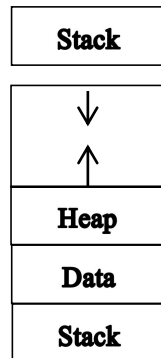


Processes and States

What is a process?

A process is a program in execution. It also includes the current activity, represented by the value of the **program counter**, it also includes a **stack**, **data section**, **heap**



Process in memory

Process Control Block (PCB)

A process control block is a data structure that contains all the per-process information including the address space, memory pointer, identifies, state. At any given point of time, while the program is executing.

Identifier
State
Priority
Program counter
Memory pointers
Context data
I/O status information
Accounting information
• • •

Identifier: A unique identifier associated with this process, to distinguish it from all

other processes.

State: current activity of that process. E.g. running state.

Priority: Priority level relative to other processes.

Program counter: The address of the next instruction to be executed.

Memory pointers: Includes pointers to the program code and data associated with this process, plus any memory blocks shared with other processes.

Context data: These are data that are present in registers in the processor while the process is executing.

I/O status information: Includes outstanding I/O requests, I/O devices assigned to this process, a list of files in use by the process etc.

Accounting information: May include the amount of processor time and clock time used, time limits, account numbers etc.

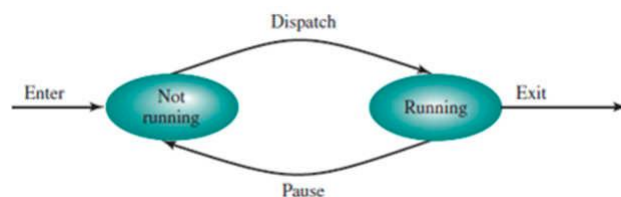
State Process Model and Diagrams

The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states:

- ✓ **New.** The process is being created.
- ✓ **Running.** Instructions are being executed.
- ✓ **Ready.** The process is waiting to be assigned to a processor.
- ✓ **Blocked/Waiting:** A process that cannot execute until some event occurs, such as the completion of an I/O operation.
- ✓ **Terminated.** The process has finished execution.

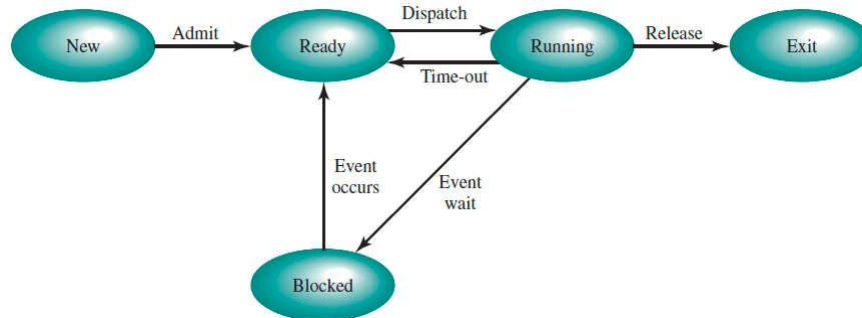
A two State process model

In this model, a process may be in one of two states: **Running** or **Not Running**.



A Five State process model

In this model, the Not Running state is split into two states: **Ready** and **Blocked**.



Null --> New: A new process is created to execute a program.

New --> Ready: The OS will move a process from the New state to the Ready state when it is prepared to take on an additional process.

Ready --> Running: The OS chooses one of the processes in the **Ready** state. This is the job of the scheduler or dispatcher

Running --> Exit: The currently running process is terminated by the OS if the process indicates that it has completed, or if it aborts.

Running --> Ready: The most common reason for this transition is that the running process has reached the maximum allowable time for uninterrupted execution.

Running --> Blocked: A process is put in the Blocked state if it requests something for which it must wait. E.g i/o, waiting for another process or the OS.

Scheduling Algorithms

Scheduling deals with the problem of deciding which of the process in the ready queue is to be allocated to the CPU.

Criteria for Scheduling:

- ✓ CPU Utilization
- ✓ Throughput
- ✓ Turnaround time
- ✓ Waiting time
- ✓ Response time

Two scheduling schemes exist. The **Preemptive** and **Non-preemptive** Schemes:

- a. **Preemptive scheduling** occurs when the scheduler interrupts the CPU after a process transitions from running to ready or waiting to ready, reallocating CPU resources. If the process still has remaining CPU burst time, it returns to the ready queue.
- b. **Non-preemptive scheduling** does not interrupt a running CPU process in the midst of its execution. Instead, it waits until the process's CPU burst duration is over before allocating the CPU to another process.

Types of Scheduling algorithms:

1. **First-In, First-Out (FIFO):** The key concept of this algorithm is that it allocates the CPU in the order in which the processes arrive. FIFO is an example of non-preemptive scheduling.
2. **Shortest-Job First (SJF) Scheduling Algorithm:** When the CPU is available, this algorithm assigned it to the process that has the smallest next CPU burst. The SJF algorithm may be either preemptive or non-preemptive
 - A preemptive SJF will preempt a currently executing process and starts the execution of newly entered process
 - While non-preemptive SJF will allow the currently executing process to complete its burst time without any interruption in its execution.
3. **Round-Robin Scheduling Algorithm:** The Round-Robin (RR) scheduling algorithm is-designed especially for time-sharing systems. A small unit of time called a time quantum (or time-slice) is defined. In this algorithm. The CPU scheduler (short-term) goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

Context Switching

Context/Process Switching: Changing the execution environment of one process to that of another is called process switching.

An O/S can acquire control by:

- **Interrupt**: an independent external event.
- **Trap**: that is associated with current instruction execution.
- **Supervisor call/system call**: that is explicit call to the O/S

Context Switching Steps

The steps involved in context switching are as follows –

- **Save** the context of the process that is currently running on the CPU. Update the **PCB** and other important fields.
- **Move** the **PCB** of the above process into the relevant queue such as the ready queue, I/O queue etc.
- **Select** a new process for execution.
- **Update** the **PCB** of the selected process. This includes updating the process state to running.
- **Update** the memory management data structures as required.
- **Restore** the context of the process that was previously running when it is loaded again on the processor. This is done by loading the previous values of the **PCB** and registers.

System Calls

System calls involve the transfer of control from user code to system (or kernel) code and back again.

Interrupts and Interrupts Handler

Type of Interrupts

Interrupt is an event that requires the operating system's attention at any situation. its sole purpose is to report the occurrence of the event to the operating system and enable it to perform appropriate event handling actions.

Here are some examples of the causes of interrupts.

- Hardware issue, such as a printer paper jam
- Key press by the user, e.g. CTRL ALT DEL
- Software error
- Phone call (mobile device)
- Disk drive indicating it is ready for more data

There are two types of interrupts: **Hardware** and **Software** interrupt

Hardware Interrupts

An electronic signal sent from an external device or hardware to communicate with the processor indicating that it requires immediate attention.

Hardware interrupts are classified into two types

- **Maskable** Interrupts –those which can be disabled or ignored by the microprocessor. These interrupts are either **edge-triggered** or **level-triggered**. level-triggered interrupt is requested by maintaining the interrupt signal at its specified active logic level (high or low). An edge-triggered interrupt is one that is triggered by a change in the level of the interrupt line, either a falling edge (from high to low) or a rising edge (from low to high).
- **Non-maskable** Interrupts (NMI) – Non-Maskable Interrupts are those which cannot be disabled or ignored by microprocessor. E.g. **Trap**

Software Interrupts

The processor itself requests a software interrupt after executing certain instructions or if particular conditions are met. E.g. subroutine calls, errors (exceptions), or traps

Interrupt Handler

The job of the interrupt handler is to service the device and stop it from interrupting

Instruction for Interrupts –

1. **Enable Interrupt (EI)** – The interrupt enable flip-flop is set and all interrupts are enabled. After a system reset, the interrupt enable flip-flop is reset, thus disabling the interrupts.
2. **Disable Interrupt (DI)** – This instruction is used to reset the value of enable flipflop hence disabling all the interrupts.
3. **Set Interrupt Mask (SIM)** – It is used to implement the hardware interrupts by setting various bits to form masks or generate output data via the Serial Output Data (SOD) line.
4. **Read Interrupt Mask (RIM)** – This instruction is used to read the status of the hardware interrupts

Three main classes of interrupts:

1. **I/O interrupts:** An I/O device requires attention; the corresponding interrupt handler must query the device to determine the proper course of action.
2. **Timer interrupts:** Some timer, either a local APIC (Advanced Programmable Interrupt Controller) timer or an external timer, has issued an interrupt; this kind of interrupt tells the kernel that a fixed-time interval has elapsed.
3. **Interprocessor interrupts:** A CPU issued an interrupt to another CPU of a multiprocessor system.

Interrupt Handler Responsibilities

1. Determine if the device is interrupting and possibly reject the interrupt.
2. Inform the device that it is being serviced.
3. Perform any I/O request-related processing.
4. Do any additional processing that could prevent another interrupt.
5. Return DDI_INTR_CLAIMED

Concurrency – Multithreading

Threads

A **thread** is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack. A **thread** of execution is the smallest sequence of programmed instructions that can be managed independently by an operating system scheduler

Multi-threading

Multi-threading refers to the ability of an operating system to support multiple, concurrent paths of execution within a single process

Threads are divided into four categories:

- 1) **One thread per process** (One to one): A simple single-threaded application has one sequence of instructions, executing from beginning to end.
- 2) **Many threads per process** (One to Many): a program may be structured as several concurrent threads, each executing within the restricted rights of the process
- 3) **Many single-threaded processes** (Many to one): The kernel has multiple threads executing concurrently in kernel mode.
- 4) **Many kernel threads** (Many to many): In this case, each kernel thread runs with the privileges of the kernel

Threads Creation

A process is always created with one thread, called the **initial** thread. The initial thread provides compatibility with previous single-threaded processes. The initial thread's stack is the process stack.

Threads Termination

One way for a thread to terminate is simply to return from its first procedure, returning a value of type void

Types of Threads

1. **User level threads** : User level threads are implemented by a thread library. The library set up the thread implementation arrangement without using the kernel, Thus, the kernel is not aware of presence of user-level threads in a process, as it sees only the process.

Advantages of User-level Threads

- Thread switching overhead is smaller than the kernel-level thread.
- This arrangement enables each process to use a scheduling policy that is best suited to it

Disadvantages:

- The kernel does not know the distinction between a thread and a process, so if a thread wants to be blocked, the kernel will block its parent.
- Process ULT cannot provide parallelism and concurrency provided by KLTs.

2. **Kernel-level Thread**: KLT is implemented by kernel. Hence, creation and termination of KLTs, and checking their status is performed by system calls.

Advantages:

- The similarity between process and thread is convenient for programmers, as programming for thread is same as programming for processes.
- In a multiprocessor system, KLTs provides parallelism

Disadvantages:

- KLT limits the saving in the switching overhead.
- When signals are sent, conflict might arise in the course of more than one thread attempting to handle signal at the same time.

3. **Hybrid Thread**: A hybrid thread model has both ULT and KLT and method of associating ULTs with KLTs.

- a) **Many-to-one association method**: In this method a single KLT is created in each process by the kernel and all ULTs created by thread library are associated with this single KLT.

- b) **One-to-one method of association:** In this method each ULT is permanently mapped into a KLT. Parallelism and switching is possible at high overhead.
- c) **Many-to-many association method:** In this method ULT can be mapped to any KLT. By this method parallelism is possible and switching is possible at kernel-level with low overhead.

Threads Data Structure and Lifecycle

Thread Control Block

Thread Control Block (TCB) is a data structure in the operating system kernel which contains thread-specific information needed to manage it. The data structure of thread contains information such as: Thread ID, Stack pointer, Program counter, CPU information, Thread priority and the Pointers.

Thread ID: A unique identifier assigned by the Operating System to the thread when it is being created.

Thread states: These are the states of the thread which changes as the thread progresses through the system

CPU information: It includes everything that the OS needs to know about, such as how far the thread has progressed and what data is being used.

Thread Priority: It indicates the weight (or priority) of the thread over other threads.

A **pointer** which points to the process which triggered the creation of this thread.

A **pointer** which points to the thread(s) created by this thread.

Thread States

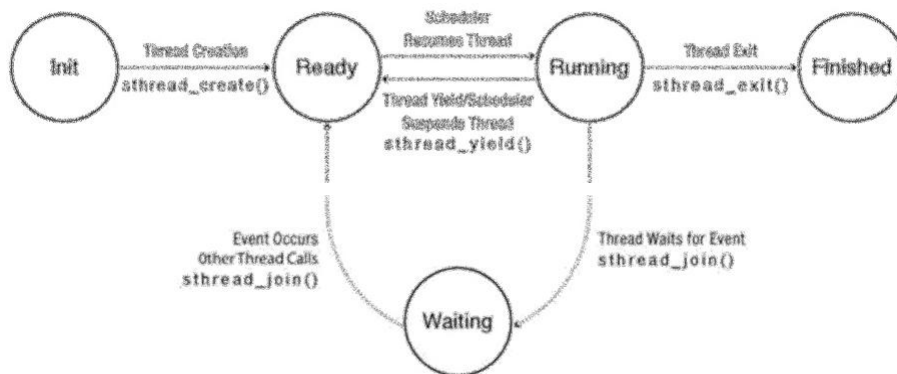
As with processes, the key states for a thread are **Running**, **Ready**, and **Blocked**.

There are four basic thread operations associated with a change in thread state:

1. **Spawn:** Typically, when a new process is spawned, a thread for that process is also spawned.
2. **Block:** When a thread needs to wait for an event, it will block (saving its user registers, program counter, and stack pointers)

3. **Unblock**: When the event for which a thread is blocked occurs, the thread is moved to the Ready queue.
4. **Finish**: When a thread completes, its register context and stacks are deallocated.

Thread lifecycle



INIT: Thread creation puts a thread into its INIT state and allocates and initializes pthread data structures.

READY: A thread in the READY state is available to be run but is not currently running.

RUNNING: A thread in the RUNNING state is running on a processor

WAITING: A thread in the WAITING state is waiting for some event.

FINISHED: A thread in the FINISHED state never runs again.

Race Condition and Critical Region

A race condition on a share data ds is a situation in which the value of ds resulting from the execution of two operations may be different.

Data Access Synchronization

Data access synchronization is coordination of processes to implement mutual exclusion over shared data.

The requirement that only one of the operations a_i , a_j can access shared data ds at any

time is known as **mutual exclusion**.

The following method can be used to prevent race conditions in an application program:

- 1) For every pair of processes that share some data, check whether a race condition exists.
- 2) If so, ensure that processes access shared data in a mutually exclusive manner

Critical Regions

A critical section for a data item *ds* is a section of code that is designed so that it cannot be executed concurrently either with itself or with other critical section(s) for *ds*

```
do {  
    entry section  
    critical section  
    exit section  
    remainder section  
} while (TRUE);
```

A solution to the critical-section problem must satisfy the following three properties:

1. **Mutual exclusion**. If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.
2. **Progress**. If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.
3. **Bounded waiting**. There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Deadlock

Deadlock is the state of permanent blocking of a set of processes each of which is waiting for an event that only another process in the set can cause.

Types of deadlock:

1. Resource deadlock
2. Synchronization deadlock

Conditions for a Resource Deadlock

1. **Non-shareable resources cannot be shared**; a process needs exclusive access to a resource.
2. **No Preemption**: A resource cannot be preempted from one process and allocated to another process.
3. A **Hold-and-wait** process continues to hold the resources allocated to it while waiting for other resources.
4. A **circular wait chain** of hold-and-wait conditions exists in the system; e.g., process P_i waits for P_j , P_j waits for P_k , and P_k waits for P_i .

Deadlock Detection

- An algorithm that examines the state of the system to determine whether a deadlock has occurred.
- An algorithm to recover from the deadlock

Deadlock Detection Algorithm

Algorithm below performs deadlock detection By simulating two sets of processes Blocked and Running, The algorithm terminates when no processes are left in the Running set.

Deadlock Avoidance

In deadlock avoidance, for each new request, the system consider the resources

currently available, the resources currently allocated to the processes and further request and releases of each process, to decide whether the current request can be satisfied or must wait to avoid a possible future deadlock.

Safe State

A **state** is **safe**, if the system can allocate resources to each process (up to its maximum) in some order and avoid deadlock. Formally, a system is said to be in safe state only if there exist a **safe sequence**.

A sequence of processes $\langle P_1, P_2, \dots, P_n \rangle$ is a safe sequence for the current allocation state iff for each process P_i , the resources that P_i can still request can be satisfied by the currently available resources plus the resources held by all the processes P_j with $P_j < P_i$.

Banker's Algorithm

When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in safe state. If it will, the resources are allocated, otherwise, the process must wait until some other process releases enough resources.

Deadlock Prevention

This approach is based on the idea of designing the system in such a way that deadlocks become impossible, by avoiding all the conditions necessary for deadlock to occur.

Deadlock Prevention Policy

All Resources Together

This is the simplest of all deadlock prevention policies. A process must ask for all resources it needs in a single request; the kernel allocates all of them together

Synchronization

Semaphores: Semaphores are integer variables used to solve the critical section problem by using two atomic operations, wait and signal for process synchronization.

Types of Semaphores

Counting Semaphores

The semaphore count is the number of available resources. If the resources are added, semaphore count is automatically incremented and if the resources are removed, the count is decremented.

Binary Semaphores

The binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0.

Advantages of Semaphores

1. Semaphores allow only one process into the critical section
2. There is no resource wastage, processor time is
3. not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.
4. Semaphores are implemented in the machine independent code of the microkernel. So they are machine independent.

Disadvantages of Semaphores

1. Semaphores are complicated so the wait and signal operations must be implemented in the correct order to prevent deadlocks.
2. Semaphores are impractical for large scale use as their use leads to loss of modularity
3. Semaphores may lead to a priority inversion where low priority processes may access the critical section first and high priority processes later.

Monitors

A **monitor** is a collection of procedures, variables, and data structures that are all grouped together in a special kind of module or package. Only one process can be active in a monitor at any instant

Synchronization Problems

A solution to a process synchronization problem should meet three important criteria:

- **Correctness:** Data access synchronization and control synchronization should be performed in accordance with synchronization requirements of the problem.
- **Maximum concurrency:** A process should be able to operate freely except when it needs to wait for other processes to perform synchronization actions.
- **No busy waits:** To avoid performance degradation, synchronization should be performed through blocking rather than through busy waits.

Producer – Consumer Problem

A producers–consumers system with bounded buffers consists of an unspecified number of producer and consumer processes and a finite pool of buffers. A producer process produces one item of information at a time and writes it into an empty buffer. A consumer process consumes information one item at a time from a full buffer.

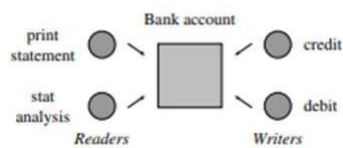
A solution to the producers – consumers problem must satisfy the following conditions:

1. A producer must not overwrite a full buffer.
2. A consumer must not consume an empty buffer.
3. Producers and consumers must access buffers in a mutually exclusive manner.
4. Information must be consumed in the same order in which it is put into the buffers, i.e., in **FIFO** order.

Reader-Writer Problem

A solution to the readers–writers problem must satisfy the following conditions:

1. Many readers can perform reading concurrently.
2. Reading is prohibited while a writer is writing.
3. Only one writer can perform writing at any time.
4. A reader has a non-preemptive priority over writers; i.e., it gets access to the shared data ahead of a waiting writer, but it does not preempt an active writer.



Dining Philosophers

```
var    successful : boolean;
repeat
    successful := false;
    while (not successful)
        if both forks are available then
            lift the forks one at a time;
            successful := true;
        if successful = false
            then
                block (Pi);
    { eat }
    put down both forks;
    if left neighbor is waiting for his right fork
    then
        activate (left neighbor);
    if right neighbor is waiting for his left fork
    then
        activate (right neighbor);
    { think }
forever
```

Memory Management

Memory Swapping and Addresses

Memory swapping is a computer technology that enables an operating system to provide more memory to a running application or process than is available in physical random access memory (RAM).

Logical and Physical Addresses

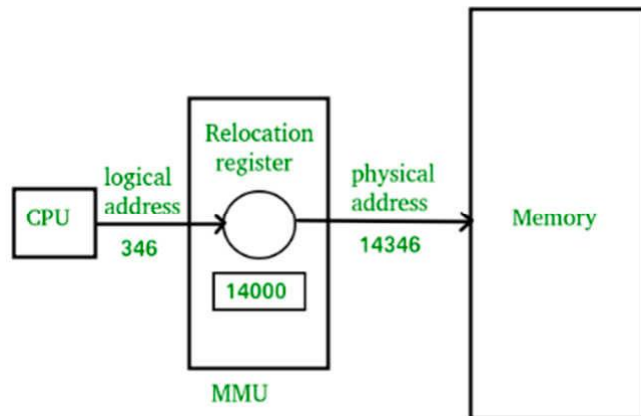
Logical Address: Also known as virtual address, is an address generated by CPU while a program is running.

The term **Logical Address Space** is used for the set of all logical addresses generated

by a program's perspective.

Physical Address: Identifies a physical location of required data in a memory. It can be accessed by its corresponding logical address.

The term **Physical Address Space** is used for all physical addresses corresponding to the logical addresses in a Logical address space.



Relocation is a method of shifting a user program from one memory location to another. There are two types of relocation: **static** and **dynamic**. Static relocation takes place at load time and remain fixed once the program is relocated, while the dynamic relocation takes place at run time.

Differences Between Logical and Physical Address in Operating System

1. The basic difference between Logical and physical address is that Logical address is **generated** by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
2. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address **mapped** to corresponding logical addresses is called Physical Address Space
3. The logical address does not **exist** physically in the memory whereas physical address is a location in the memory that can be accessed physically.
4. Identical logical addresses are generated by Compile-time with Load time address binding methods whereas they differ from each other in run-time address binding method.

5. The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

Address Binding Mechanism

Memory consists of large array of words or bytes, each with its own address

Address binding of instructions and data to memory addresses can happen at three different stages:

1. **Compile time:** Binding at compile time generates absolute addresses where a prior knowledge is required in the main memory.
2. **Load time:** If it is not known at compile time where a process will reside in memory then the compiler must generate relocatable address.
3. **Execution time:** This method permits moving of a process from one memory segment to another during run time

Memory Swapping

Swapping is a memory management technique and is used to temporarily remove the inactive programs from the main memory of the computer system

Swap In and Swap Out in OS

The procedure by which any process gets **removed** from the hard disk and placed in the main memory or **RAM** commonly known as **Swap In**.

On the other hand, **Swap Out** is the method of removing a process from the main memory or RAM and then adding it to the Hard Disk.

Advantages of Swapping

1. Helps the CPU to manage multiple processes within a single main memory.
2. Helps to create and use virtual memory.
3. CPU can perform several tasks simultaneously. Thus, processes need not wait too long before their execution.

4. economical.
5. improve performance of priority-based scheduling

Disadvantages of Swapping

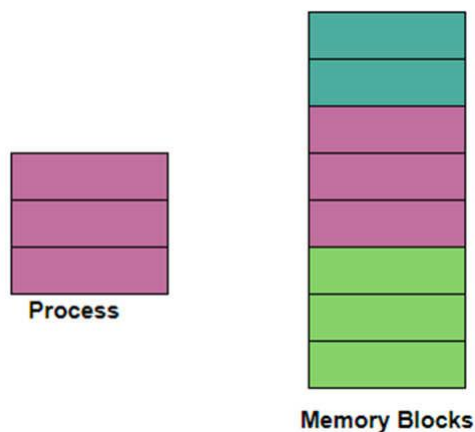
1. There may be inefficiency if a resource or variable is commonly used by processes participating in the swapping process.
2. If the algorithm used for swapping is not good swapping can increase the number of page faults and thus decline the overall performance of processing.
3. If the computer system loses power at the time of high swapping activity, the user might lose all the information related to the program.

Memory Partition

Contiguous and non-contiguous Memory Allocation

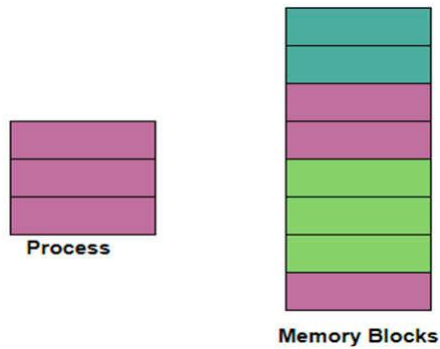
Contiguous Memory Allocation

Contiguous memory allocation is achieved by dividing the memory into the fixed-sized partition.



Non-Contiguous Memory Allocation

In a non-contiguous memory allocation, a process is allowed to acquire several memory blocks at different locations in the memory according to its need.



Memory Partitioning

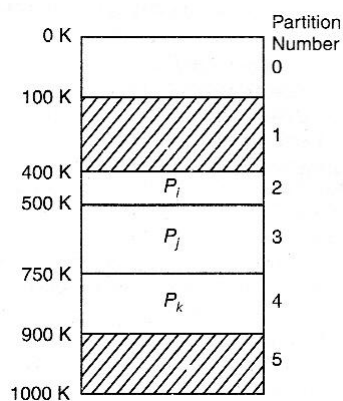
Memory partitioning is the system by which the memory of a computer system is divided into sections for use by the resident programs. There are different ways in which memory can be partitioned (i.e fixed, variable, and dynamic partitioning).

Fixed partitioning

It is defined as the system of dividing memory into non-overlapping sizes that are fixed, unmovable, static. A process may be loaded into a partition of equal or greater size and is confined to its allocated partition.

Unoccupied space is known as **fragmentation**. Within the fixed partition context, this is known as **internal fragmentation (IF)**.

Internal fragmentation is the difference in size between the process and its allocated partition.

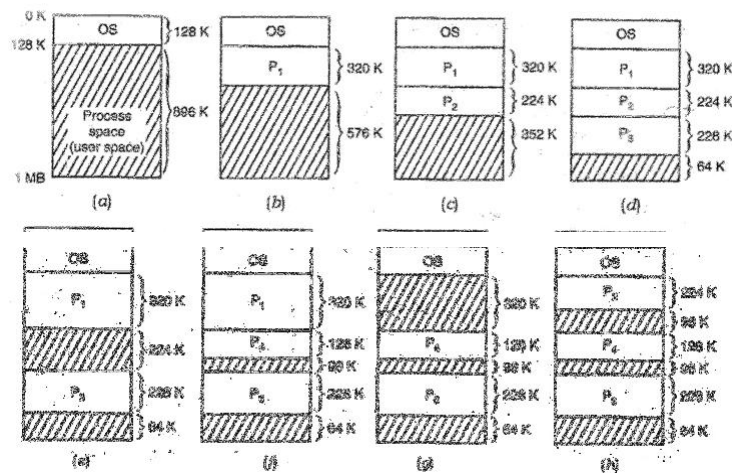


Variable partitioning

Variable partitioning is the system of dividing memory into non-overlapping but variable sizes. Small processes are fit into small partitions (item 1) and large processes fit into larger partitions (items 2 and 3).

Dynamic partitioning

In dynamic partitioning, the partitions used are of variable length. When a process is brought into the memory, it is allocated exactly to that amount of memory that it needs, not more than that it requires.



Partitioning Descriptive Table (PDT)

The OS keeps track current partitions status and attributes, such as free or in use, for allocation purposes in a data structure called **PDT**.

Partition No.	Partition base	Partition size	Partition status
0	0 K	100 K	Allocated
1	100 K	300 K	Free
2	400 K	100 K	Allocated
3	500 K	250 K	Allocated
4	750 K	150 K	Allocated
5	900 K	100 K	Free

Partition Allocation Methods

1. **Single contiguous allocation:** All memory (except some reserved for OS) is available to a process.

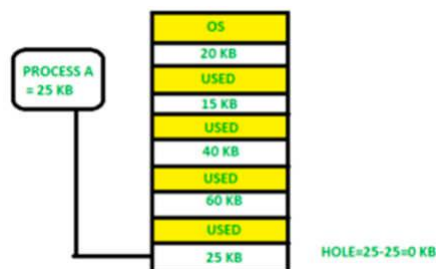
2. **Partition allocation:** Memory is divided into different blocks or partitions. Each process is allocated according to the requirement.

There are different Placement Algorithm:

A. **First Fit:** In the first fit, the partition allocated is the first sufficient block from the top of Main Memory.



B. **Best Fit:** Allocate the process to the partition which is the first smallest sufficient partition among the free available partition.



C. **Worst Fit:** Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory.



D. **Next Fit:** Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

Virtual Memory

Virtual memory is a part of the memory hierarchy that consists of memory and a disk. implemented through the non-contiguous memory allocation model and comprises both hardware components and a software component called a **virtual memory manager**

Paging

Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous.

The physical memory is conceptually divided into a number of fixed –size blocks called **frames** and the logical address space is also split into fixed size blocks, called **pages**

Memory Segmentation

Segmentation is different from paging as pages are physical in nature and hence are of **fixed** size, whereas segmentation are logical division of a program and hence are of **variable** size.

Segmentation is a method of dividing memory into independent address space called segments.

Memory Caching

Cache Concept

A cache is a copy of a computation or data that can be accessed more quickly than the original.

The simplest kind of a cache is a **memory cache**. It stores (address, value) pairs.

If the cache has the value of a certain memory, that is called a **cache hit**. If the cache does not, that is called a **cache miss**.

$$\text{Latency}(\text{read request}) = \text{Prob}(\text{cache hit}) \times \text{Latency}(\text{cache hit}) + \text{Prob}(\text{cache miss}) \times \text{Latency}(\text{cache miss})$$

Memory Hierarchy

1. **First-level cache:** designed to keep the processor fed with instructions and data at the clock rate of the processor.
2. **Second-level cache:** physically addressed cache to handle cache misses from the first-level cache.
3. **Third-level cache.** An even larger, slower third-level cache to catch second-level cache misses.
4. **First-and second-level TLB.** The translation look-aside buffer (TLB) organized with multiple levels: a small, fast first-level TLB designed to keep up with the processor, backed up by a larger, slightly slower, second-level TLB to catch first-level TLB misses.
5. **Main memory (DRAM):** from a software perspective, main memory itself can be viewed as a cache
6. **Data center memory (DRAM):** With a high-speed local area network such as a data center, the latency to fetch a page of data from the memory of a nearby computer is much faster than fetching it from disk.
7. **Local disk or non-volatile memory:** For client machines, local disk or non-volatile flash memory can serve as backing store when the system runs out of memory
8. **Data center disk:.** The aggregate disks inside a data center provide enormous storage capacity compared to a computer's local disk
9. **Remote data center disk.** Geographically remote disks in a data center are much slower because of wide-area network latency, but they provide access to even larger storage capacity in aggregate. Typically accessed only in event of a failure.

Memory Cache Lookup

A memory cache maps a sparse set of addresses to the data values stored at those addresses

Three common mechanisms for cache lookup are:

1. **Fully associative:** With a fully associative cache, the address can be stored anywhere in the table, and so on a lookup the system must check the address against all of the entries in the table.
2. **Direct mapped:** With a direct mapped cache, each address can only be stored in one location in the table.
3. **Set associative:** A set associative cache melds the two approaches, allowing a tradeoff of slightly slower lookup than a direct mapped cache in exchange for most of the flexibility of a fully associative cache.

Thrashing and Replacement Policies

Thrashing

Thrash is the poor performance of a virtual memory (or paging) system when the same pages are being loaded repeatedly due to a lack of main memory to keep them in memory

1. Global Page Replacement

Global page replacement tries to bring more pages whenever thrashing is found. However, no process gets enough frames as a result thrashing will increase, making this an unsuitable algorithm.

2. Local Page Replacement

Unlike the global page replacement algorithm, local page replacement will select pages which only belong to that process.

Causes of Thrashing

If CPU utilization is too low, The CPU scheduler sees the decreasing CPU utilization and increases the degree of multi-programming. As the degree of multi-programming increases, CPU utilization also increases. If the degree of multi-programming is increased further, thrashing sets in, and CPU utilization drops sharply. So, at this point, to increase CPU utilization and to stop thrashing, we must decrease the degree of multi-programming

How to Eliminate Thrashing

1. Adjust the swap file size
2. Increase the amount of RAM
3. Decrease the number of applications running on the computer
4. Replace programs

Replacement Policies

Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Page Replacement Algorithms

1. **First In First Out(FIFO):** When a page needs to be replaced, the page in the front of the queue is selected for removal.
2. **Optimal Page replacement:** In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.
3. **Least Recently Used:** In this algorithm page will be replaced which is least recently used.