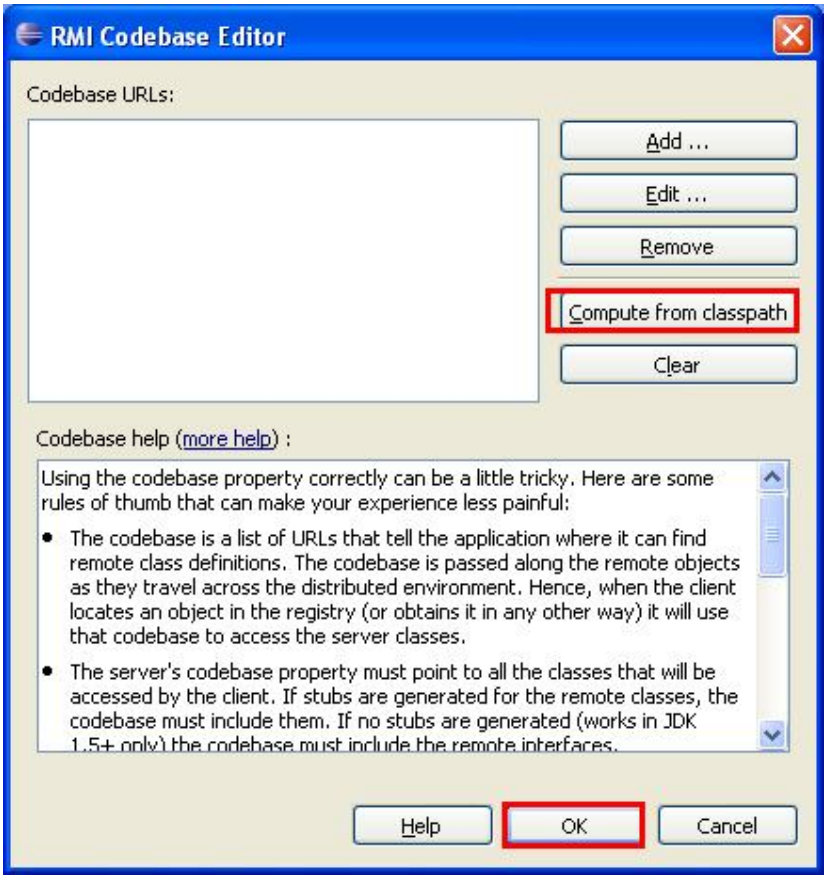
نقرتتين مزدوجتين على RMI Application



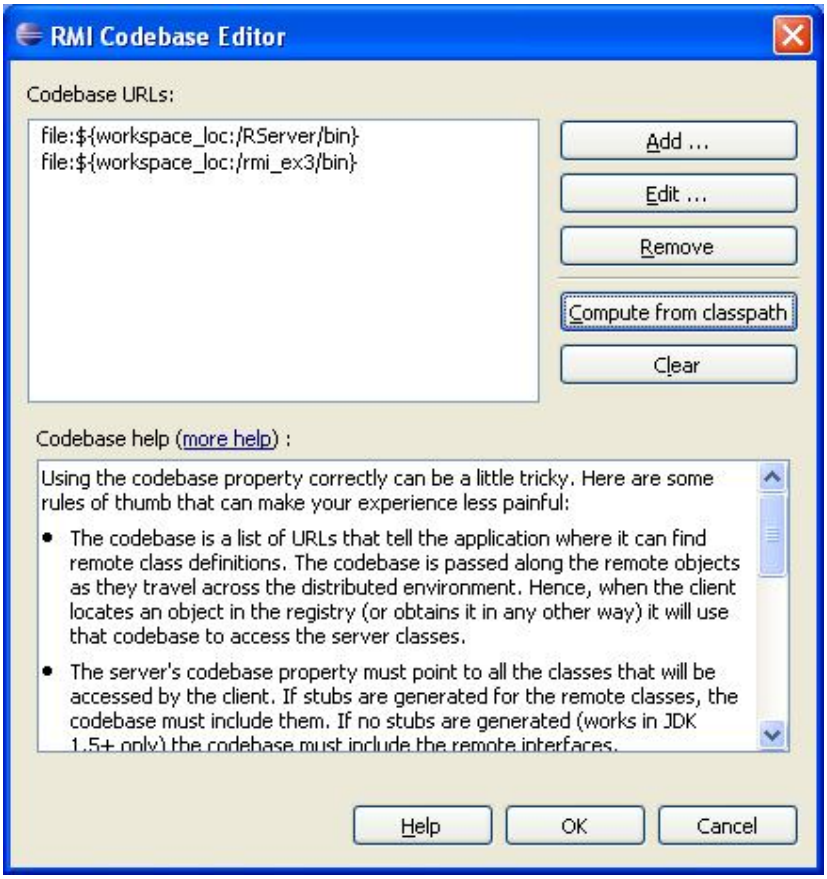
انقر على RMI VM Properties لتظهر

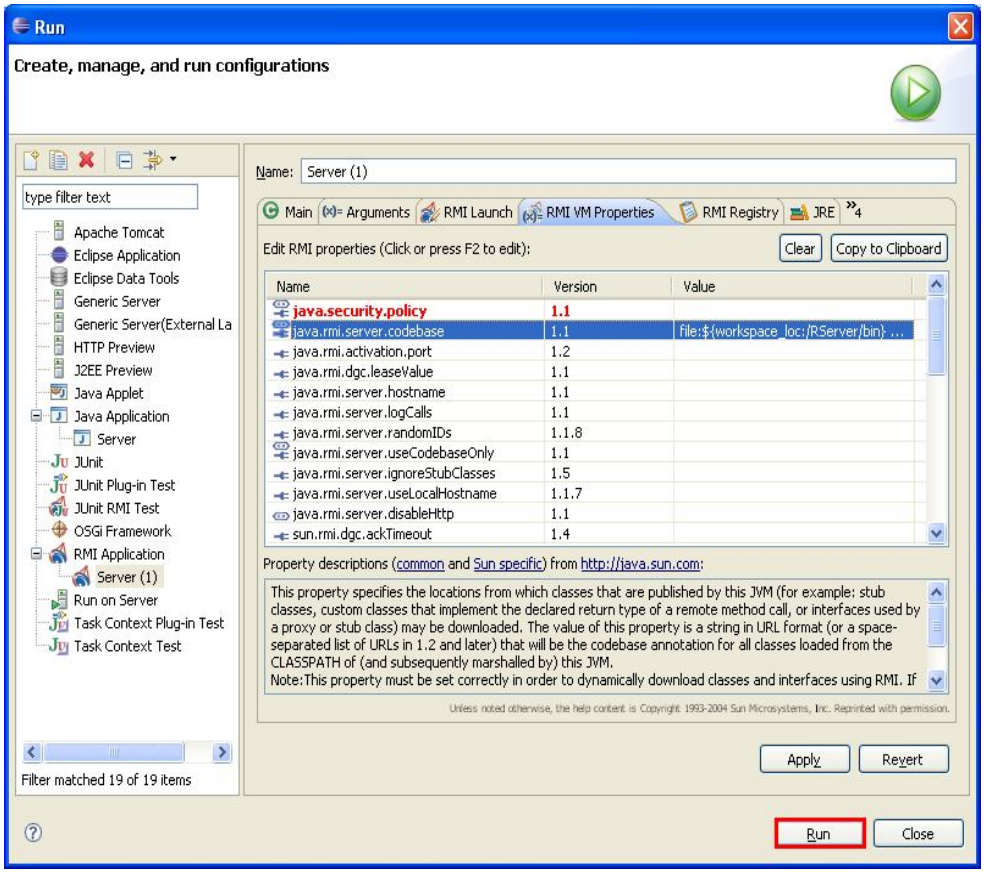


انقر على empty لتظهر



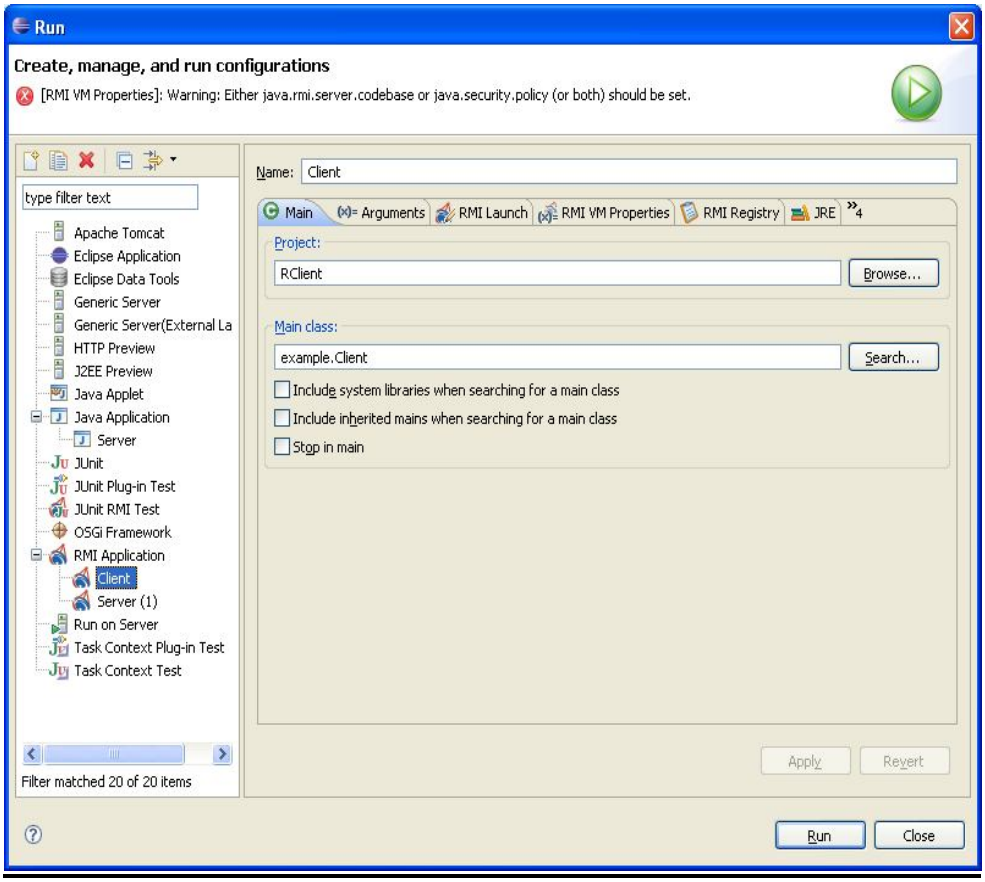
انقر على compute from classpath ثم ok


RMI Codebase Editor
Codebase URLs:
file:${workspace_loc:/RServer/bin}
file:${workspace_loc:/rmi_ex3/bin}
Add ...
Edit ...
Remove
Compute from classpath
Clear
Codebase help (more help) :
Using the codebase property correctly can be a little tricky. Here are some rules of thumb that can make your experience less painful:
The codebase is a list of URLs that tell the application where it can find remote class definitions. The codebase is passed along the remote objects as they travel across the distributed environment. Hence, when the client locates an object in the registry (or obtains it in any other way) it will use that codebase to access the server classes.
The server's codebase property must point to all the classes that will be accessed by the client. If stubs are generated for the remote classes, the codebase must include them. If no stubs are generated (works in JDK
Help
OK
Cancel



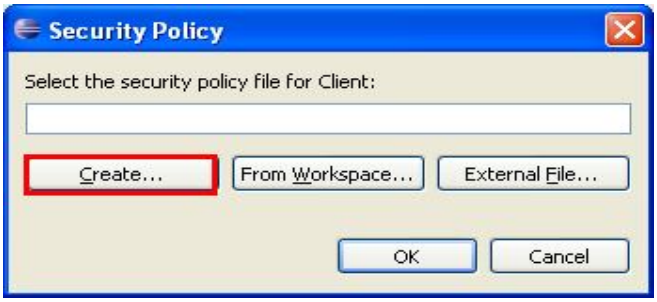
ثم Run

لتشغيل برنامج العميل، انقر يمين الماوس على Client.java ثم Run As ثم: Open run dialog



نقرتين مزدوجتين على RMI Application ثم انقر على Client ثم على

نختار tab properties VM RMI ثم نختار منه java.security.policy لتظهر النافذة



انقر على create

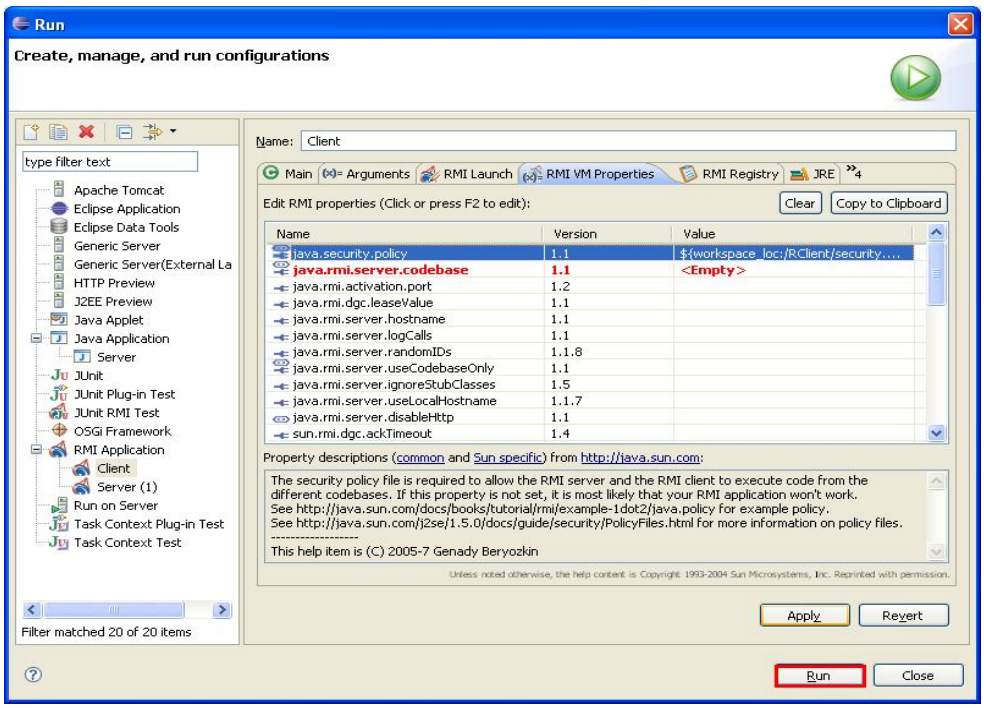


انقر على finish لتظهر الشاشة:



انقر OK

ثم Run



لترى الناتج.

```
<terminated> Client [RMI Application] C:\Program Files\Java\jdk1.6.0_03\bin\javaw.exe (Dec 7, 2007 6:49:18 AM
Server saidAlslam alikum
```

الآن وبعد تنفيذ التطبيق سنرى كم من السهل التعديل.

مثلا إذا أردت اضافة الدالة add لاستدعائها بواسطة العميل سأقوم بالآتي:

1. نضيف الكود التالي للواجهة:

```
public int add(int x, int y) throws RemoteException;
```

نضيف الكود التالي للخادم Server.java

```
public int add(int x, int y) throws RemoteException
 {return x+y;}
```

نستدعي الدالة في العميل ونظهر النتيجة بالأوامر:

```
 int x= sa.add(5,7);
System.out.print("addition="+x);
```

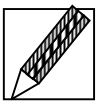
أسئلة تقويم ذاتي



1- كيف يتعرف العميل على المنفذ الذي يتصنت عليه الخادم؟

2- هل الدليل Registry مهم لتشغيل نظام RMI ؟

تدريب (1)



- **ماهي بريمجات جافا (Java Applets)**
- هنالك ثلاث عناصر أساسية أحتاجها لعمل بريمج جافا
- اكتب بريمج الصفحة الفائقة.
- اكتب البرنامج المصدري.

الخلاصة

عزيزي الدارس، يمكن اعتبارالطبقة الوسيطة برنامجاً يربط تطبيقات منفصلة مع بعضها. فالطبقة الوسيطة في النظام الموزع مسؤولة عن معالجة المسائل المتعلقة بالاتصال بين مكونات برمجية موزعة في أجهزة مختلفة. ومن مهامها أيضا توفير طريقة لتسجيل الخدمات الجديدة واكتشاف الخدمات المتاحة للعملاء.تمثل RMI طبقة وسيطة توفرها لغة جافا، حيث تسمح لكائن يعمل في آلة جافا افتراضية (JVM) من التعامل والاتصال بكائن آخر يعمل في آلة جافا افتراضية أخرى (JVM).فهي توفر اتصالاً بعيداً بين برامج مكتوبة بلغة جافا. الهدف الرئيسي من RMI توفير بيئة سهلة للمبرمجين لتطوير برامج موزعة بنفس الطريقة البسيطة التي يطوروا بها البرامج العادية (غير الموزعة). فلا يحتاج المبرمج إلى التعامل مع الشبكات والبروتوكولات وغيرها من العقبات أعلاه. فهو يستطيع بناء تطبيق موزع كامل يعمل على شبكة محلية أو واسعة دون أن يكون لديه خبرة في برمجة الشبكات، ولا كيف تعمل الشبكة.

وأيضاً تدعم لغة جافا مفهوم الكائنات الموزعة بصورة واسعة. ومن الأدوات المستخدمة في هذه اللغة لإنشاء الكائنات الموزعة والتعامل معها طبقة RMI الوسيطة (RMI middleware). سندرس هنا كيفية الاستفادة من هذه الطبقة في بناء كائنات موزعة في الشبكة وكيفية التعامل معها واستدعائها من بعيد. ثم سنقوم ببناء تطبيق موزع باستخدام هذه التقنية.

لمحة مسبقة عن الوحدة القادمة :

عزيزي الدارس، في الوحدة القادمة سنتناول خدمات طبقة الويب الوسيطة نتعرف فيها على خدمات الويب (web services)التي تعتبر نموذجاً لطبقة وسيطة (middleware)، وكيف تسمح ببناء تطبيقات موزعة في الإنترانت وأمكانية توفر بيئة سهلة لبناء التطبيقات ومن خلالها يستطيع المبرمج بناء تطبيقات موزعة تعامل مع بعضها البعض حتى ولو كانت في منصات متباينة.

أيضا سنتعرف فيها على استخدام خدمات الويب لبروتوكول SOAP الذي ينقل داخل HTTP في اتصالاتها، وكيف ترسل بياناتها في شكل XML. توضع خدمة الويب على مخدم ويب (web server) ويكون لها عنوان URL للوصول إليها.

إجابات التدريبات

بريمجات جافا (Java Applets)

- صممت بريمجات الجافا لتكون:
- 1. سريعة.
- 2. صغيرة.
- 3. بالتالي من السهل نقلها عبر الشبكة.
- بريمجات جافا ليست مثل برامج جافا التي درسناها مسبقا فهي ممنوعة من الوصول الى:
  - موارد الشبكات.
  - موارد الجهاز.
- إلا في حالات نادرة جدا. بالتالي هذا يجعل تحميل البريمج من الإنترنت إلى جهازك وأنت تتصفح الإنترنت آمنا ، لأنها لا تستطيع الوصول إلى ملفاتك ومواردك.

هنالك ثلاث عناصر أساسية أحتاجها لعمل بريمج:

1. صفحة فائقة مكتوبة بلغة html تحتوي على وسم <applet> الذي يستخدم لإدراج بريمج بالصفحة.
2. بريمج جافا المصدر (.java)
3. بريمج جافا بلغة آلة جافا (javabytecode) وهو بالامتداد (.class) وهذا ينتج من ترجمة بريمج جافا المصدري في 2 أعلاه.

1. الصفحة الفائقة:

```
<HTML>
    <HEAD>
        <TITLE>Example</TITLE>
    </HEAD>
    <BODY>
        <APPLET Code = "salam.class"
                Width = "400"
                Height = "200">
        </APPLET>
    </BODY>
</HTML>
```

بريمج جافا المصدري:

```
import java.applet.Applet;
import java.awt.Graphics;
        public class salam extends Applet {
                public void paint(Graphics g){
                        g.drawString("Aslamu Alikum…",25,25);
                }
}
```

2. نترجم البريمج أعلاه بوساطة مترجم جافا ونضع الملف المترجم (class.) في المكان الذي توجد به الصفحة الفائقة في 1 أعلاه، بنقرتين مزدوجتين على الصفحة الفائة تتنفذ بالمتصفح ويتنفذ معها البريمج.

مسرد المصطلحات

الطبقة الوسيطة Java RMI Middleware

هي المسؤولة عن معالجة المسائل المتعلقة بالاتصال بين مكونات برمجية موزعة في أجهزة مختلفة ومن مهامها توفير طريقة لتسجيل الخدمات الجديدة واكتشاف الخدمات المتاحة للعملاء.

نداء الطريقة البعيدة(RMI)

طبقة جافا الوسيطة RMI هي طبقة تمكننا من استدعاء طرق (دوال) بعيدة موجودة بكائنات منتشرة في الشبكة.

الواجهة في لغة الجافا (interfaces)

هي فئات فارغة لا يوجد بها أي شفرات فقط تعريفات الطرق (الدوال) مثل اسم الطريقة، القيمة التي ترجعها، المعطيات (parameters) التي تمررها. ثم نستخدم فئة أخرى تطبق هذه الواجهة (implement)، بطريقة تشبه مفهوم الوراثة إلى حدٍ ما.

المراجع

1- 1.Coulouris, G. et al., (2004), Distributed Systems, Concepts and Design, Addison Wesley

2- Farley, J. (1998), Java Distributed Computing, O'reilly & Associates

3- Fedorov, A, (2002), A Programmer Guide to .NET, Addison Wesley

4- Ince, D. (2002), Developing Distributed and E-Commerce Applications, Pearson Education

5- Liu, M. (2004), Distributed Computing Principles and Applications, Pearson Education

6- Schonauer, W. (1998), Scientific Supercomputing: Architecture and use of Shared and Distributed Memory Parallel Computers, Willi Schonauer, Germany

7- Tanenbaum, A., (2002), The Amoeba Distributed Operating System, Vrije Universiteit Technical Report, URL ftp://ftp.cs.vu.nl/pub/amoeba/Intro.ps.Z

8- Tanenbaum , A. AND Van Steen, M. (2002), Distributed Systems Principles and Paradigms, Prentice Hall

9- Tanenbaum, A. (1995), Distributed Operating Systems, 1/e, Prentice Hall, USA

10- Osman, A., (may 2006), A Multi-level Web-based Parallel Processing System, TRANSACTIONS ON ENGINEERING, COMPUTING AND TECHNOLOGY VOLUME 13 MAY 2006 ISSN 1305-5313

11-Forouzan, A. (2007), Data Communications and Networking, Mc Graw Hill, Fourth Edition

12-Tanenbaum, A., (2001), Modern Operating Systems, Prentice Hall, Second Edition

13-Tanenbaum, A., (1999), Structred Computer Organization, Prentice Hall, Fourth Edition


الوحدة الحادية عشر
طبقة خدمات الويب الوسيطة
Web Services Middleware

محتويات الوحدة

الموضوع	الصفحة
المقدمة	**255**
تمهيد	**255**
أهداف الوحدة	**255**
1. طبقة خدمات الويب الوسيطة	**256**
2. صفات خدمات الويب	**256**
3. أجيال الويب	**257**
1.3 الجيل الأول (الصفحات الساكنة)	**258**
2.3 الجيل الثاني (الصفحات الديناميكية)	**259**
3.3 الجيل الثالث (خدمات الويب(web services))	**260**
4. الاختلاف بين المواقع وخدمات الويب:	**261**
الخلاصة	**292**
إجابات التدريبات	**293**
مسرد المصطلحات	**294**
المراجع	**295**

المقدمة

تمهيد

عزيزي الدارس مرحبا بك في الوحدة الأخيرة من هذا المقرر والتي بعنوان طبقة خدمات الويب الوسيطة نتعرف فيها علي خدمات الويب (web services) التي تعتبر نموذجاً لطبقة وسيطة (middleware)، وكيف تسمح ببناء تطبيقات موزعة في الإنترانت وإمكانية توفر بيئة سهلة لبناء التطبيقات ومن خلالها يستطيع المبرمج بناء تطبيقات موزعة تعامل مع بعضها البعض حتى ولو كانت في منصات متباينة.

أيضا سنتعرف فيها علي استخدام خدمات الويب لبروتوكول SOAP الذي ينقل داخل HTTP في اتصالاتها، وكيف ترسل بياناتها في شكل XML. توضع خدمة الويب على مخدم ويب (web server) ويكون لها عنوان URL للوصول إليها.

أهداف الوحدة



عزيزي الدارس بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

1. تبين البروتوكول المستخدم في خدمة الويب (web service)
2. تفهم خصائص الصفحات الساكنة والصفحات الديناميكية وخدمات الويب
3. تفرق بين لغة HTML ولغة XML

1. طبقة خدمات الويب الوسيطة

خدمات الويب (web services) تعتبر نموذجاً لطبقة وسيطة (middleware)، تسمح ببناء تطبيقات موزعة في الإنترانت والإنترنت. حيث توفر بيئة سهلة لبناء التطبيقات ومن خلالها يستطيع المبرمج بناء تطبيقات موزعة تعامل مع بعضها البعض حتى ولو كانت في منصات متباينة.

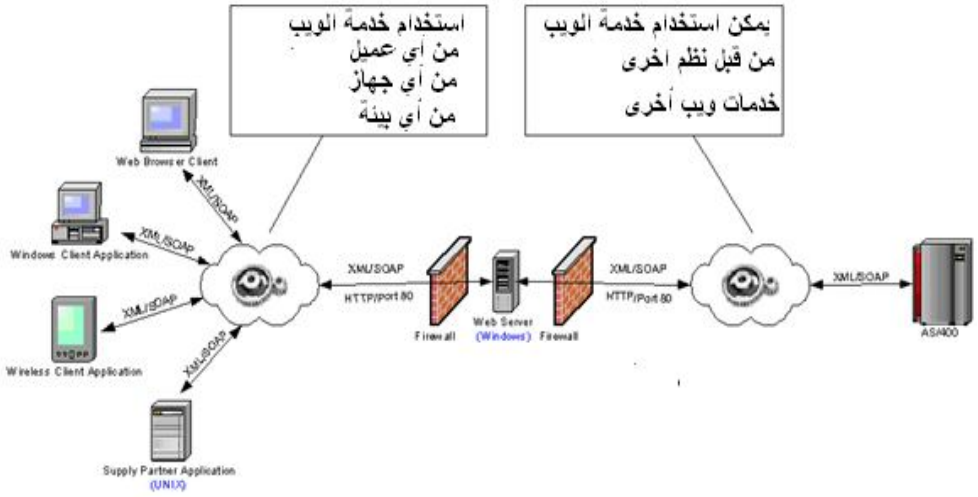
تستخدم خدمات الويب بروتوكول SOAP الذي ينقل داخل HTTP في اتصالاتها، وترسل بياناتها في شكل XML. توضع خدمة الويب على مخدم ويب (web server) ويكون لها عنوان URL للوصول إليها.



2. صفات خدمات الويب

- سهلة الإنشاء.
- توفر **خدمات** لتطبيقات أخرى.
- ليس لها واجهة مستخدم (user interface) وإنما لها واجهة للبرامج (program interface).
- تتيحها خدماتها عبر الإنترنت أو الإنترانت.
- يمكن استدعاء طرقها من بعيد (تطبق نداء الإجراء البعيد (RPC).)
- توفر خاصية الشفافية (transparency).
- تتعامل مع منصات متباينة (platform) وأجهزة مختلفة (خاصية التباين في الأجهزة والبرامج).

- تعمل مع تطبيقات الويب وتطبيقات الويندوز.
- لا توقفها جدر النار (firewall) لأنها تستخدم بروتوكول HTTP.



شكل رقم (11-2): خدمة ويب (web service) تتعامل مع بيئات متباينة

3. أجيال الويب

انتشار الويب وتطورها بلغ مبلغاً عجباً فأصبح كل شي مرتبط بها. هذا التطور بدأ بصفحات html ساكنة تعرض محتواها للمستخدم بصورة ثابتة. ثم تطور وتوسع إلى أن وصل ما نحن عليه. هذا التطور كان متمرحلاً على ثلاث أجيال هي:

1. جيل الصفحات الساكنة (static pages).
2. جيل الصفحات الديناميكية (dynamic pages).
3. جيل خدمات الويب (web services).

أدناه سنوضح تفاصيل هذه الأجيال.

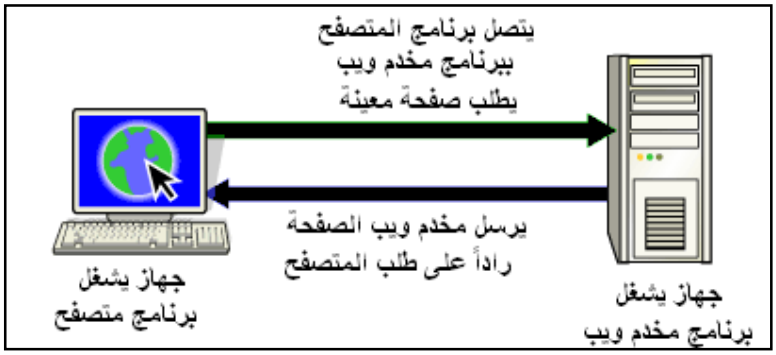
1.3 الجيل الأول (الصفحات الساكنة)

بدأت الويب بصفحات ثابتة المحتوى (ساكنة)، فمحتواها واحد لكل المستخدمين و يظهر بنفس الشكل وفي كل الأوقات، تسمى هذه الصفحات صفحات ساكنة (static pages). طريقة عملها بسيطة جدا، وهذا سبب انتشارها. فكل ملف html يخزن بالويب يمكن استعراضه كصفحة ويب (كل صفحة تكون مخزنة في ملف).

طريقة تخزين الصفحة واستعراضها بسيط جدا حيث يكون كالتالي:

1. نصمم الصفحة بلغة html ثم نخزنها بمخدم ويب.
2. يحصل المستخدم على عنوان هذه الصفحة (URL).
3. يفتح المستخدم المتصفح بجهازه ويكتب عنوان الصفحة (URL).
4. يرسل برنامج المتصفح طلب الصفحة لبرنامج مخدم ويب بمخدم الويب.
5. يرد برنامج مخدم ويب مرسلاً الصفحة المطلوبة للمتصفح.
6. بروتوكول http ولغة html هما أساس تبادل وعرض الصفحات ويستخدمان منفذ **80** الذي لا تغلقه برامج الحماية (جدار النار) وبالتالي لا شيء يوقف عملية التصفح.

عنصري هذا الجيل الأساسين هما بروتوكول HTTP ولغة HTML.



شكل رقم (11-3): تصفح الويب

2.3 الجيل الثاني (الصفحات الديناميكية)

عيوب الصفحات الساكنة أن تعديلها صعب ويستغرق وقتاً كبيراً ، فمثلاً لو كان لدي موقع مؤلف من مائة ملف صفحة وأردت تعديل معلومة واحدة في كل هذه الصفحات فلابد من الآتي:

1. فتح كل الملفات وتعديلها.
2. إعادة تحميل الصفحات إلى مخدم ويب.

أيضاً من مشاكل الصفحات الساكنة عدم وجود خصوصية للمستخدم، فمثلاً لا طالب مشاهدة معلوماته الخاصة به في صفحة، فلا يمكن للصفحة الساكنة القيام بذلك، فلا بد من عرض كل الدرجات لكل الطلاب فيراها الجميع وإلا فلا. أيضاً لن يكون هنالك بريد إلكتروني خاص بك فكل الرسائل ستعرض للجميع. ولن يكون هنالك اسم مستخدم أو كلمة مرور أو غيرها.

صعوبة التعديل وعدم وجود الخصوصية أسهمت في ظهور الصفحات الديناميكية. هذه الصفحات تظهر محتواها غالباً من قاعدة بيانات. فإذا أردت تعديل صفحة فلن أحتاج إلى فتح كل ملفات html وإنما أعدل في محتوى قاعدة البيانات فيتغير محتوى الصفحات كلها تلقائياً، وبالتالي لن أحتاج لإعادة رفع الصفحات بمخدم الويب مرة ثانية.

> لغة html ما هي إلا أداة للتنسيق مثلها مثل ميكروسوفت وورد ، الفرق بينها وبينه أنها تنسق بيانات لتظهر على المتصفح بينما ينسق الوورد (MS Word) البيانات لتظهر على الورق.

لا تستطيع لغة html التعامل مع قواعد البيانات لإظهار محتواها على الصفحة، لذلك نستخدم لغات تسمى لغات برمجة إنترنت، هي لغات مصممة خصيصا لإنشاء الصفحات الديناميكية، فهي تأخذ محتوى قاعدة البيانات ثم تنسقه بلغة html ليظهر على

المستعرض كصفحة ساكنة. بمعنى آخر تنشئ لغة برمجة الإنترنت من قاعدة البيانات صفحات ساكنة عند ما يفتحها المستخدم.
الصفحات الديناميكية تكون الجيل الثاني من أجيال ويب.

تسمى ديناميكية لأنها تظهر محتواها من قاعدة بيانات وبالتالي لو تغير محتوى قاعدة البيانات سيتغير محتوى الصفحة. حيث تقوم أكواد نكتبها بلغة برمجة إنترنت بإنشاء الصفحة ديناميكياً بتعبئة محتواها من قاعدة البيانات وتنظيمها في شكل صفحة html.

تختلف طريقة عمل الصفحات الديناميكية عن عمل الصفحات الساكنة في أنها تصمم بلغة برمجة إنترنت مع لغة html ثم نحملها بمخدم الويب، عندما يطلب المستخدم صفحة ديناميكية يتم تنفيذ شفرات لغات البرمجة لتنشأ صفحة ساكنة تعرض على المتصفح.

3.3 الجيل الثالث (خدمات الويب(web services))

خدمات الويب (Web Services) هي عبارة عن برامج مكتبية، تخزن بمخدم ويب وتكون متاحة للكل، ولاستخدامها لابد من معرفة مكانها وأسماء الدوال التي بها و نوع الخدمات التي تقدمها هذه الدوال. حيث نرسل طلب باسم الدالة التي نريد تنفيذها (Request)، فتستجيب له خدمة الويب وتنفذه بالجهاز البعيد ثم ترسل الرد على الطلب بنتائج تنفيذ هذه الدالة (response).
تشبه خدمات الويب في تخزينها والوصول إليها مواقع الويب التي نتصفحها يومياً ولكنها تختلف عنها في الكثير.

4. الاختلاف بين المواقع وخدمات الويب:

الجدول التالي يوضح الاختلاف بين المواقع وخدمات الويب:

المقارنة	المواقع	خدمات الويب (web services)
البروتوكول	HTTP	SOAP/HTTP
تمثيل البيانات	HTML	XML
الفائدة منها	عرض معلومات	توفير خدمات
المستفيد	المستخدمين	التطبيقات
كيف نستخدمها	العنوان URL	مكان خدمة ويب (URL)، نوع الخدمات التي توفرها وطريقة طلب الخدمات (استدعاء الدوال (methods)).
البحث	محركات البحث	UDDI

الفائدة الرئيسية من خدمة الويب هي إعادة استخدام الكود (code reuse)، فلو كان لدى برنامج أريد استخدامه أكثر من مرة يمكني عمله في شكل خدمة ويب أتاحته لجميع التطبيقات للاستفادة منه واستخدامه كأنه مكتوب بالتطبيق. فهو هنا يطبق مفهوم الوراثة في اللغات الكائنية التوجه ولكن على أوسع.

في الوراثة تستفيد الكائنات من فئة قريبة (بنفس الحاسب)، أما في خدمات الويب فيستفيد التطبيق من فئة بعيدة (بالإنترنت).

تساعد خدمات الويب على بناء تطبيقات موزعة بسهولة ويسر. فهي تعمل على منصات مختلفة ويمكن الوصول إليها واستدعاء دوالها من أي مكان.

عنصراً هذا الجيل الأساسيان هما بروتوكول SOAP ولغة XML.

أمثلة لخدمات ويب متوفرة عبر الإنترنت:

1. خدمات الويب الخاصة بحالة بالطقس. حيث تستدعي دالة وتمرر لها قيم (مثلا المنطقة التي تريد حالة طقسها) فيأتيك الناتج وكأن خدمة الطقس مبنية بتطبيقك.
2. خدمات الويب التي توفرها جوجل (Google web service) للبحث عن مواقع بالإنترنت. حيث تستطيع إضافة خدمة البحث عن المواقع في تطبيقك دون كتابتها.

مثال عملي (Practical example)

تدعم كل من شركة صن وشركة ميكروسوفت خدمات الويب. حيث وفرت ميكروسوفت خدمات الويب في بيئة دوت نيت (Net.).

سنرى هنا كيف يمكننا بناء خدمة الويب في Net. وكيفية استخدامها لبناء تطبيق موزع (distributed application).

ولكن قبل الشروع في بناء التطبيق لابد من معرفة بعض المصطلحات والتعريفات التي نحتاج لها في عملنا هذا.

اختصارات متعلقة بخدمة الويب

UDDI

هو اختصار للكلمات Universal Description, Discovery and Integration وهو دليل يستخدم للآتي:

- تخزن به معلومات عن خدمات الويب (ليعرف العميل موقع الخدمة بالإنترنت).
- تخزن به واجهات خدمات الويب (interfaces)، ليستطيع العميل التعامل معها.
- مستقل عن أي منصة (platform independent).
- ترسل المعلومات في شكل XML (مستقل عن أي لغة).
- يستخدم بروتوكول SOAP.

WSDL

هو اختصار للكلمات Web Services Description Language.

- مكتوبة بلغة XML.
- هي وثيقة XML تصف خدمات الويب.
- تستخدم لمعرفة مكان خدمة الويب.
- تستخدم لمعرفة نوع الخدمات التي توفرها خدمة الويب وكيفية استدعاءها.

SOAP

هو بروتوكول واختصار للكلمات Simple Object Access Protocol.

- بروتوكول للاتصال.
- يستخدم للاتصال بين التطبيقات.
- يمثل صيغة لإرسال الرسائل.
- مستقل عن أي منصة (platform independent).
- مستقل عن أي لغة (language independent).
- مبني على XML.
- يستطيع عبور جدر النار.

XML

إختصار للكلمات Extensible Markup Language.

- تستخدم لتمثيل البيانات.
- تستخدم لتبادل المعلومات بين بيئات مختلفة (different platforms).
- تستخدم لتبادل المعلومات بين لغات مختلفة.
- تستخدم لتبادل المعلومات بين نظم قواعد بيانات مختلفة.
- تستخدم لنقل رسائل معقدة وشفرات.
- تعتمد على البروتوكول SOAP.

HTTP

هو بروتوكول و حروفه **h**اختصار من الكلمات Hypertext Transfer Protocol.

- أكثر بروتوكول مستخدم بالإنترنت.
- يستخدم لنقل المعلومات في الويب (www).
- يستخدم لنشر وعرض الصفحات المكتوبة بلغة HTML.
- هو بروتوكول طلب (request) و رد (response) بين العميل والخادم.
- تستخدم بين برنامج المتصفح (browser) و برنامج مخدم ويب (web server).
- يستخدم المنفذ 80 للتراسل.
- جدر النار لا تغلق هذا المنفذ (إذا أغلق لن يستطيع شخص تصفح الويب).

تتبادل خدمات الويب (web services) البيانات في شكل XML عبر بروتوكول SOAP، لكن بروتوكول SOAP هو بدوره يستخدم بروتوكول HTTP (حيث تكون رسائل SOAP مغلفة داخل رسائل HTTP) فتمر عبر الشبكة كأنها HTTP وهذا يجعل حركتها سهلة ولا تصدها برامج الحماية (جدر النار)، ذلك لخاصية HTTP (يستخدم المنفذ 80 والذي لايغلقه أي جدار نار).

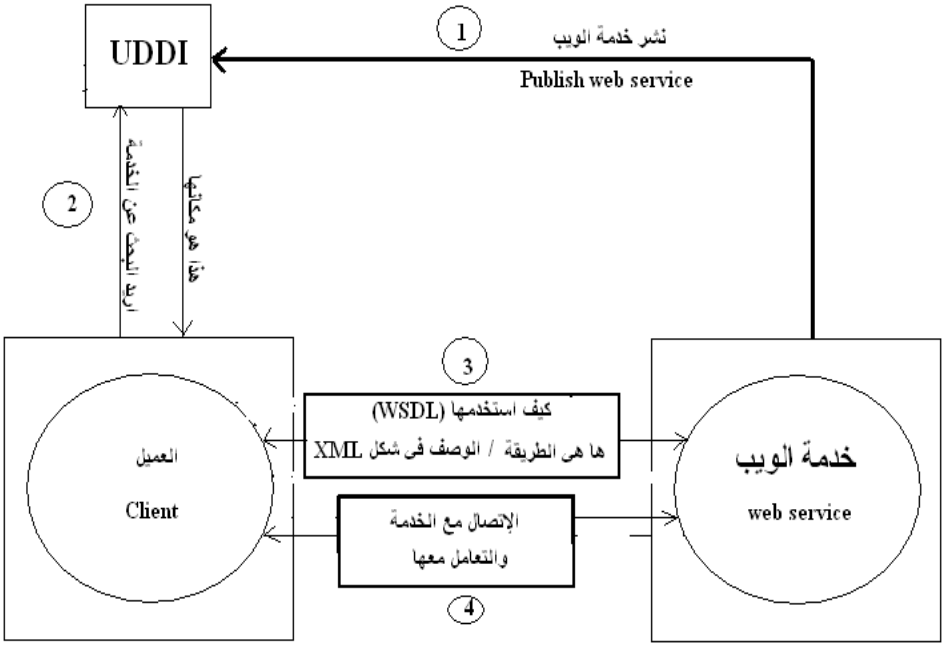
أسئلة تقويم ذاتي

1. ما هو الفرق بين الصفحات الساكنة والصفحات الديناميكية وخدمات الويب ؟
2. ما هو الفرق بين لغة HTML ولغة XML ؟
3. ما هو الفرق بين UDDI و WDSL ؟

لاستخدام خدمة ويب لابد من التالي:

1. معرفة مكان خدمة الويب (بالبحث في دليل UDDI).
2. معرفة كيفية استدعاء الخدمات التي توفرها خدمة الويب بالحصول على واجهات الخدمة من WSDL.
3. الإستفادة من هذا الوصف في استدعاء الخدمات (استدعاء الدوال الموجودة بالواجهات).

الخطوات أعلاه تتم في شكل رسائل XML وعبر بروتوكول SOAP المغلف داخل بروتوكول HTTP.



شكل رقم (11-4): نشر خدمة الويب والتعامل معها

إنشاء خدمة ويب

من السهولة بمكان إنشاء خدمة ويب باستخدام فيجوال ستديو دوت نيت، فما هي إلا خطوات بسيطة كما سنرى أدناه، نقوم بها لإنشاء خدمة الويب.

خدمة الويب باختصار هي فئة مثلها مثل أي فئة أخرى (مكونة من متغيرات وطرق) وتختلف عن الفئة العادية في الآتي:

1. أنه يمكن استدعاءها من بعيد عبر الشبكة (distributed object).
2. نكتب أمام كل طريقة (method) بهذه الفئة الكلمة المحجوزة WebMethod لتجعل الطريقة متاحة للأجهزة البعيدة. الدوال التي لانضع أمامها هذه الكلمة تعتبر خاصة لايمكن الوصول إليها.
3. إمتداد الملف (الذي يحوي هذه الفئة) يكون asmx.

الآن يمكننا أن نبدأ التطبيق ...

<u>بناء خدمة الويب</u>

البرنامج التالي يوضح خدمة ويب مكتوبة بلغة سي شارب (C#).

```
1  <%@ WebService Language ="C#" Class="Salam" %>
2  using System.Web.Services;
3
4   public class Salam: WebService
5   {
6    [WebMethod]
7    public string SaySalam()
8    {
9     return "Aslamu Alikum";
10   }
11   [WebMethod]
12   public int add(int x, int y)
13   {
14    return x+y;
15   }
16  }
```

شرح البرنامج:

البرنامج عبارة عن فئة اسمها Salam وهي ترث فئة أخرى اسمها WebService، وبهذه الفئة طريقتين هما SaySalam و add. فقط هذا كل مافي الموضوع. الفئة Salam أعلاه تمثل خدمة ويب، إنتهى.

يستطيع أي تطبيق استدعاء الطرق أعلاه إذا توفر له شيئان:

1. مكان خدمة الويب (للوصول إليها)، وهذه توفر بواسطة UDDI.
2. واجهة خدمة الويب (لمعرفة أسماء الطرق بها)، وهذه توفر بواسطة WSDL.

السطر 1:

1. Webservice وهي كلمة محجوزة تعني أنني أريد إنشاء خدمة ويب.
2. Language: معناها اللغة المستخدمة لإنشاء خدمة الويب (هنا #C).
3. Class="Salam": اسم الفئة التي تحتوي خدمة الويب.

السطر 2:

1. using معناه أنني أريد استخدام المكتبة (System.Web.Services) التي تحوي كل ما نحتاجه لإنشاء خدمة ويب ، وأهم فئة في هذه المكتبة هي WebService.

السطر 3:

1. Public class salam: تعني أننا نريد إنشاء فئة خدمة ويب تسمى salam ، وهنا لابد من أن تنتبه أن اسم الفئة هنا يطابق اسم الفئة في السطرالأول.
2. WebService: مقصود بها أن الفئة salam ستورث صفات الفئة المكتبية WebService، أي أن كل ما يوجد بالفئة Webservice أصبح متاحاً للفئة salam ويمكنها استخدامه.

السطر 6:

[WebMethod]: موجه يكتب أمام الطرق التي أريد اتاحتها للعملاء.

السطر 7:

طريقة ويب ترجع قيمة نصية.

السطر 11:

طريقة ويب أخرى تقبل رقمين وترجع حاصل جمعهما.

نستخدم مع الموجه [WebMethod] الكلمة public لإتاحة الدالة للوصول العام. حيث تمثل public إمكانية استدعاء الدالة من خارج الفئة ، بينما WebMethod تعني إمكانية نداء الدالة من من بعيد.

وضع الخدمة بمخدم ويب

لابد من وجود برنامج مخدم ويب بالجهاز الذي سنضع به خدمة ويب (سواء كان في نفس الجهاز أو في الإنترانت (شبكة محلية) أو بشبكة واسعة أو بالإنترنت).

برنامج مخدم ويب هو برنامج يعمل بالخادم ويتفاهم مع المتصفح ويستخدم البروتوكول http غالباً . وهو الذي يرسل الصفحات التي نكتب عنوانها في شريط العناوين بالمتصفح كرد على طلبنا عبر المتصفح. فالمتصفح يرسل عنواننا الذي كتبناه لمخدم الويب ليعرف من خلاله مكان الصفحة التي نريد فيرسلها لنا.

سأقوم هنا بإنشاء مجلد بالسواقة C وأسميه ws_folder ثم أحفظ بداخله الملف خدمة ويب أعلاه بالاسم salam.asmx. ثم أضع هذا المجلد داخل مخدم ويب المحلي بالخطوات التالية:

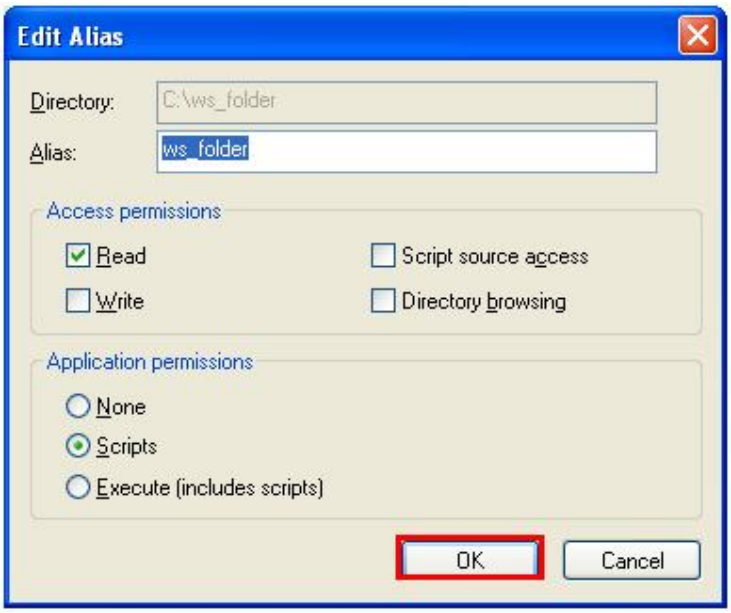
1. أنقر يمين الماوس على هذا المجلد



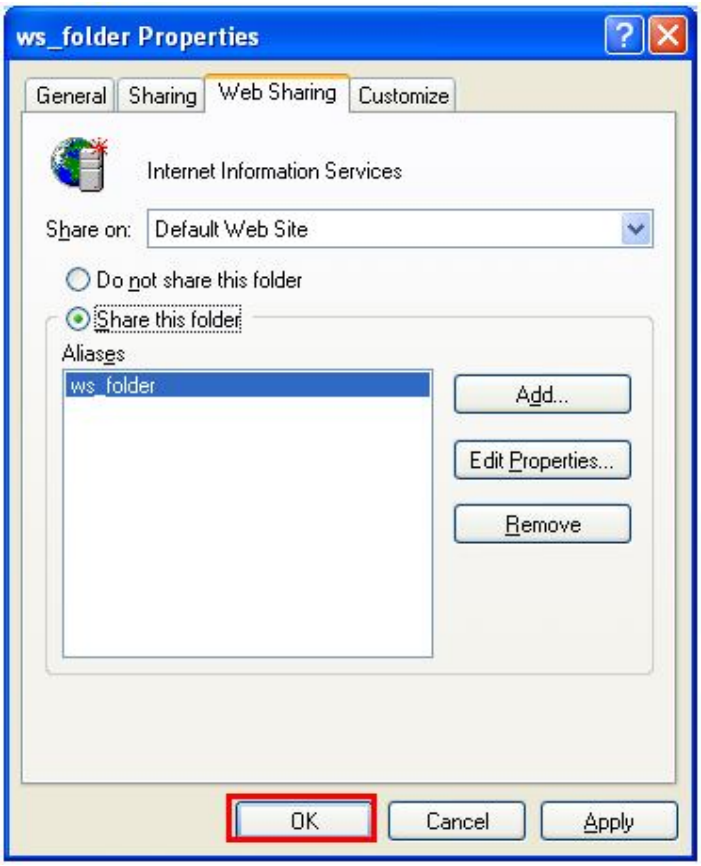
2. ثم أختار Sharing and security لتظهر الشاشة التالية:



3. ثم أختار التبويب web sharing ثم أنقر على folder this share فتظهر الشاشة التالية:



أنقر على OK.



أنقر على OK.

4. هكذا أصبح المجلد داخل مخدم ويب المحلي ويمكنني عرض محتوياته بواسطة الأمر http://localhost/ws_folder/

اختبار خدمة الويب salam

إذا أردت اختبار خدمة ويب وتجربة تنفيذ طرقها مباشرة ما عليك سوى كتابة اسمها وعنوانها في المتصفح (URL) كما نفعل مع مواقع الويب العادية ، فتظهر أسماء كل طرقها، بالنقر على اسم الطريقة يتم تنفيذها وتظهر مخرجاتها على المتصفح في شكل XML.



شكل رقم (11-5): اختبار خدمة الويب salam

مثلاً لتجربة خدمة ويب أعلاه نكتب على شريط العنوان بالمتصفح :

http://localhost/ws_folder/salam.asmx

فتظهر الشاشة التالية وبها الطرق المتاحة للإستدعاء البعيد.



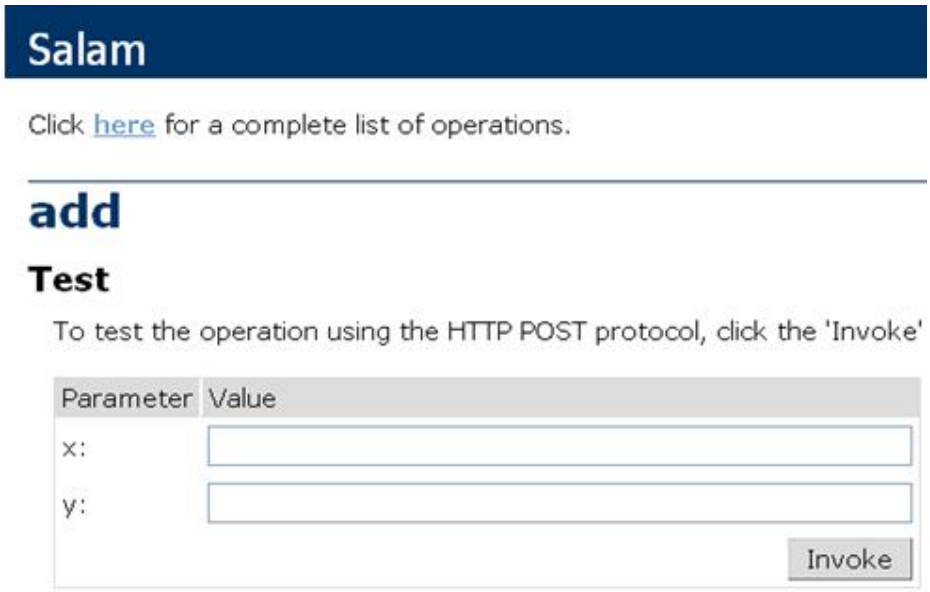
إذا أردت تجربة أي طريقة من هذه الطرق أنقر عليها، مثلا لو نقرنا على الطريقة SaySalm يظهر التالي:



بالنقر على Invoke سيتم استدعاء الطريقة و يظهر الرد في شكل XML كما يلي:



أما إذا نقرت على add فستظهر الدالة التالية:



حيث كل مربع نص يمثل معطى من معطيات الطريقة add ، إذا أدخلت رقمين في مربعي النص مثلا 5 و 7 ثم نقرت Invoke سيظهر التالي:



حيث 12 يمثل القيمة الراجعة بعد تنفيذ الطريقة add.

هكذا وضعنا خدمة الويب داخل مخدم ويب المحلي واستدعيناها محلياً لتجربتها. الخطوة التالية هي كيف نستدعيها من داخل تطبيق موزع (عميل ويب).

بناء عميل يستخدم خدمة ويب salam

بعد تجربة خدمة الويب والتأكد من عملها، يمكننا بناء برنامج يقوم باستدعاء طرق خدمة الويب من بعيد.

وليتم ذلك لابد أولاً من إنشاء وكيل (proxy) لخدمة الويب بالجهاز المحلي.

الوكيل (proxy): هو عبارة عن واجهة لخدمة ويب البعيدة تكون موجودة بالجهاز المحلي ليتعامل معها التطبيق وكأنها خدمة الويب البعيدة.

يتعامل التطبيق مع الوكيل (proxy) كالآتي:

1. يقوم التطبيق بإستدعاء طريقة خدمة الويب من الوكيل.
2. يستلم الوكيل (موجود مع التطبيق في نفس الجهاز) الاستدعاء من التطبيق.
3. يقوم الوكيل بتحزيم أوامر الاستدعاء ومعطيات الدالة وإرسالها للجهاز البعيد (الذي توجد به خدمة الويب)، ثم ينتظر الرد.
4. يتم تنفيذ الطريقة بخدمة الويب الموجودة بالجهاز البعيد.
5. يرسل الجهاز البعيد النتائج للوكيل بالجهاز المحلي.
6. يحول الوكيل النتائج للتطبيق.

دون أن يعلم التطبيق بكل هذه الخطوات التي تمت فهو يتعامل مع الوكيل كأنه هو من قام بتنفيذ الطرق (وهكذا تتحقق الشفافية).

واجهات خدمة ويب

واجهات خدمة ويب (interfaces) التي يتكون منها الوكيل (proxy) تنشأ بواسطة WSDL، حيث يرسل العميل اسم الخدمة مع الامر WSDL ، كأنه يقول لخدمة الويب "كيف اتعامل معك؟". فترسل خدمة ويب تعريفات خدماتها في شكل XML مفادها "هاهي الطريقة التي يمكنك التعامل بها معي"، انظر شكل (11-4).

التعامل مع خدمة الويب Salam

هنا نحاول الرد على الآتي:

- كيف يستطيع العميل الاتصال بخدمة الويب ؟
- كيفية إنشاء وكيل (proxy) لخدمة ويب موجودة ؟
- كيف ننشئ عميل:
  - ويب
  - واب

كيف يتصل العميل مع خدمة الويب

إذا كان لدي خدمة ويب في الموقع B وهنالك مستخدم في الموقع A فسيتم الاتصال عبر الانترنت لتنفيذ الطريقة البعيدة بخدمة ويب بالموقع B (مخدم ويب). حيث يتم الاتصال عن طريق البروتوكول SOAP على HTTP.



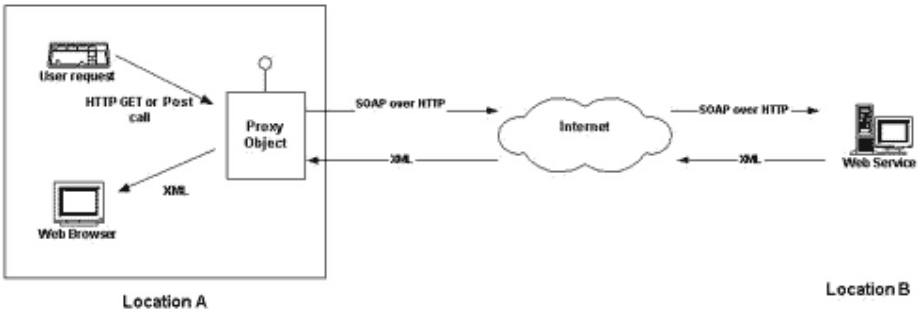
شكل رقم (11-6): استدعاء طريقة بعيدة بخدمة ويب

هل يقوم العميل حقيقة بإستدعاء الدوال الموجودة بخدمة الويب بالموقع B ؟ لا، وإنما يتم عمل نسخة مكررة (replicate) من خدمة ويب البعيدة (بالموقع B)، ونضع نسخة منها بالموقع A. هذه النسخة المحلية والتي تحتوي نفس واجهات النسخة الأصلية بالخادم

نسميها الوكيل (proxy). بعدها يكون هنالك تزامن بين الوكيل (بجهاز العميل) وخدمة الويب بجهاز الخادم.

سيقوم برنامج العميل بإرسال طلباته للوكيل المحلي ، يتصل بعدها الوكيل بخدمة الويب بالخادم ، فتنفذ خدمة ويب الأوامر وتسلم النتيجة للوكيل، فيسلمها للعميل.

كيف يتم ذلك؟



شكل رقم (11-7): وضع الوكيل بين المستخدم وخدمة الويب

الشكل (11-7) يوضح ذلك.

الوكيل هو كائن يمثل طريق الوصل الوحيد بين العميل وخدمة الويب.

إذا أول خطوة لإنشاء عميل لخدمة الويب هي إنشاء وكيل. بعدها يمكننا إنشاء عملاء مختلفين من منصات مختلفة للتعامل مع هذا الوكيل.

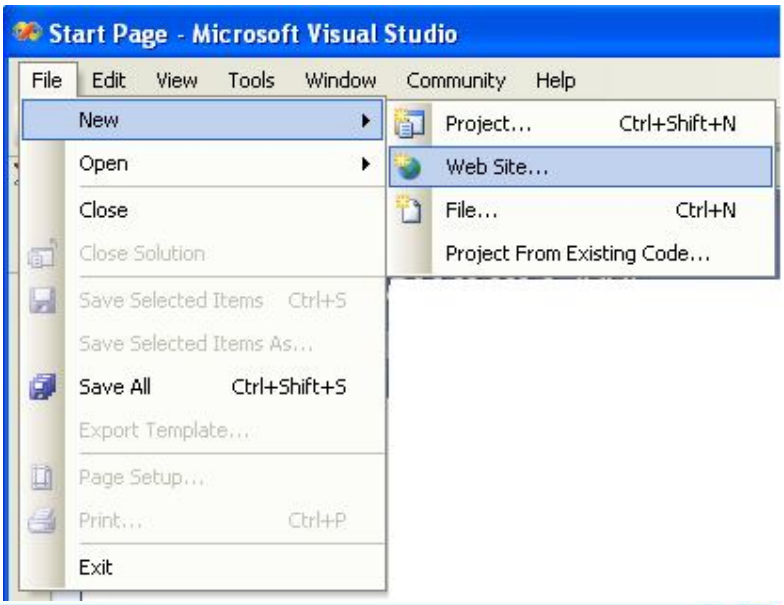
<u>إنشاء وكيل لخدمة ويب</u>

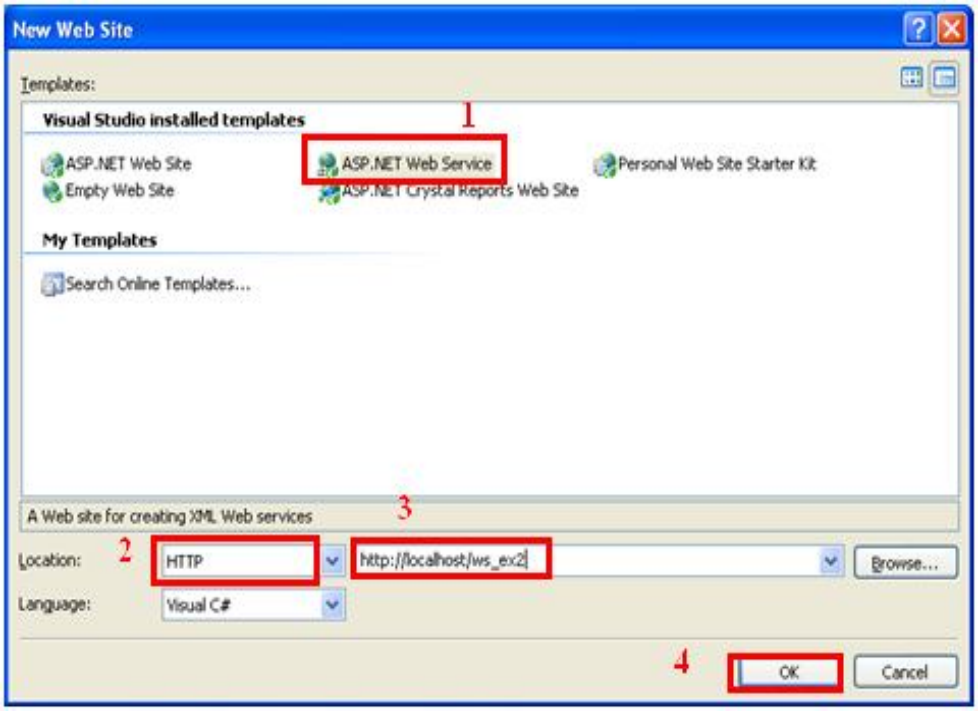
يمكننا إنشاء وكيل باستخدام برنامج WSDL.exe الذي يأتي مع حزمة .NET Framework.

<u>استخدام فيجوال ستديو 2005</u>

يمكنني استخدام فيجوال استديو دوت نيت لبناء خدمات الويب وإنشاء وكيل لها، وتصميم تطبيقات تتعامل معها بطريقة سهلة. أدناه سنوضح كيف يتم ذلك.

بناء خدمة ويب







انتهى !!!، فقط ثلاث نقرات بالماوس وتجد أمامك خدمة ويب كاملة جاهزة للاستخدام. لقد قام هنا فيجوال ستديو بإنشاء خدمة ويب اسمها Service وبها طريقة بعيدة اسمها HelloWorld تقوم بإرجاع القيمة النصية Hello World عند استدعائها. إذا أردت يمكنك إضافة طرق كما تشاء.

تجربة خدمة ويب

فقط وبالنقر على هذا الزر  يتم تجربة خدمة ويب أعلاه.



أنقر ok.



أنقر على HelloWorld.



أنقر على Invoke.

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">Hello World</string>
```

النتيجة في شكل XML.

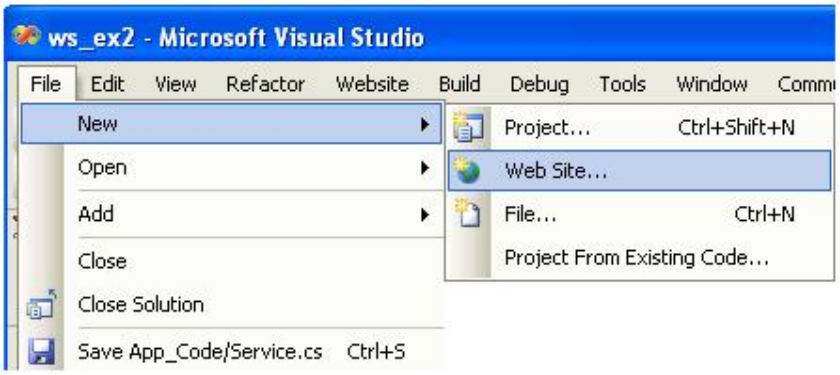
<u>إنشاء عميل</u>

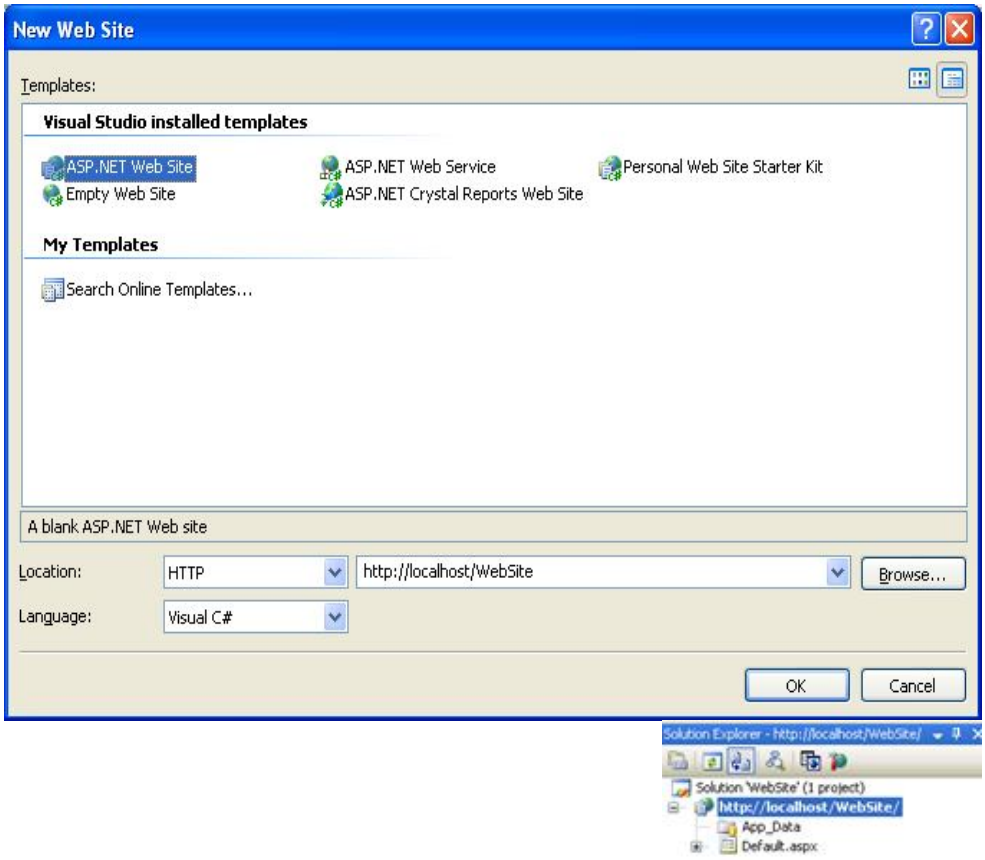
خطوات إنشاء العميل هي:

1. إنشاء الوكيل.
2. تنفيذ طرق الوكيل.
3. استلام البيانات الراجعة من خدمة الويب.
4. تصميم واجهة بالعميل لإظهار هذه البيانات.

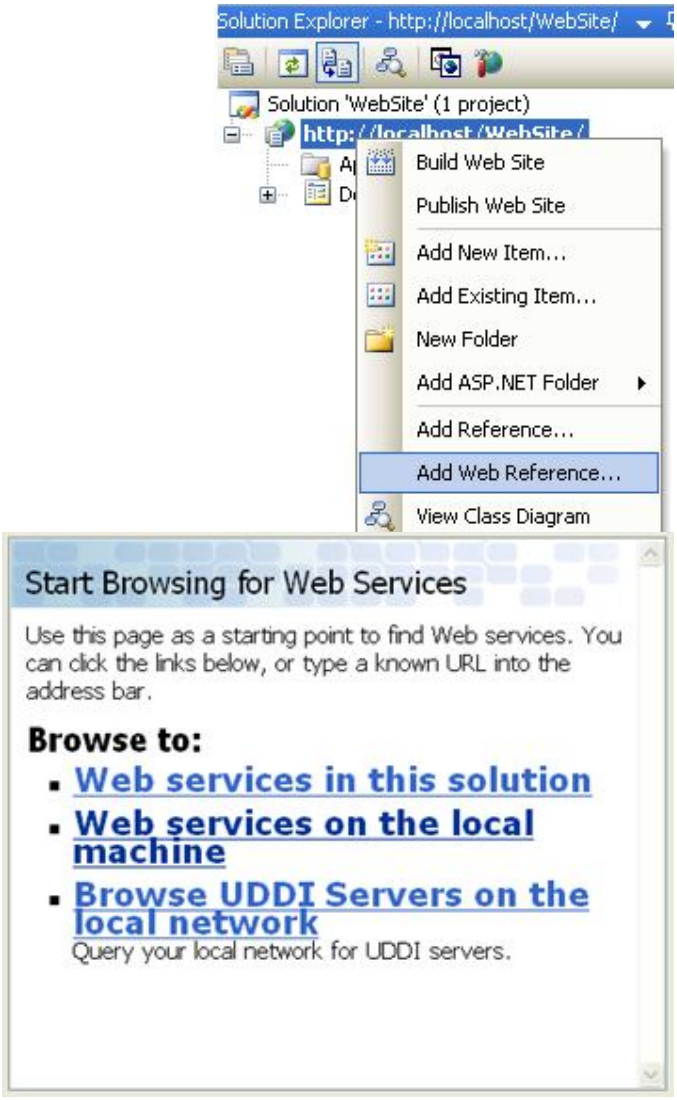
إنشاء عميل ويب لخدمة الويب service التي أنشأناها بواسطة فيجوال استديو.

نبدأ بإنشاء مشروع ويب بالخطوات التالية:




New Web Site
Templates:
Visual Studio installed templates
ASP.NET Web Site
ASP.NET Web Service
Personal Web Site Starter Kit
Empty Web Site
ASP.NET Crystal Reports Web Site
My Templates
Search Online Templates...
A blank ASP.NET Web site
Location:
HTTP
http://localhost/WebSite
Browse...
Language:
Visual C#
OK
Cancel

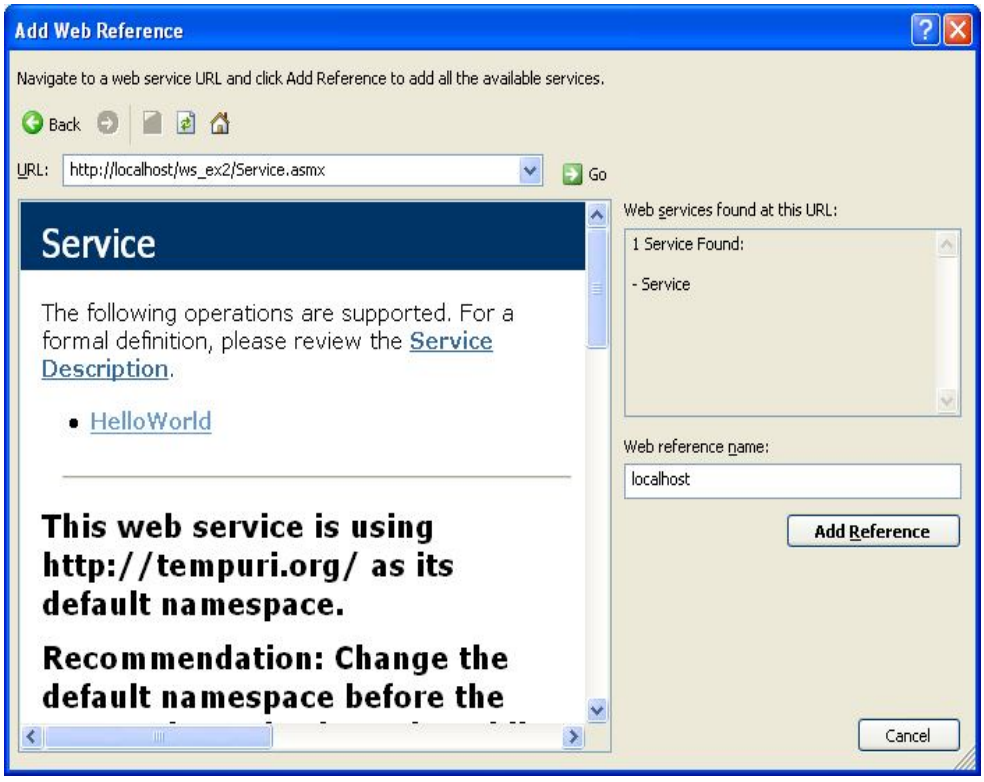
Solution Explorer - http://localhost/WebSite/
Solution 'WebSite' (1 project)
http://localhost/WebSite/
App_Data
Default.aspx



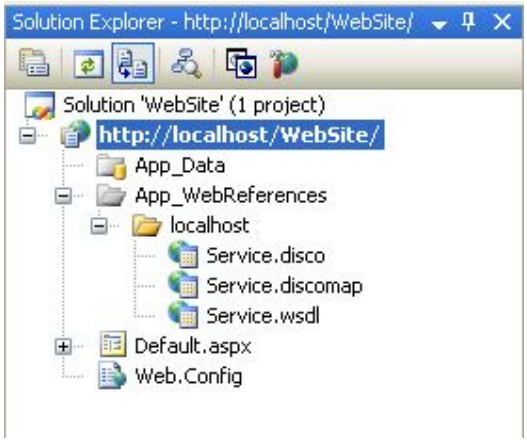
Web services on the local machine


Web Services on the Local Machine
The Web services and Discovery Documents available on your VS.NET developer machine are listed below. Click the service link to browse that service.
Services
URL
SecurityWebService
http://localhost/ds2/download4book
Service
http://localhost/ws_ex2/Service.asm
salam
http://localhost/ws_folder/salam.asr
ws_ex1
http://localhost/ws_folder/ws_ex1.a

Service

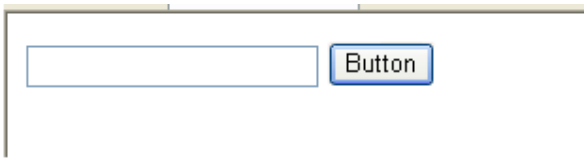


أنقر Add Reference






```
localhost.Service myS = new localhost.Service();
String s = myS.HelloWorld();
TextBox1.Text = s;
```



عند النقر على زر Button يتم استدعاء طريقة خدمة ويب لترجع لنا Hello World في مربع النص.



واجهة WAP (إنشاء نموذج ويب للهاتف المحمول (mobile_web_form))

بنفس الطريقة ولنفس الخدمة يمكنني إنشاء عميل يتصل بها من هاتف نقال باستخدام بروتوكول الواب (WAP).



كانت هنالك أداة تسمى Microsoft Mobile Internet Toolkit (MIT) تحتوي على تقنيات تسمح بالتعامل مع كم هائل من الهواتف النقالة. هذه الأداة أصبحت مبنية في نسخة فيجوال ستديو 2005 عبر نموذج يسمى mobile web form.

يمكنك إنشاء مشروع ويب ثم إضافة mobile web form به، فتظهر عندئذ أدوات في ToolBox تخص هذا النموذج.

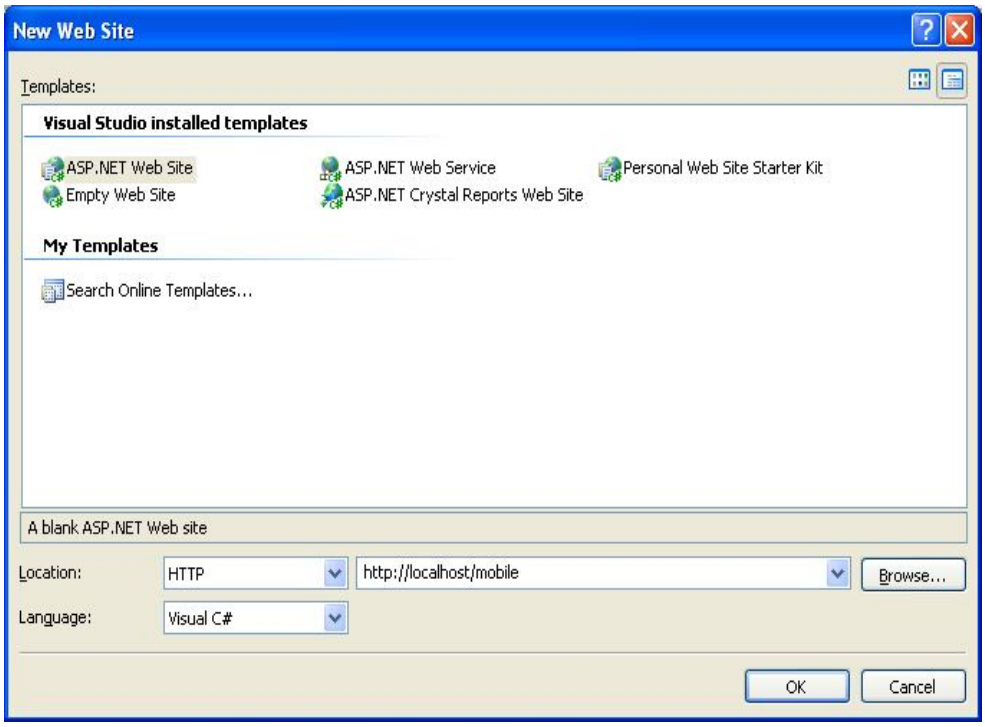
تلاحظ أنها تشبه كثيراً تلك المستخدمة في نموذج ويب العادي، وتستخدم بنفس الطريقة (سحب ووضع على النموذج). وهي توفر واجهة مرئية لبرمجة الهواتف النقالة، اذا لم تستخدمها ستضطر لتصميم واجهات الهواتف النقالة بلغة تشبه لغة HTML تسمى WML وهي بسيطة ولكنها تصبح صعبة عند بناء حلول كبيرة (تكتب اوامر WMP على محرر نصوص وتستخدم متصفح هاتف محمول مثل Nokia Toolkit browser لبناء تطبيقات واب). أما بواسطة mobile web form فيمكنك تصميم تطبيقات الهواتف المحمولة بطريقة مشابهة لتصميم واجهات ويندوز أو ويب العادية.

أيضا يقلل نموذج ويب للهاتف المحمول العمل المطلوب للتعامل مع متصفحات مختلفة لهواتف محمولة مختلفة وذلك تصميم تطبيق ويب واحد للتعامل مع الأجهزة المحمولة المختلفة.

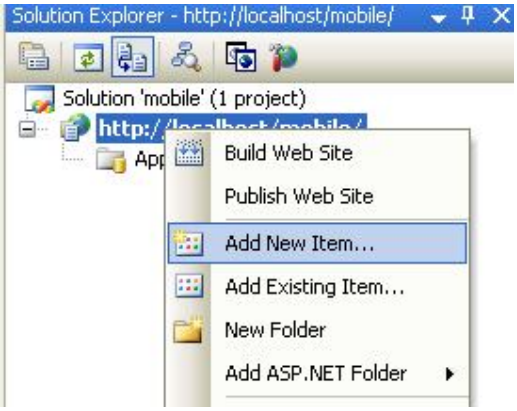
أدوات تحكم mobile web form تنشئ أكواد لغات WML, HTML, XHTML تلقائياً وتتعامل مع شاشات مختلفة المقاس واجهزة متعددة القدرات. حيث يتم إنشاء عناصر التحكم بلغة Markup للجهاز الذي يطلب الخدمة في زمن التنفيذ ، وهذه هي الخاصية التي تمكن نموذج تطبيق ويب من التعامل مع الأجهزة المختلفة.

لنبدأ إنشاء صفحة ويب للهاتف المحمول ثم نجعلها تتصل مع خدمة الويب البعيدة وتنفذ دوال بعيدة، أي التحكم بالنظام الموزع من داخل هاتفك المحمول وأنت في العمل أو في الطريق أو في البيت تتكي على أريكتك.

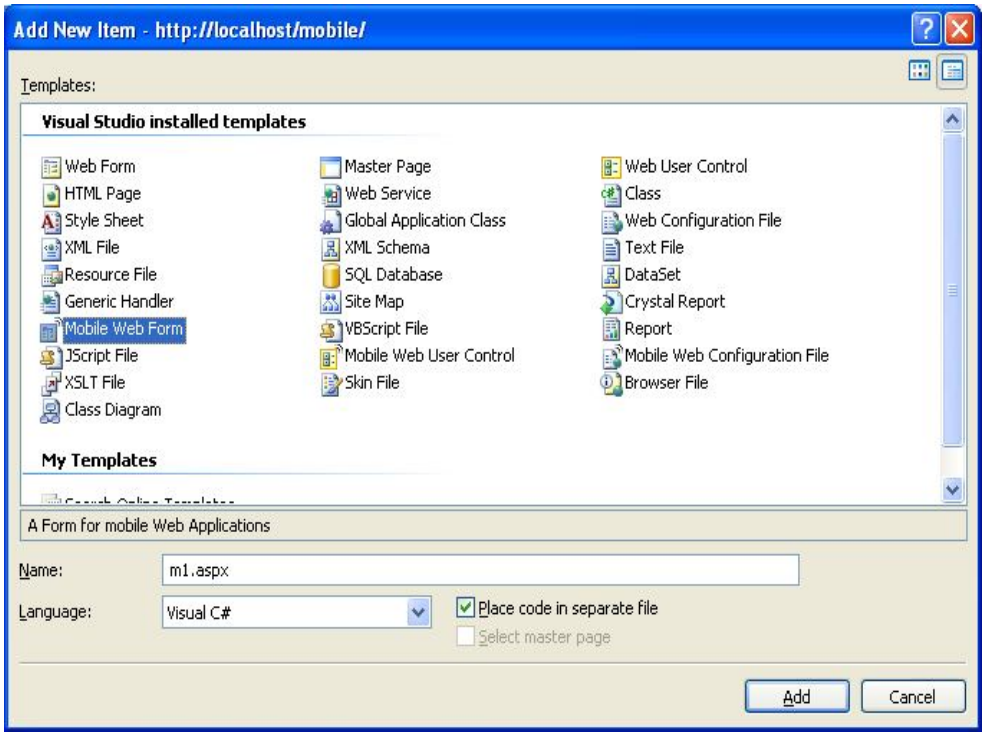
الخطوة الأولى هي إنشاء تطبيق ويب عادي (ASP.NET Web Site).



احذف نموذج ويب الإفتراضي default.aspx ثم أنقر يمين الماوس على اسم المشروع ثم أختر Add New Item.

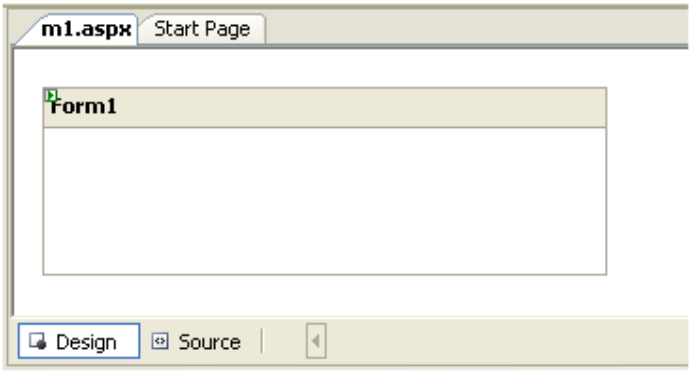


أختار Mobile Web Form.

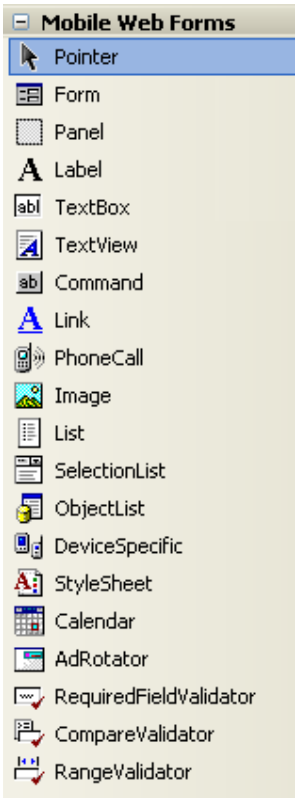


أنقر على Add.

أختر design من أسفل النموذج.

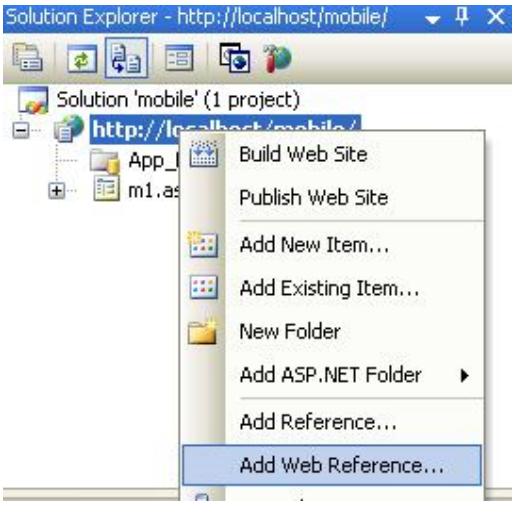


أنقر على Toolbox يظهر مربع الأدوات.



اسحب ما تريد داخل نموذج الهاتف المحمول (هنا قمنا بسحب Label ثم Command).

ثم أنقر يمين الماوس على اسم المشروع وأختر Add Web Reference



من الشكل أدناه أختر Web Services on the local machine، ذلك لأن خدمة الويب التي أنشأناها سابقاً كانت في الجهاز المحلي هنا.



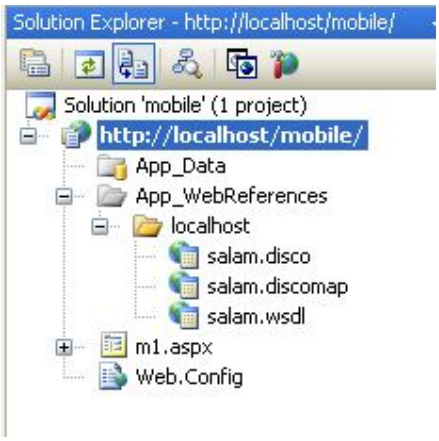
تظهر كل خدمات الويب المنشأة في هذا الجهاز ، سنختار منها Salam.



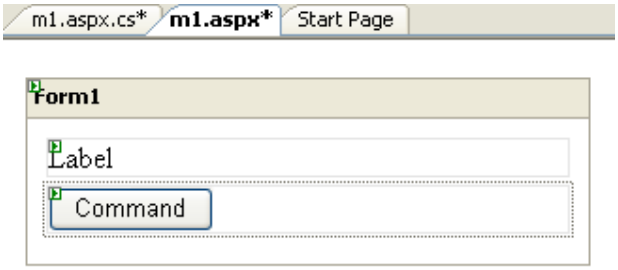
تظهر دوال Salam التي يمكن استدعاؤها من بعيد وهي (add و saySalam).



أنقر على Add Reference لإضافة وكيل للخدمة داخل مشروعي. حيث يظهر في المشروع وهو مكون من ثلاثة ملفات (salam.disco, salam.dicomap, slam.wsdl).



الآن أنقر نقرا مزدوجا على زر command لكتابة كود ينفذ عند النقر على هذا الزر.



نكتب الكود التالي:

```
localhost.Salam sa = new localhost.Salam();
int i = sa.add(5,7);
String s = sa.SaySalam();
Label1.Text = "return value from add ="+s +"and from saySalam= "+
i.ToString();
```

حيث يمثل السطر الأول إنشاء الكائن sa من الوكيل، ثم نستخدم هذا الكائن لاستدعاء دوال خدمة الويب البعيدة بالأوامر sa.add() و sa.SaySalam(). ثم نظهر النتيجة الراجعة من الدالتين على Label1.

عند تشغيل البرنامج تظهر الشاشة التالية:



وبالنقر على زر command ينفذ الكود أعلاه وتظهر النتيجة.

أسئلة تقويم ذاتي

1. ماهو البروتوكول المستخدم في خدمة الويب (web service) هل هو :
   a. HTTP
   b. SOAP
   c. SOAP & HTTP
   d. غير ذلك

تدريب (1)

في شكل جدول بين معاني الاختصار السابقة المستخدمة في خدمة الانترنت بين أهم استخدامات الاختصار ات السابقة.

الخلاصة

عزيزي الدارس، نلخص أهم النقاط التي وردت في هذه الوحدة ومن أهمها أنّ خدمات الويب (web services) تعتبر نموذجاً لطبقة وسيطة (middleware)، تسمح ببناء تطبيقات موزعة في الإنترانت والإنترنت. حيث توفر بيئة سهلة لبناء التطبيقات ومن خلالها يستطيع المبرمج بناء تطبيقات موزعة تتعامل مع بعضها البعض حتى ولو كانت في منصات متباينة. وعرفنا أن صفات خدمات الويب في سهلولة الإنشاء، وتوفر خدمات لتطبيقات أخرى وأنها ليس لها واجهة مستخدم (user interface) وإنما لها واجهة للبرامج

وبعد انتشار الويب وتطوره أصبح كل شيء مرتبطاً به. هذا التطور بدأ بصفحات html ساكنة تعرض محتواها للمستخدم بصورة ثابتة. ثم تطور وتوسع إلى أن وصل إلى ما نحن عليه. هذا التطور كان متمرحلاً على ثلاثة أجيال هي:جيل الصفحات الساكنة (static pages)، جيل الصفحات الديناميكية (dynamic pages)، جيل خدمات الويب (web services).

إجابات التدريبات

1/ اختصارات متعلقة بخدمة الويب

الرمز	الاختصار
UDDI	Universal Description, Discovery and Integration
WDSL	Web Services Description Language
SOAP	.Simple Object Access Protocol
XML	Extensible Markup Language.
HTTP	Hypertext Transfer Protocol

اختصارات خدمات الويب وخدماتها

الرمز	اهم الاستخدامات
UDDI	تخزن به معلومات عن خدمات الويب (معرف موقع الخدمة بالإنترنت). ليستطيع العميل interfacesتخزن به واجهات خدمات الويب
WDSL	تستخدم لمعرفة مكان خدمة الويب. تستخدم لمعرفة نوع الخدمات التي توفرها خدمة الويب وكيفية استدعائها.
SOAP	يستخدم للاتصال بين التطبيقات. يمثل صيغة لإرسال الرسائل
XML	تستخدم لتبادل المعلومات بين بيئات مختلفة (different platforms). تستخدم لتبادل المعلومات بين لغات مختلفة.
HTTP	يستخدم لنقل المعلومات في الويب (www). يستخدم لنشر وعرض الصفحات المكتوبة بلغة HTML.

مسرد المصطلحات

الصفحات الساكنة (static pages).

بدأت الويب بصفحات ثابتة المحتوى (ساكنة) ، فمحتواها واحد لكل المستخدمين و يظهر بنفس الشكل وفي كل الأوقات، تسمى هذه الصفحات صفحات ساكنة (static pages). طريقة عملها بسيطة جداً ، وهذا سبب انتشارها

الصفحات الديناميكية (dynamic pages).

هذه الصفحات تظهر محتواها غالباً من قاعدة بيانات. فإذا أردت تعديل صفحة فلن أحتاج إلى فتح كل ملفات html وإنما أعدل في محتوى قاعدة البيانات فيتغير محتوى الصفحات كلها تلقائياً ، وبالتالي لن أحتاج لإعادة رفع الصفحات بمخدم الويب مرة ثانيةعيوب الصفحات الساكنة أن تعيلها صعب ويستغرق وقتاً كبيراً .

خدمات الويب (web services).

هي عبارة عن برامج مكتبية، تخزن بمخدم ويب وتكون متاحة للكل، ولاستخدامها لابد من معرفة مكانها وأسماء الدوال التي بها و نوع الخدمات التي تقدمها هذه الدوال. حيث نرسل طلب باسم الدالة التي نريد تنفيذها (Request)، فتستجيب له خدمة الويب وتنفذه بالجهاز البعيد ثم ترسل الرد على الطلب بنتائج تنفيذ هذه الدالة (response).

المراجع

1- 1.Coulouris, G. et al., (2004), Distributed Systems, Concepts and Design, Addison Wesley

2- Farley, J. (1998), Java Distributed Computing, O'reilly & Associates

3- Fedorov, A, (2002), A Programmer Guide to .NET, Addison Wesley

4- Ince, D. (2002), Developing Distributed and E-Commerce Applications, Pearson Education

5- Liu, M. (2004), Distributed Computing Principles and Applications, Pearson Education

6- Schonauer, W. (1998), Scientific Supercomputing: Architecture and use of Shared and Distributed Memory Parallel Computers, Willi Schonauer, Germany

7- Tanenbaum, A., (2002), The Amoeba Distributed Operating System, Vrije Universiteit Technical Report, URL ftp://ftp.cs.vu.nl/pub/amoeba/Intro.ps.Z

8- Tanenbaum , A. AND Van Steen, M. (2002), Distributed Systems Principles and Paradigms, Prentice Hall

9- Tanenbaum, A. (1995), Distributed Operating Systems, 1/e, Prentice Hall, USA

10- Osman, A., (may 2006), A Multi-level Web-based Parallel Processing System, TRANSACTIONS ON ENGINEERING, COMPUTING AND TECHNOLOGY VOLUME 13 MAY 2006 ISSN 1305-5313

11-Forouzan, A. (2007), Data Communications and Networking, Mc Graw Hill, Fourth Edition

12-Tanenbaum, A., (2001), Modern Operating Systems, Prentice Hall, Second Edition

13-Tanenbaum, A., (1999), Structred Computer Organization, Prentice Hall, Fourth Edition

{الملاحق}

الملحق (أ)

المرجع المختصر في الجافا

مفهوم اللغة كائنية التوجه

تعتبر جافا لغة كائنية التوجه (Object oriented programming) حيث يكون عنصرها الأساسي البرمجي هو الكائن (object). الكائن كل شئ يمكن وصفه والتعامل معه (له صفات وله سلوك). فاكل شئفي الكون حولنا كائن سوى كان جماداً أو حيواناً أو نباتاً .

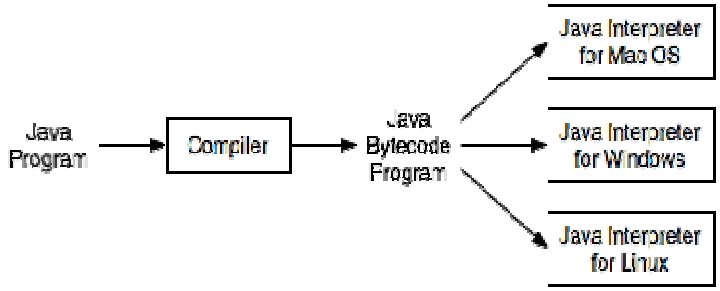
فكرة إدخال الكائنات في البرمجة هي ببساطة سهلة جداً، فما عليك سوى التعبير عن سلوك الكائن بوحدات برمجية (دالة لكل سلوك) تسمى الطرق (methods) والتعبير عن صفات الكائن بمتغيرات. نجمع الدوال والمتغيرات لكل كائن في وحدة برمجية تسمى الفئة (class).

هنالك فوائد كثيرة من فكرة الكائنات هذه (اللغة كائنية التوجه) مثل:

- إعادة استخدام الكود Code re-use
- التغليف Encapsulation
- الوراثة Inheritance
- تعدد الاشكال Polymorphism

آلة جافا الإفتراضية ((Java Virtual Machine (JVM)

لا يستطيع الحاسب فهم او تنفيذ لغة جافا مباشرة، فلا بد من تحويلها إلى لغة آلة تسمى جافا بايت كود (java bytecode) و ملفاتها تكون بالامتداد class. جافا بايت كود تنفذ بواسطة مفسر جافا يسمى آلة جافا الإفتراضية (java virtual machine (JVM، كما في الشكل ادناه.



نقوم أولاً بكتابة كود جافا المصدري (.java) ثم نترجمه بواسطة مترجم جافا (compiler) فينتج عن ذلك ملف بلغة الآلة الخاصة بجافا (java bytecode)، هذا الملف يكون امتداده class، يمكن تشغيل هذا الملف (.class) في أي بيئة بأي نظام تشغيل بشرط واحد هو "وجود مفسر جافا (JVM)" بالجهاز الذي نريد تشغيله فيه.

عن لغة جافا

- ☒ عملتها شركة صن ميكروسيستم
- ☒ لغة كائنية التوجه
- ☒ مطورة من C/C++
- ☒ يمكن استخدامها لتصميم تطبيقات الويب والانترنت
- ☒ واسعة الانتشار
- ☒ بسيطة Simple
- ☒ متنقلة Portable
- ☒ الاعتمادية – قابلية الاستمرار Reliable (ففيها الإستثناءات)
- ☒ السرية والحماية Secure
- ☒ إمكانية تنفيذ أكثر من برنامج في وقت واحد Multithreaded

عيوب الجافا

بطيئة أكثر من اللغات المترجمة مثل C

اجريت تجربة في 1999 وجد فيها أن جافا ابطأ حوالي ثلاث إلى أربع مرات عن C++

title of the article: "Comparing Java vs. C/C++ Efficiency Issues to Interpersonal Issues" (Lutz Prechelt)

الفئة الرئيسية والدالة الرئيسية

ملفات جافا المصدرية لها الإمتداد java.

تنظم فئات وواجهات جافا في شكل حزم ويمكنك استخدام الفئات الموجودة بحزمة معينة بالأمر التالي:

import name_of_package.*;

الدالة الرئيسية (main)

كل برامج جافا تعتبر فئات ويمكن أن يكون هنالك العديدمن الفئات ببرنامج واحد ولكن دائما تكون لدينا فئة واحدة بالبرنامج تعتبر هي الرئيسية، هذه الفئة لابد من أن تحتوي على الدالة main، مثال:

```
public class myMainClass
 {
        public static void main(String[] args) {
                        أكتب اكوادك هنا
        إنشاء كائن من الفئة الثانية واستخدامها هنا
        }
        يمكن أن يكون لديك دوال أخرى هنا
}
Class MySecondClass
{اكواد الفئة الثانية}
```

تنبيه هام

لغة جافا حساسة تجاه الحروف الكبيرة والصغيرة.

أنواع البيانات

أنواع البيانات (Data types) :

1. يوجد 8 أنواع أساسية (8 primitive types) هي
   - boolean, byte, short, int, long, float, double, char
2. بقية الانواع مثل: الفئات سواء وفرت بواسطة الجافا ام قام بإنشائها المستخدم مثل
   - String, Integer, Array, Frame, Object, Person, Animal, …
3. المصفوفات : متغير يحتوي عدة قيم
4. المتغيرات Variables

التعليق comment : نوعين

تعليق في سطر واحد يبدأ بـ // و ينتهي بنهاية السطر

تعليق في اكثر من سطر يبدأ */ وينتهي بـ /*

العوامل (Operator)

- العوامل الرياضية : +، – ، * ، / ، %
- العوامل المنطقية : and (&&), or (||), not (!)
- عوامل المقارنات : =! ,== ,>= ,<,<=,>

أوامر التحكم

طبيعة البرنامج هي تنفيذ الاوامر تسلسليا واحد تلو الآخر ولكن يمكننا تغيير هذه الطبيعة باستخدام ادوات التحكم في تسلسل البرنامج (Flow of control) مثل:

- if, if-else, if-else if
- switch
- for, while, do-while
- Break
- continue

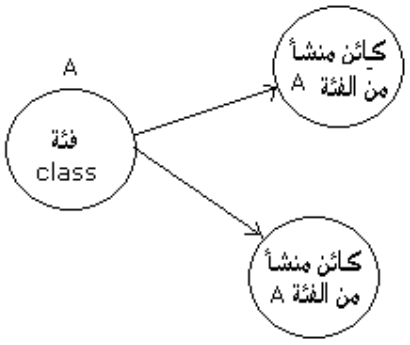
إنشاء الفئات والكائنات

ما هو الفرق بين الفئة class والكائن object ؟

1. الفئة تمثل نوع ، فعندما انشئ فئة فهذا يعني أنني أنشأت نوع جديد
2. يمكنني إنشاء متغير من هذا النوع الجديد يسمى الكائن
3. عند إنشاء أكثر من متغير (كائن) من الفئة الواحدة أكون قد حققت خاصية إعادة استخدام الكود (code reuse) ، أنظر الشكل أدناه.
4. تعرف الفئة بالامر class
5. يعرف الكائن عن طريق الامر new حيث نكتب:

class_name object_name = new class_name();

حيث يمثل class_name اسم الفئة ، object_name اسم الكائن الذي نريد تعريفه



خطوات إنشاء فئة واستخدامها

1. إنشاء الفئة (النوع).
2. إنشاء الكائنات التي تريد من الفئة (متغيرات).
3. إستخدام الكائنات داخل برنامجك.

كيفية إنشاء فئة:

```
Class classname
{
المتغيرات تمثل صفات الكائن
الدوال تمثل سلوك الكائن – ما يفعله الكائن
}
```

إنشاء الكائن

```
Classname objectname = new classname();
```

الوراثة

مثال (1)

إذا أردت إنشاء فئة خاصة من النوع Fruit سأحتاج لكل الدوال الموجودة في Fruit : أقوم بوراثة صفات الفئة Fruit بالأمر

```
class Citrus extends Fruit{
  void squeeze(){….}
```

حيث تمثل العبارة اعلاه : "الحمضيات (Citrus) هي نوع خاص من الفواكه (Fruit)"، أى انني سأحتاج لكل صفات الفواكه و بدلا من اعادة كتابتها اورثها بواسطة الامر extends ثم اضع داخل الفئة الجديدة Citrus الاصفات والسلوك التي توجد في الحمضيات (Citrus) و لاتوجد في الفواكه.

مثال (2)

```
class employee extends person {
        String employeeNum;
        double salary;
}
```

هنا نقول ان الموظف (employee) هو شخص (person) و لكن لديه رقم وظيفي ومرتب، فورثنا صفات الأشخاص واضفنا إليها المعلومات التي يتميز بها الموظف على الشخص العادي.

كيفية استخدام الفئة الوارثة؟

كل الدوال والمتغيرات التي تم تعريفها في الفئة الأب (التي ورثنا منها الصفات) يمكن استخدامها في الفئة الوارثة (الإبن)، كأنها موجودة بهذه الفئة.

الاستثناءات Exceptions

هنالك ثلاث انواع من الاخطاء قد تحدث:

1. خطأ لغوي (syntax error) وهذا غالبا ما يحدث في زمن الترجمة (compile time error).
2. خطأ قد يحدث في زمن التنفيذ (run time error).
3. خطأ منطقي.

الاستثناء في الجافا هو كائن يوضح الاخطاء التي تحدث في زمن التنفيذ في برامج الجافا.

معالجة الاستثناء Exception Handling

قد تحدث اخطاء في زمن التنفيذ يمكن اصطيادها بواسطة الامر catch.

اذا لم نستخدم الامر catch فيصل الخطأ لمعالج الاخطاء الافتراضي الموجود بالجافا.

الشكل العام لكيفية كتابة كود يصطاد اخطاء زمن التنفيذ كالآتي:

```
try {
  statements . . .
} catch (ExceptionType1 ename1) {
  error handling statements . . .
} catch (ExceptionType2 ename2) {
  error handling statements . . .
} finally {
  … this code always executed …
}
```

- نستخدم الأمر try لمحاولة تنفيذ اوامر متوقع حدوث أخطاء فيها
- نستخدم فقرات catch، كل واحدة يمكنها اصطياد إستثناء معين
- نستخدم الأمر finally الذي ينفذ دائما في النهاية

بريمجات جافا (Java Applets)

صممت بريمجات الجافا لتكون:

1. سريعة
2. صغيرة
3. بالتالي من السهل نقلها عبر الشبكة

بريمجات جافا ليست مثل برامج جافا التي درسناها مسبقا فهي ممنوعة من الوصول إلى:

- موارد الشبكات
- موارد الجهاز
- إلا في حالات نادرة جدا. بالتالي هذا يجعل تحميل البريمج من الانترنت إلى جهازك وانت تتصفح الانترنت آمنا ، لانها لا تستطيع الوصول إلى ملفاتك ومواردك.

هنالك ثلاث عناصر أساسية أحتاجها لعمل بريمج:

1. صفحة فائقة مكتوبة بلغة html تحتوي على وسم <applet> الذي يستخدم لإدراج بريمج بالصفحة.
2. بريمج جافا المصدر (.java)
3. بريمج جافا بلغة آلة جافا (javabytecode) وهو بالإمتداد (.class) وهذا ينتج من ترجمة بريمج جافا المصدري في 2 أعلاه.

<u>مثال</u>

1. الصفحة الفائقة:

```
<HTML>
    <HEAD>
        <TITLE>Example</TITLE>
    </HEAD>
    <BODY>
        <APPLET Code = "salam.class"
                Width = "400"
                Height = "200">
        </APPLET>
    </BODY>
```

```
</HTML>
```

2. بريمج جافا المصدري:

```
import java.applet.Applet;
import java.awt.Graphics;
        public class salam extends Applet {
                public void paint(Graphics g){
                        g.drawString("Aslamu Alikum…",25,25);
                }
}
```

3. نترجم البريمج أعلاه بوساطة مترجم جافا ونضع الملف المترجم (.class) في المكان الذي توجد به الصفحة الفائقة في 1 أعلاه، بنقرتين مزدوجتين على الصفحة الفائة تتنفذ بالمتصفح ويتنفذ معها البريمج.

المحلق (ب)

برنامج Eclipse مفتوح المصدر

يعتبر برنامج eclipse برامج لتطوير تطبيقات جافا (يحتاج تنصيب SDK بجهازك وتكون نسخة 5 أو أعلى) ، حيث يتميز هذا البرنامج بأنه:

- مجاني.
- مفتوح المصدر.
- يعمل على بيئات مختلفة (يندوز، لينكس، وغيرها).
- يعمل مع لغات برمجة عديدة مثل C/C++, PHP.
- به IDE غنية بكل ما تحتاج.

يمكنك تنزيل النسخة التي توافقك من موقع البرنامج الرسمي وهو
http://www.eclipse.org .

الجهة المسئولة من هذا البرنامج منظمة غير ربحية وهذا هو الجيد في الأمر، فمطوري هذا البرنامج هو متطوعين من مبرمجي المصادر المفتوحة والإتحادات والمنظات الطوعية ويمكن أن تكون أنت منهم.

كيفية استخدام eclipse

اختار أحد حزم البرنامج من الموقع المذكور أعلاه وذلك بتحميلها في جهازك:

إذا كنت مطور جافا وتحتاج IDE للجافا اختار التحميل من الموقع التالي :
http://www.eclipse.org/downloads/moreinfo/java.php

إذا كنت مطور جافا وتنشئ تطبيقات EE يمكنك تحميل النسخة الموجودة بهذا الموقع
http://www.eclipse.org/downloads/moreinfo/jee.php

إذا كنت مطور C/C++ وتحتاج IDE لها فعليك بالموقع التالي
http://www.eclipse.org/downloads/moreinfo/c.php

أما اذا أردت بناء Eclipse plugins أو تطبيقات RCP فيمكنك تحميل نسخة من هذا الموقع

http://www.eclipse.org/downloads/moreinfo/rcp.php

برنامج Eclipse لا يحتاج إلى اعداد، فبعد تحميله من الموقع قم بك الضغط، ستجد ملف تنفيذي بالنقر عليه يعمل Eclipse وتبدأ في استخدامه مباشرة. هذه الطريقة تسري على نسخة ويندوز ونسخة لينكس على حدٍ سواء.

تركيب RMI في برنامج Eclipse

عند تحميل Eclipse لا تتحمل معها طبقة RMI بالتالي لا نستطيع استخدامها لبناء تطبيقات موزعة على RMI. لكن من السهل تحديث برنامج eclipse بتركيب طبقة RMI به ومن ثم نستخدمه (version 2.0 RMI Plug-in for Eclipse). هنالك طريقتين لتحديث eclipse لإضافة طبقة RMI هما:

تنزيل ملفات RMI المضغوطة من الموقع

http://www.genady.net/rmi/index.html

حسب نسخة Eclipse التي تستخدم:

Eclipse 3.3 (Europa): **net.genady.rmi_2.0.7_Eclipse33.zip**
Eclipse 3.2 (Callisto) and 3.1: **net.genady.rmi_2.0.6.zip**

أو اختيار الأمر update من داخل برنامج Eclipse فيركب RMI المناسبة تلقائياً (وهذه الأسهل والأفضل).

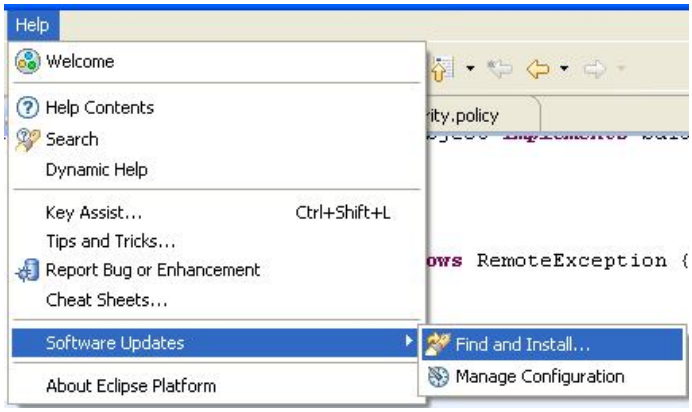
التحديث يتم من المواقع التالية:

Eclipse 3.3 (Europa) **http://www.genady.net/rmi/v20/install_e33**
Eclipse 3.2 (Callisto) and 3.1: **http://www.genady.net/rmi/v20/install**

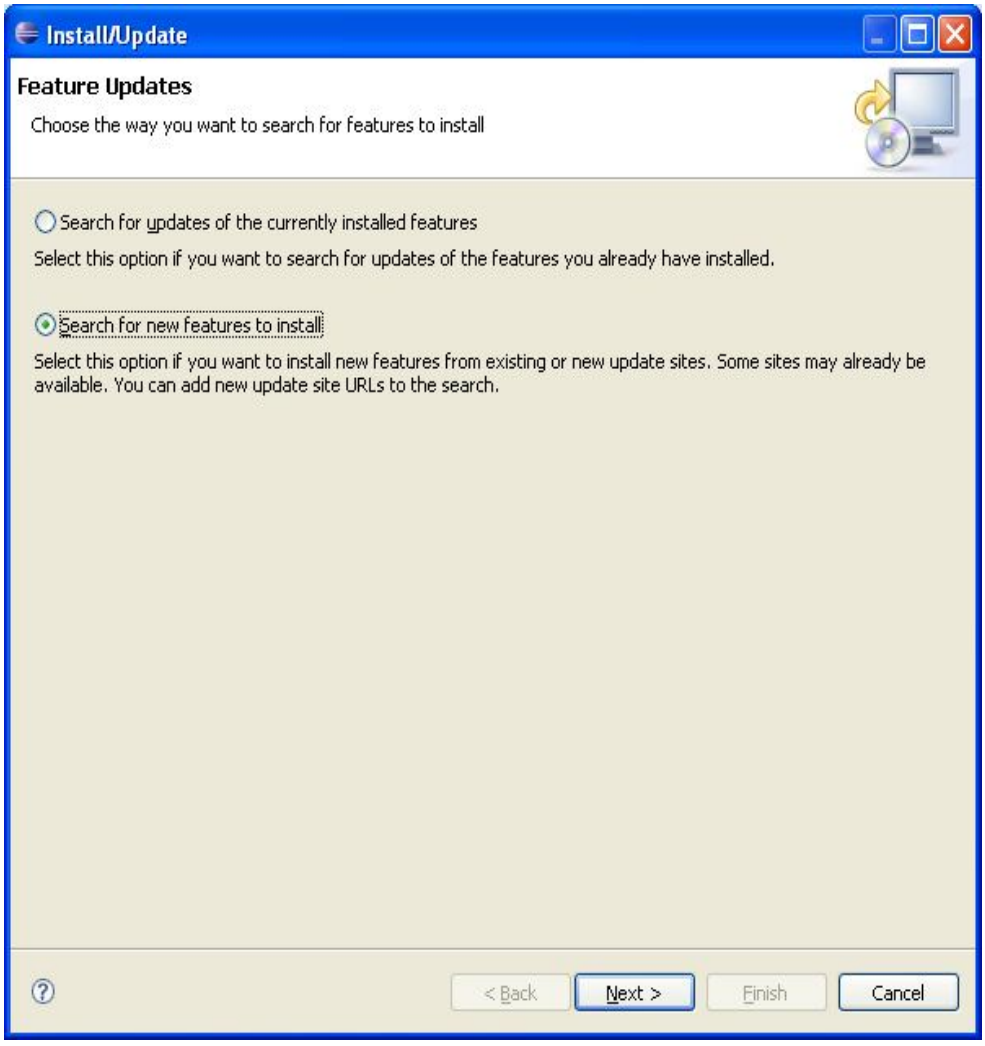
استخدام الطريقة الثانية لإضافة RMI على eclipse

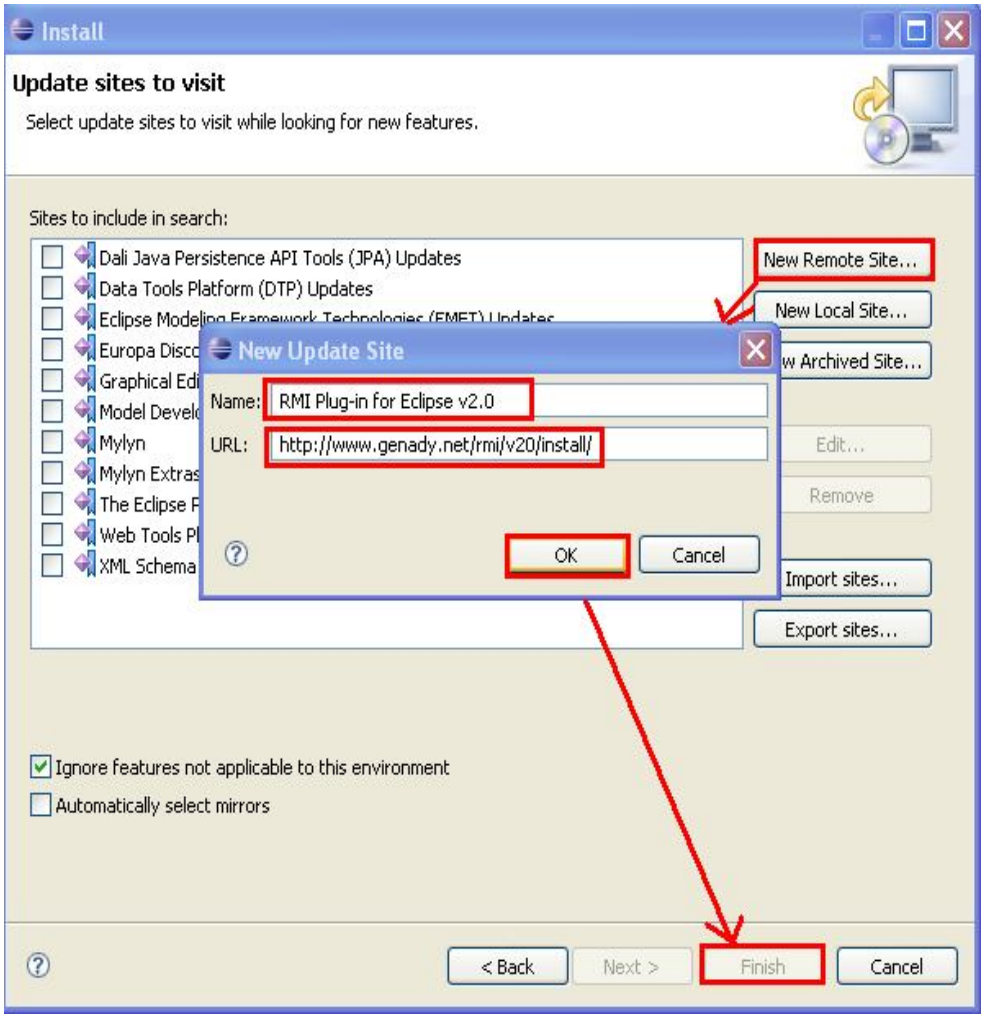
خطوات update هي كالآتي:

اختار "Help -> Software Updates -> Find and Install..."



أدناه اختار Search for new features to install ثم اختار Next


Install/Update
Feature Updates
Choose the way you want to search for features to install
Search for updates of the currently installed features
Select this option if you want to search for updates of the features you already have installed.
Search for new features to install
Select this option if you want to install new features from existing or new update sites. Some sites may already be available. You can add new update site URLs to the search.
< Back
Next >
Finish
Cancel

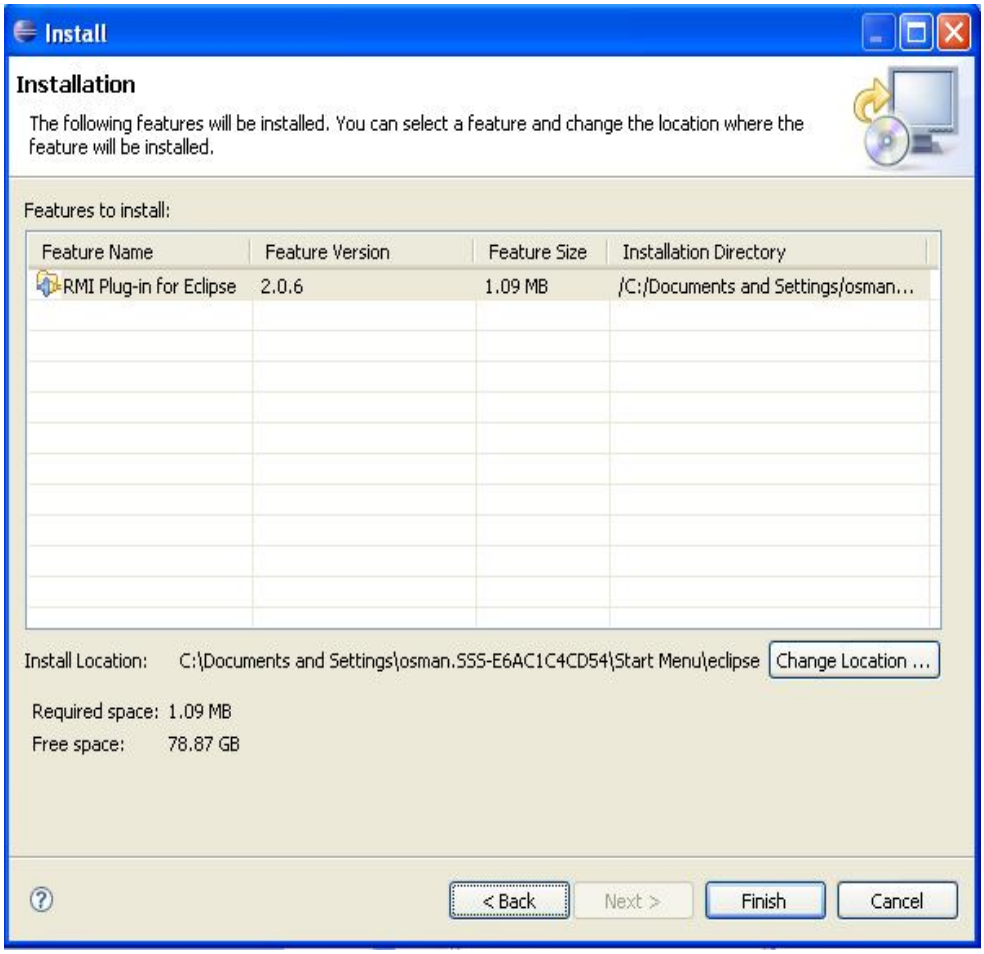


في الصفحة التالية اختار new remote site وادخل المعلومات التلية:

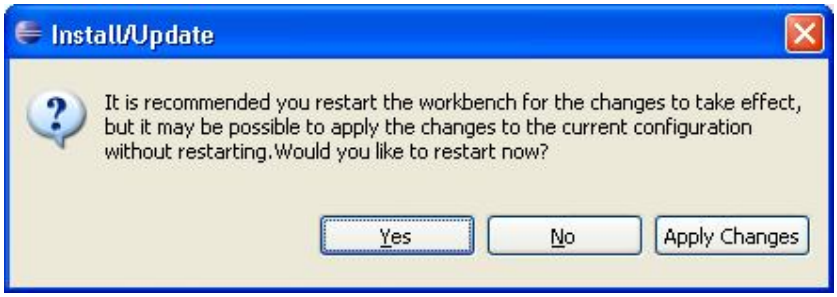
Name: RMI Plug-in for Eclipse v2.0
URL: http://www.genady.net/rmi/v20/install/

أنقر OK ثم اضغط "Finish" للاستمرار.

سيتم البحث عن أدث إصدار وإضافته لبرنامج Eclipse



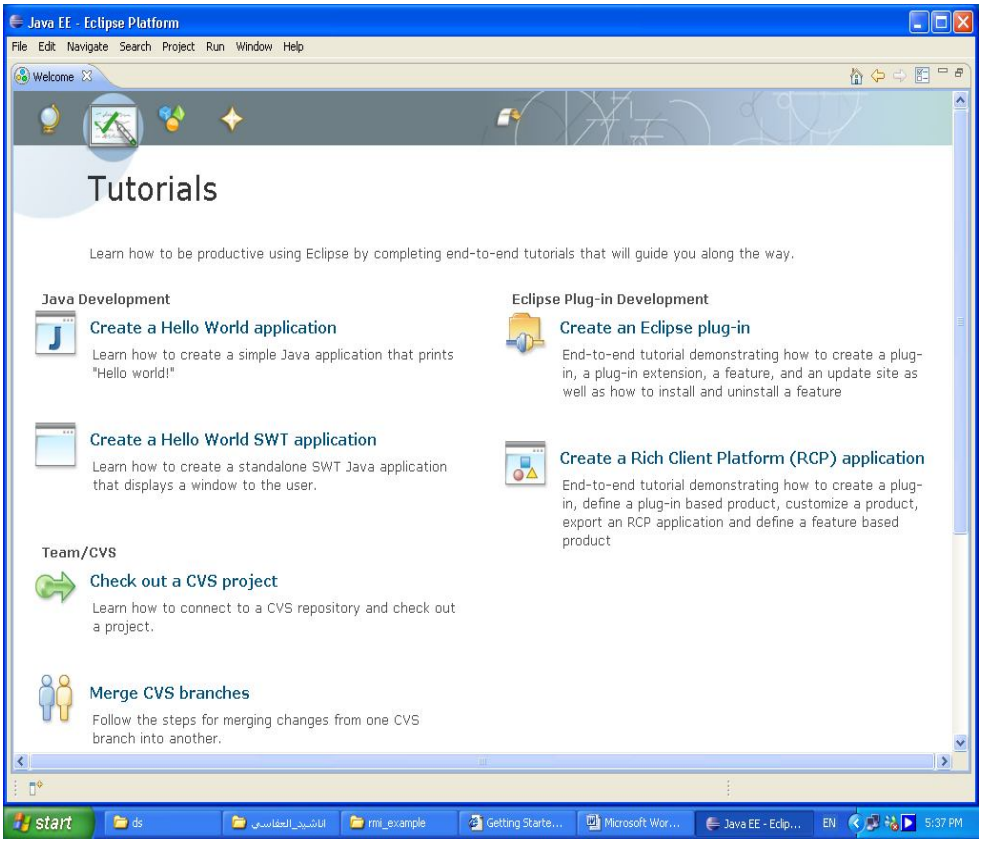
ثم إذا سؤلت عن اعادة تشغيل Eclipse انقر على Yes



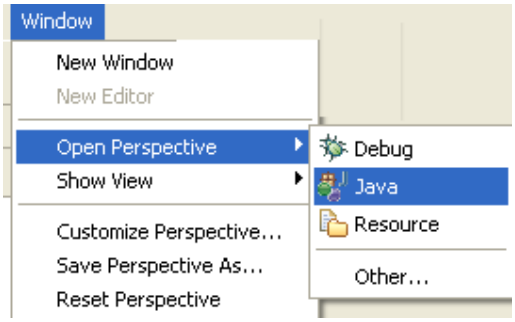
قد تحتاج إلى إعادة تشغيل perspective داخل eclipse وذلك من -> windows Reset Perspective ليظهر زر RMI Registry Control.

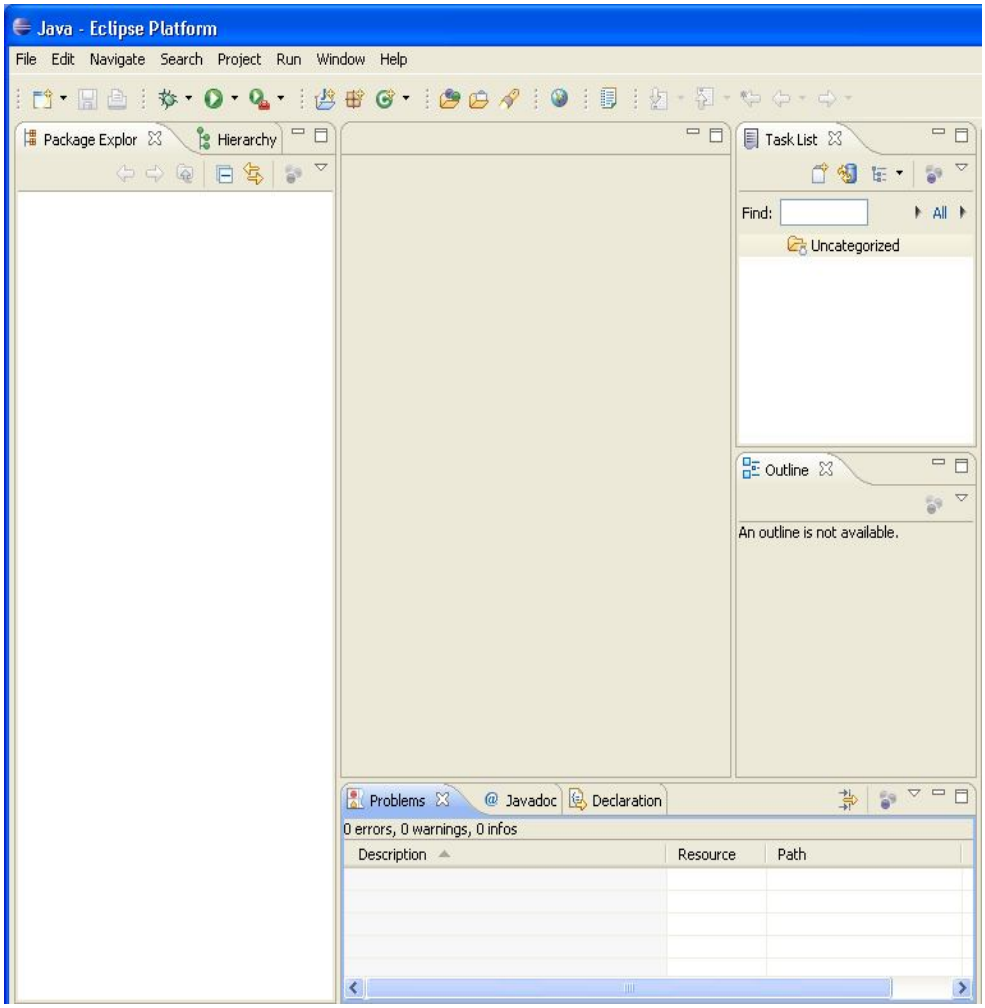
استخدام برنامج Eclipse

قم بفتح برنامج Eclipse



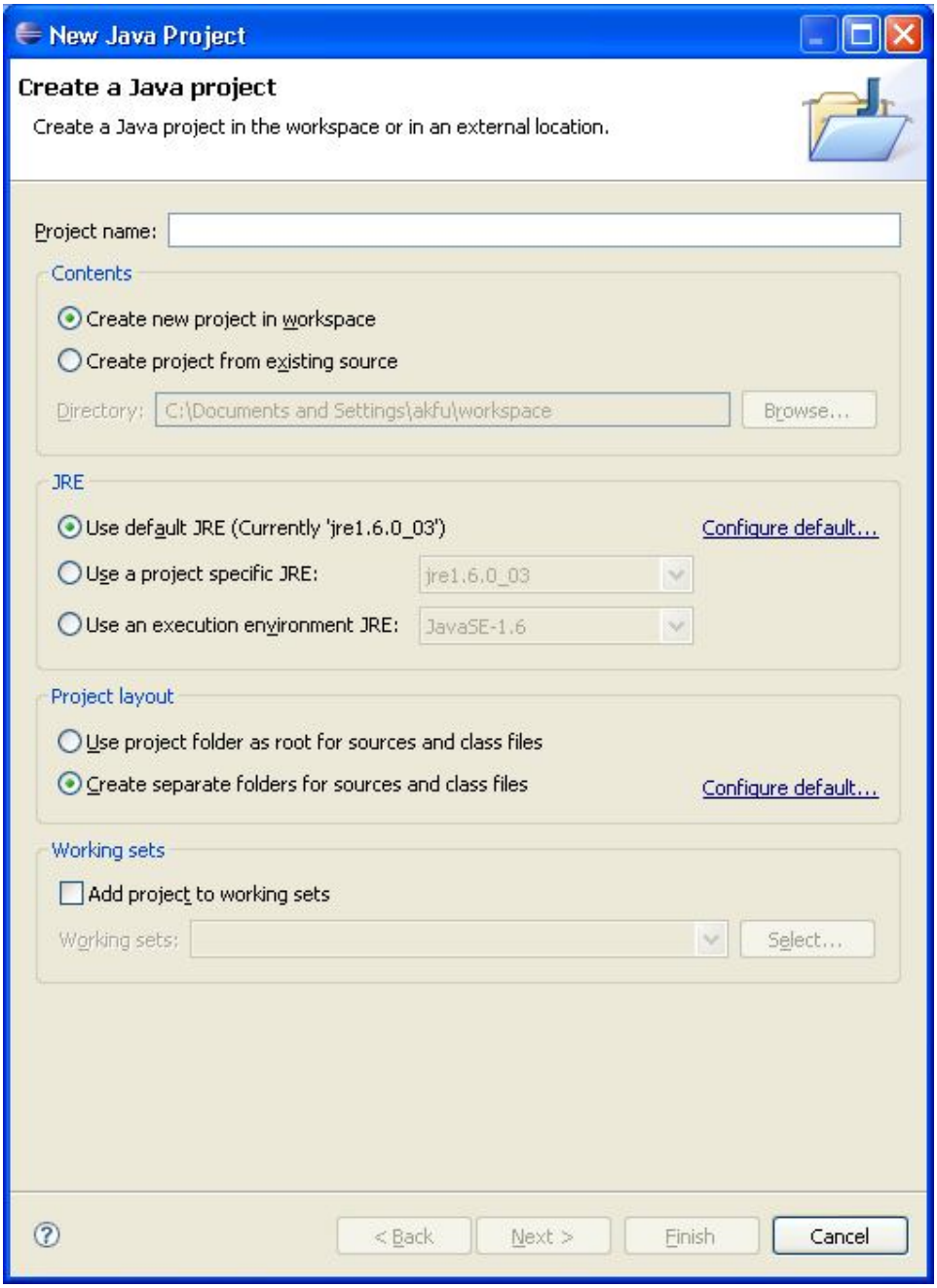
إنشاء تطبيق جافا





إنشاء java project



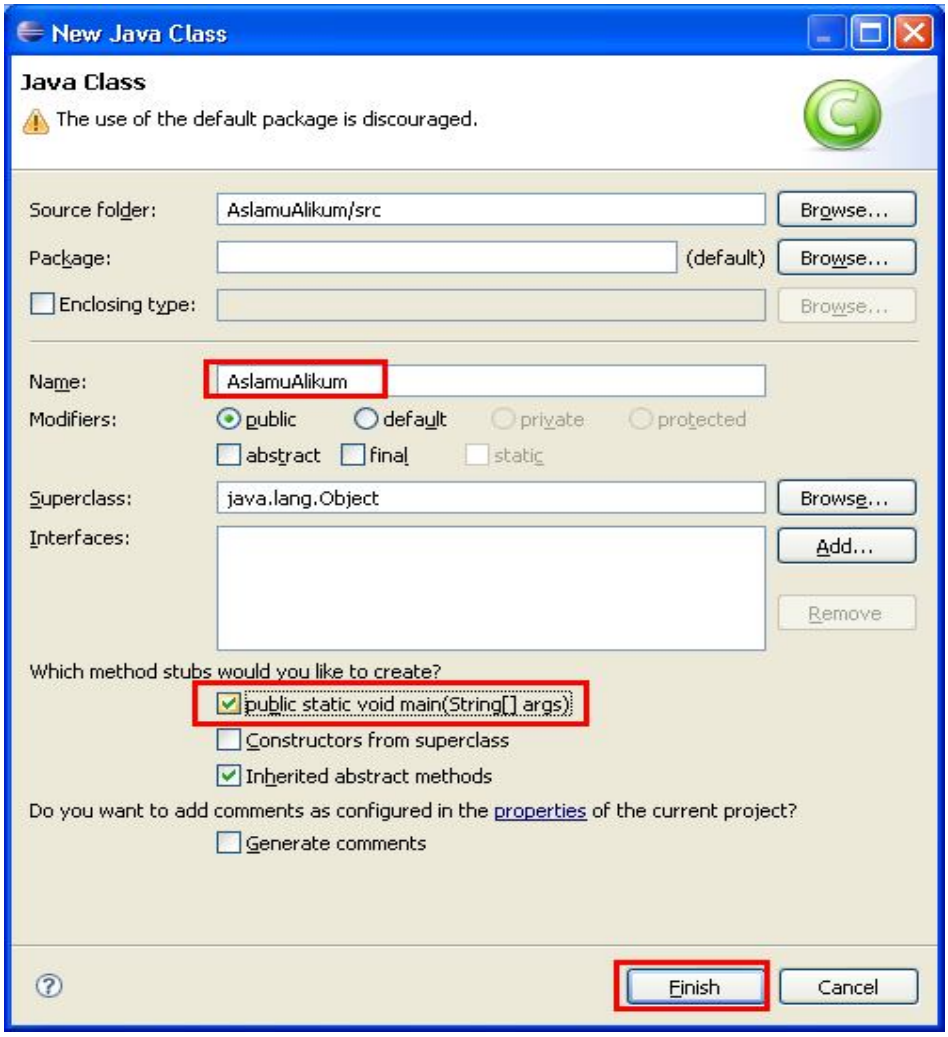


اكتب اسم المشروع في project name ثم أنقر finish

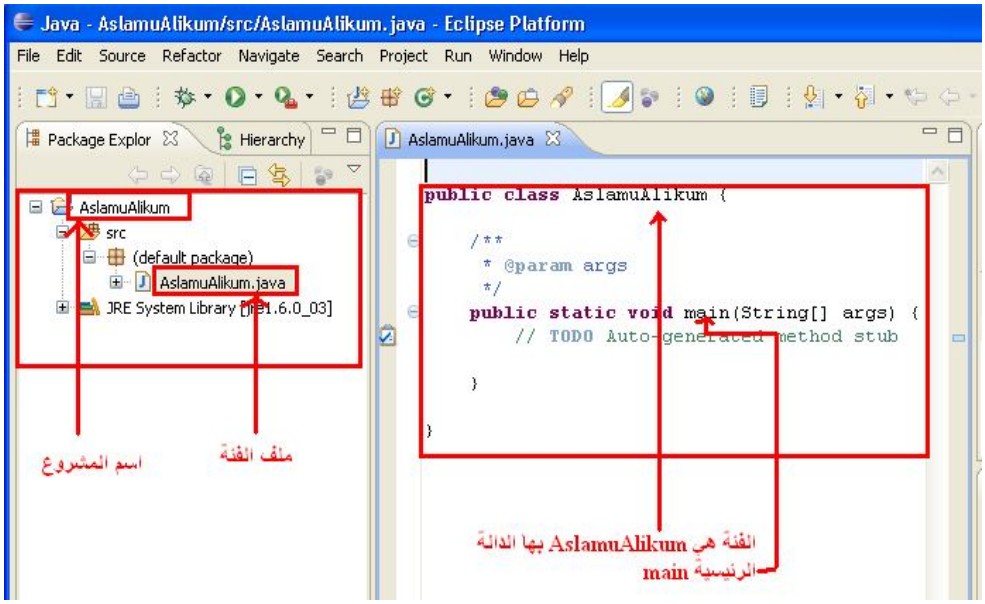
ننشئ فئة داخل المشروع بالأمر file-new-class



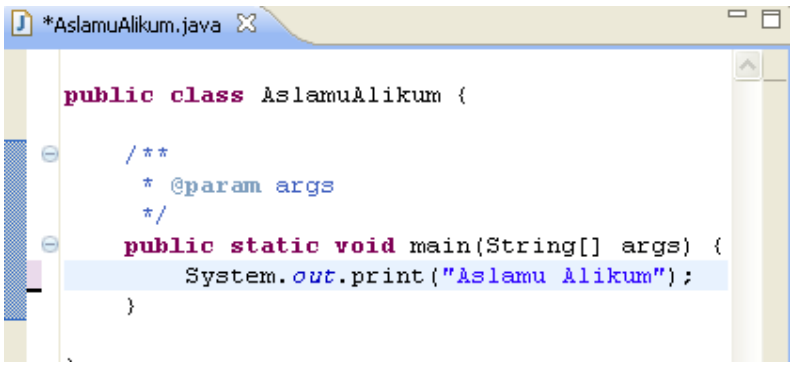
اكتب اسم الفئة (AslamuAlikum) ثم انقر على main ثم finish كما بالشكل أدناه:



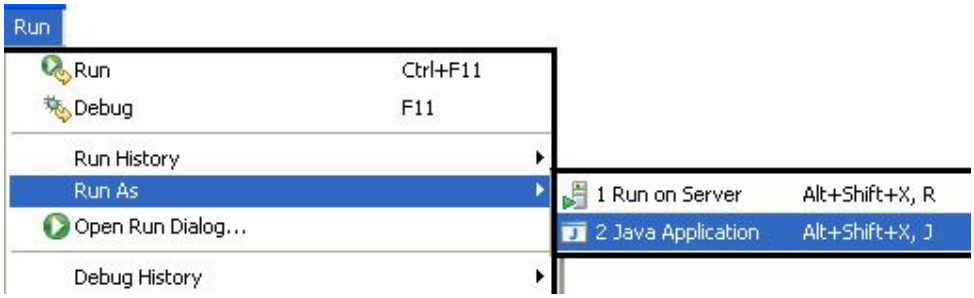
تظهر الفئة ادناه:



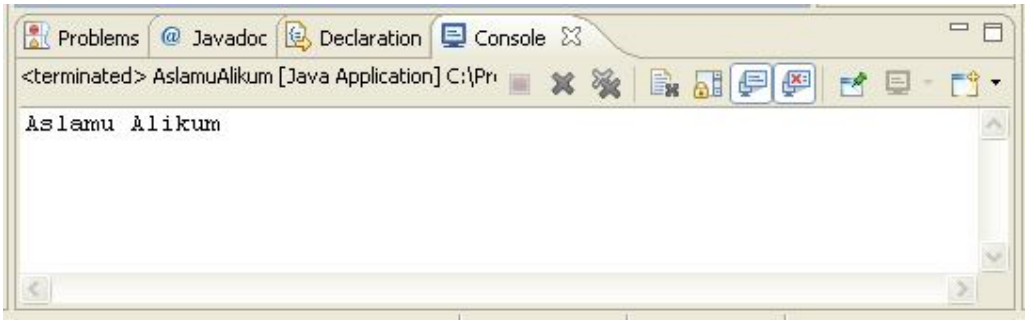
نضيف System.out.print(“Aslamu Alikum”);



تشغيل البرنامج : Run > run as > java application



تظهر النتيجة



مبروك.