جامعة الفيوم

كلية الحاسبات والذكاء الاصطناعى

مؤسسة تعليمية معتمدة

الكتاب الجامعى

2025 - 2024

نظم التشغيل

FCAIFYM

Fayoum University
Faculty of Computers & Artificial Intelligence

Introduction to Operating System

3rd Level Students
2024 - 2025

Table of Contents

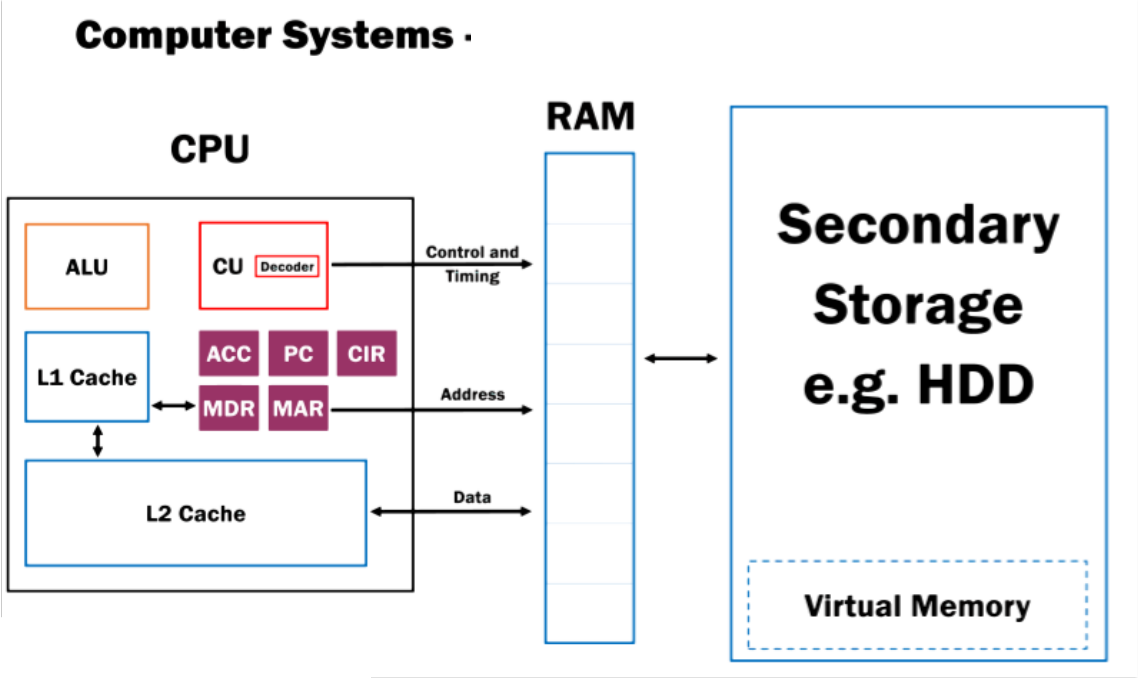
Chapter 1
The Basics of Computer System

1.1 Processor

The processor is an important part of a computer architecture, without it nothing would happen. It is a programmable device that takes input, perform some arithmetic and logical operations and produce some output. In simple words, a processor is a digital device on a chip which can fetch instruction from memory, decode and execute them and provide results.

A processor takes a bunch of instructions in machine language and executes them, telling the processor what it has to do. Processors performs three basic operations while executing the instruction:

1. It performs some basic operations like addition, subtraction, multiplication, division and some logical operations using its Arithmetic and Logical Unit (ALU).
2. Data in the processor can move from one location to another.
3. It has a Program Counter (PC) register that stores the address of next instruction based on the value of PC.



A typical processor structure looks like this:-

Control Unit (CU)

A control unit (CU) handles all processor control signals. It directs all input and output flow, fetches the code for instructions and controlling how data moves around the system.

Arithmetic and Logic Unit (ALU)

The arithmetic logic unit is that part of the CPU that handles all the calculations the CPU may need, e.g. Addition, Subtraction, Comparisons. It performs Logical Operations, Bit Shifting Operations, and Arithmetic Operation.

Registers

1.**Accumulator (AC):** Stores the results of calculations made by ALU.

2. **Program Counter (PC):** Keeps track of the memory location of the next instructions to be dealt with. The PC then passes this next address to Memory Address Register (MAR).

3. **Memory Address Register (MAR):** It stores the memory locations of instructions that need to be fetched from memory or stored into memory.

4. Memory Data Register (MDR): It stores instructions fetched from memory or any data that is to be transferred to, and stored in, memory.

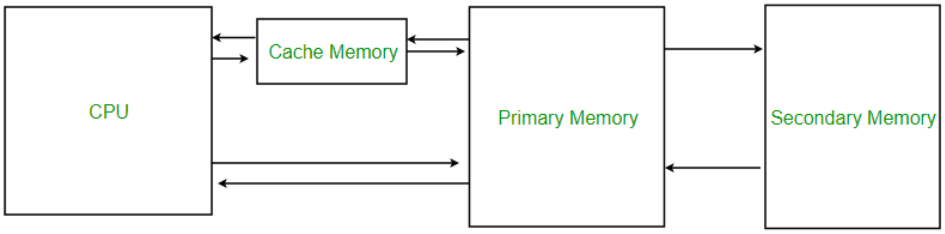
5. Instruction Register (IR): It stores the most recently fetched instructions while it is waiting to be coded and executed.

6. Instruction Buffer Register (IBR): The instruction that is not to be executed immediately is placed in the instruction buffer register IBR.

1.2 Memory

Memory attached to the CPU is used for storage of data and instructions and is called internal memory The internal memory is divided into many storage locations, each of which can store data or instructions. Each memory location is of the same size and has an address. With the help of the address, the computer can read any memory location easily without having to search the entire memory. when a program is executed, it's data is copied to the internal memory and is stored in the memory till the end of the execution. The internal memory is also called the Primary memory or Main memory. This memory is also called as RAM, i.e. Random Access Memory. The time of access of data is independent of its location in memory, therefore this memory is also called Random Access memory (RAM).

Levels of memory:



Level 1 or Register

It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.

Level 2 or Cache memory

It is the fastest memory which has faster access time where data is temporarily stored for faster access. Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.

Level 3 or Main Memory (Primary Memory in the image above)

It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.

Level 4 or Secondary Memory

It is external memory which is not as fast as main memory but data stays permanently in this memory.

Cache Performance:

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache.

If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

Hit ratio = hit / (hit + miss) = no. of hits/total accesses

We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty, and reduce the time to hit in the cache.

Application of Cache Memory

1. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.
2. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

1.3 Input/Output Devices

Program or data is read into main memory from the *input device* or secondary storage under the control of CPU input instruction. *Output devices* are used to output the information from a computer.

Buses – Data is transmitted from one part of a computer to another, connecting all major internal components to the CPU and memory, by the means of Buses. Types:

1. **Data Bus (Data):** It carries data among the memory unit, the I/O devices, and the processor.
2. **Address Bus (Address):** It carries the address of data (not the actual data) between memory and processor.
3. **Control Bus (Control and Timing):** It carries control commands from the CPU (and status signals from other devices) in order to control and coordinate all the activities within the computer.

1.4 Bus System

Data is transmitted from one part of a computer to another, connecting all major internal components to the CPU and memory, by the means of Buses. Types:

Control Bus

In computer architecture, a control bus is part of the system bus, used by CPUs for communicating with other devices within the computer. While the address bus carries the information about the device with which the CPU is communicating and the data bus carries the actual data being processed, the control bus carries commands from the CPU and returns status signals from the devices. For example, if the data is being read or written to the device the appropriate line (read or write) will be active (logic one).

Address bus

An address bus is a bus that is used to specify a physical address. When a processor or DMA-enabled device needs to read or write to a memory location, it specifies that memory location on the address bus (the value to be read or written is sent on the data bus). The width of the address bus determines the amount of memory a system can address. For example, a system with a 32-bit address bus can address 232 (4,294,967,296) memory locations. If each memory location holds one byte, the addressable memory space is 4 GiB.

Data / Memory Bus

The memory bus is the computer bus which connects the main memory to the memory controller in computer systems. Originally, general-purpose buses like VMEbus and the S-100 bus were used, but to reduce latency, modern memory buses are designed to connect directly to DRAM chips, and thus are designed by chip standards bodies such as JEDEC. Examples are the various generations of SDRAM, and serial point-to-point buses like SLDRAM and RDRAM. An exception is the Fully Buffered DIMM which, despite being carefully designed to minimize the effect, has been criticized for its higher latency.

1.5 Instruction Cycle

The **instruction cycle** (also known as the **fetch–decode–execute cycle**, or simply the **fetch-execute cycle**) is the cycle that the central processing unit (CPU) follows from boot-up until the computer has shut down in order to process instructions. It is composed of three main stages: the fetch stage, the decode stage, and the execute stage.

Role of components

The program counter (PC) is a special register that holds the memory address of the next instruction to be executed. During the fetch stage, the address stored in the PC is copied into the memory

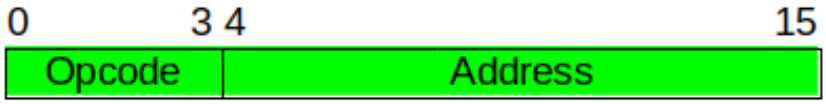
address register (MAR) and then the PC is incremented in order to "point" to the memory address of the next instruction to be executed. The CPU then takes the instruction at the memory address described by the MAR and copies it into the memory data register (MDR). The MDR also acts as a two-way register that holds data fetched from memory or data waiting to be stored in memory (it is also known as the memory buffer register (MBR) because of this). Eventually, the instruction in the MDR is copied into the current instruction register (CIR) which acts as a temporary holding ground for the instruction that has just been fetched from memory.

During the decode stage, the control unit (CU) will decode the instruction in the CIR. The CU then sends signals to other components within the CPU, such as the arithmetic logic unit (ALU) and the floating point unit (FPU). The ALU performs arithmetic operations such as addition and subtraction and also multiplication via repeated addition and division via repeated subtraction. It also performs logic operations such as AND, OR, NOT, and binary shifts as well. The FPU is reserved for performing floating-point operations.

Each computer's CPU can have different cycles based on different instruction sets, but will be similar to the following cycle:

1. **Fetch Stage**: The next instruction is fetched from the memory address that is currently stored in the program counter and stored into the instruction register. At the end of the fetch operation, the PC points to the next instruction that will be read at the next cycle.

2. **Decode Stage**: During this stage, the encoded instruction presented in the instruction register is interpreted by the decoder.



Read the effective address: In the case of a memory instruction (direct or indirect), the execution phase will be during the next clock pulse. If the instruction has an indirect address, the effective address is read from main memory, and any required data is fetched from main memory to be processed and then placed into data registers. If the instruction is direct, nothing is done during this clock pulse. If this is an I/O instruction or a register instruction, the operation is performed during the clock pulse.

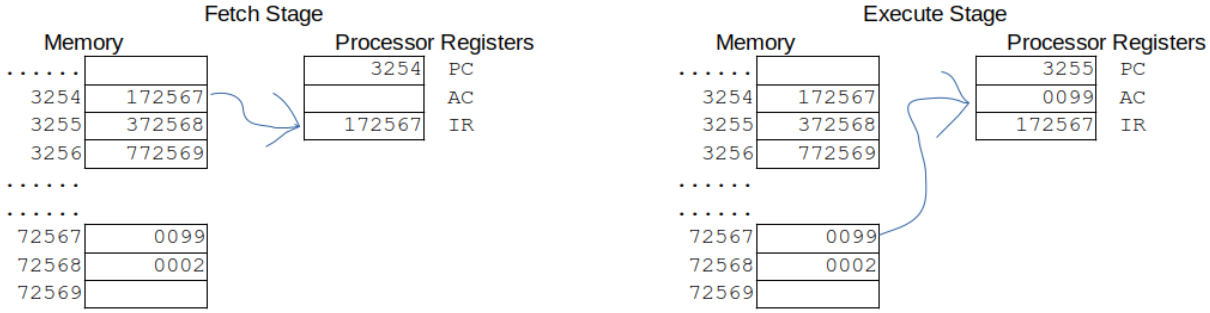
3. **Execute Stage**: The control unit of the CPU passes the decoded information as a sequence of control signals to the relevant functional units of the CPU to perform the actions required by the instruction, such as reading values from registers, passing them to the ALU to perform

mathematical or logic functions on them, and writing the result back to a register. If the ALU is involved, it sends a condition signal back to the CU. The result generated by the operation is stored in the main memory or sent to an output device. Based on the feedback from the ALU, the PC may be updated to a different address from which the next instruction will be fetched.

Example:-

we include an Accumulator (AC) which will be used as a temporary storage location for the data. There will be 3 opcodes - PLEASE understand these are sample opcode for this example...do NOT confuse these with actual processor opcodes.

1. 0001 - this opcode tells the processor to load the accumulator (AC) from the given memory address.
2. 0011 - this opcode tells the processor to add to the value currently stored in the AC from the specified memory address.
3. 0111 - this opcode tells the processor to move the value in the AC to the specified memory address.



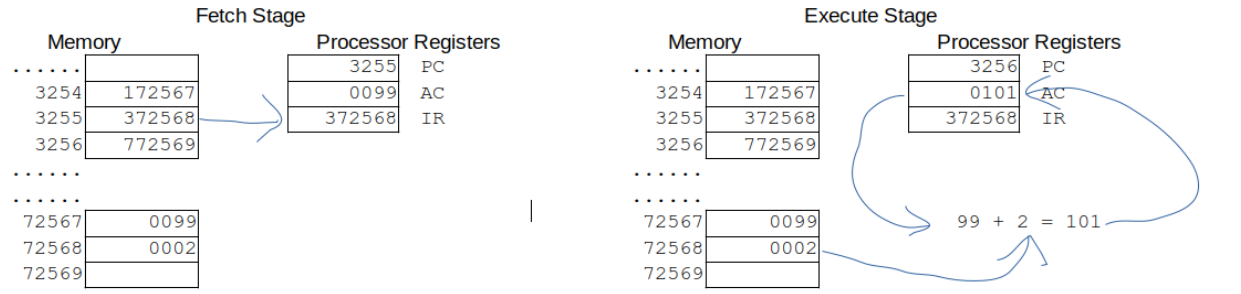
1. At the beginning the PC contains 3254, which is the memory address of the next instruction to be executed. In this example we

have skipped the micro-steps, showing the IR receiving the value at the specified address.

2. The instruction at address 3254 is 172567. Remember from above - the first 4 bits are the opcode, in this case it is the number 1 (0001 in binary).

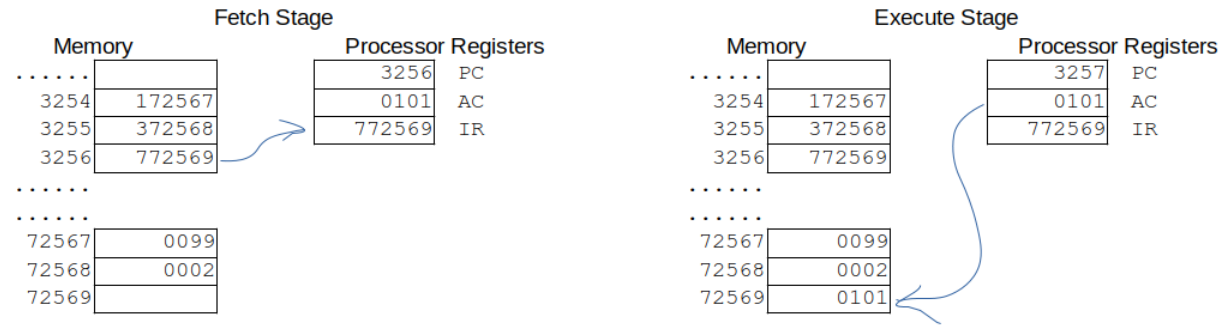
3. This opcode tells the processor to load the AC from the memory address located in the last 12 bits of the instruction - 72567. 4. Go to address 72567 and load that value, 0099, into the accumulator.

ALSO NOTICE - the PC has been incremented by 1, so it now points to the next instruction in memory.



1. Again - we start with the PC - and move the contents found at the memory address, 3255, into the IR.
2. The instruction in the IR has an opcode of 3 (0011 in binary).
3. This opcode in our example tells the processor to add the value currently stored in the AC, 0099, to the value stored at the given
   memory address, 72568, which is the value 2.
4. This value, 101, is stored back into the AC.

AGAIN, the PC has been incremented by one as well.



1. The PC points to 3256, the value 772569 is moved to the IR.

2. This instruction has an opcode of 7 (0111 in binary)

3. This opcode tells the processor to move the value in the AC, 101, to the specified memory address, 72569.

The PC has again been incremented by one - and when our simple 3 instructions are completed, whatever instruction was at that address would be executed.

1.6 Data Transfer Modes

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

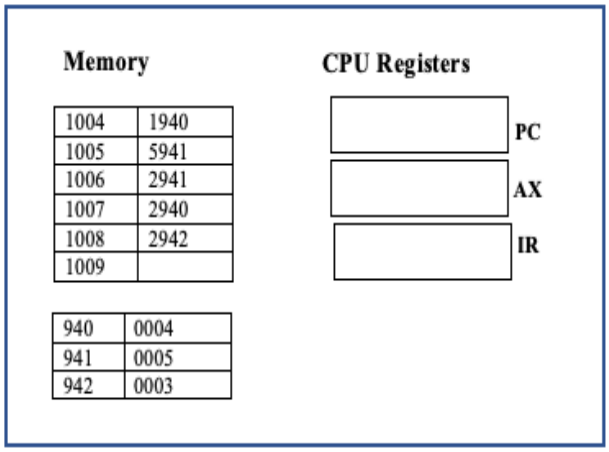
1. **Programmed I/O:** is the result of the I/O instructions written in the program's code. Each data transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and/or memory. In this case it requires constant monitoring by the CPU of the peripheral devices.
2. **Interrupt- initiated I/O:** using an interrupt facility and special commands to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed processing other programs. The interface meanwhile keeps monitoring the device. When it is determined that the device is ready for a data transfer it initiates an interrupt request signal to the CPU. Upon detection of an external interrupt signal the CPU momentarily stops the task it was processing, and services program that was waiting on the interrupt to process the I/O transfer> Once the interrupt is satisfied, the CPU then return to the task it was originally processing.
3. **Direct memory access(DMA)**: The data transfer between a fast storage media such as magnetic disk and main memory is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as direct memory access, or DMA. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

EXERCISES

1. If the instruction that will be executed is an add operation and Ax contains 0009H with the following format:

   a. What would be the contents of the registers PC and IR during this operation?
   b. What would be the contents of Ax after completion of this operation?

opcodes:
0001: load from memory
0010: Store AC to memory
0101: Add to AC from memory



2. Consider a hypothetical 32-bit microprocessor having 32-bit instructions composed of two fields. The first byte contains the opcode and the remainder an immediate operand or an operand address.
   a. What is the maximum directly addressable memory capacity in bytes?
   b. Discuss the impact on the system speed if the microprocessor bus has
      - A 32-bit local address bus and 16-bit local data bus.
      - A 16-bit local address bus and 16-bit local data bus.
   c. How many bits are needed for the program counter and the instruction register?

3. Consider a hypothetical microprocessor generating a 16-bit address where the address counter and the address register are 16 bits wide and having a 16-bit data bus.
   a. What is the maximum memory address space that the processor can access directly if it is connected to a 16-bit memory?
   b. What is the maximum memory address space that the processor can access directly if it is connected to an 8-bit memory?
   c. If an input and an output instruction can specify an 8-bit I/O port number, how many 8-bit I/O ports can the microprocessor support? How many 16-bit I/O ports?

Chapter 2

Introduction to Operating System

An operating system acts as an interface between the user and computer hardware. The purpose of an operating system is to provide a platform on which a user can execute programs in a convenient and efficient manner. The coordination of the hardware must be appropriate to ensure the correct working of the computer system and to prevent user programs from interfering with the proper working of the system.

Example: Just like a boss gives order to his employee, in the similar way we request or pass our orders to the operating system. The main goal of the operating system is to thus make the computer environment more convenient to use and the secondary goal is to use the resources in the most efficient manner.

2.1 Definition

An operating system is a program on which application programs are executed and acts as an communication bridge (interface) between the user and the computer hardware.

The main task an operating system carries out is the allocation of resources and services, such as allocation of: memory, devices, processors and information. The operating system also includes programs to manage these resources, such as a traffic controller, a scheduler, memory management module, I/O programs, and a file system.

2.2 OS Functions

Important functions of an operating system:

1. **Security**

The operating system uses password protection to protect user data and similar other techniques. it also prevents unauthorized access to programs and user data.

2. **Control over system performance**

Monitors overall system health to help improve performance. records the response time between service requests and system response to have a complete view of the system health. This can help improve performance by providing important information needed to troubleshoot problems.

3. **Job accounting**

Operating system keeps track of time and resources used by various tasks and users, this information can be used to track resource usage for a particular user or group of user.

4. **Error detecting aids**

Operating system constantly monitors the system to detect errors and avoid the malfunctioning of computer system.

5. **Coordination between other software and users**

Operating systems also coordinate and assign interpreters, compilers, assemblers and other software to the various users of the computer systems.

6. **Memory Management**

The operating system manages the primary memory or main memory. Main memory is made up of a large array of bytes or words where each byte or word is assigned a certain address. Main memory is a fast storage and it can be accessed directly by the CPU. For a program to be executed, it should be first loaded in the main memory. An operating system performs the following activities for memory management:

It keeps tracks of primary memory, i.e., which bytes of memory are used by which user program. The memory addresses that have already been allocated and the memory addresses of the memory that has not yet been used. In multi programming, the OS decides the order in which process are granted access to memory, and for how long. It Allocates the memory to a process when the process requests it and deallocates the memory when the process has terminated or is performing an I/O operation.

7. **Processor Management**

In a multi programming environment, the OS decides the order in which processes have access to the processor, and how much processing time each process has. This function of OS is called process scheduling. An operating system performs the following activities for processor management.

Keeps tracks of the status of processes. The program which perform this task is known as traffic controller. Allocates the CPU that is processor to a process. De-allocates processor when a process is no more required.

8. **Device Management**

An OS manages device communication via their respective drivers. It performs the following activities for device management. Keeps tracks of all devices connected to system. designates a program responsible for every device known as the Input/Output controller. Decides which process gets access to a certain device and for how long. Allocates devices in an effective and efficient way. Deallocates devices when they are no longer required.

9. **File Management**

A file system is organized into directories for efficient or easy navigation and usage. These

directories may contain other directories and other files. An operating system carries out the following file management activities. It keeps track of where information is stored, user access settings and status of every file and more... These facilities are collectively known as the file system.

2.3 OS Services

The operating system provides certain services to the users which can be listed in the following manner:

1. **Program Execution**

The operating system is responsible for execution of all types of programs whether it be user programs or system programs. The operating system utilizes various resources available for the efficient running of all types of functionalities.

2. **Handling Input/Output Operations**

The operating system is responsible for handling all sort of inputs, i.e, from keyboard, mouse, desktop, etc. The operating system does all interfacing in the most appropriate manner regarding all kind of inputs and outputs.

For example, there is difference in nature of all types of peripheral devices such as mouse or keyboard, then operating system is responsible for handling data between them.

3. **Manipulation of File System**

The operating system is responsible for making of decisions regarding the storage of all types of data or files, i.e, floppy disk/hard disk/pen drive, etc. The operating system decides as how the data should be manipulated and stored.

4. **Error Detection and Handling**

The operating system is responsible for detection of any types of error or bugs that can occur while any task. The well secured OS sometimes also acts as countermeasure for preventing any sort of breach to the computer system from any external source and probably handling them.

5. **Resource Allocation**

The operating system ensures the proper use of all the resources available by deciding which resource to be used by whom for how much time. All the decisions are taken by the operating system.

6. **Accounting**

The operating system tracks an account of all the functionalities taking place in the computer

system at a time. All the details such as the types of errors occurred are recorded by the operating system.

7. **Information and Resource Protection**

The operating system is responsible for using all the information and resources available on the machine in the most protected way. The operating system must foil an attempt from any external resource to hamper any sort of data or information.

All these services are ensured by the operating system for the convenience of the users to make the programming task easier. All different kinds of operating system more or less provide the same services.

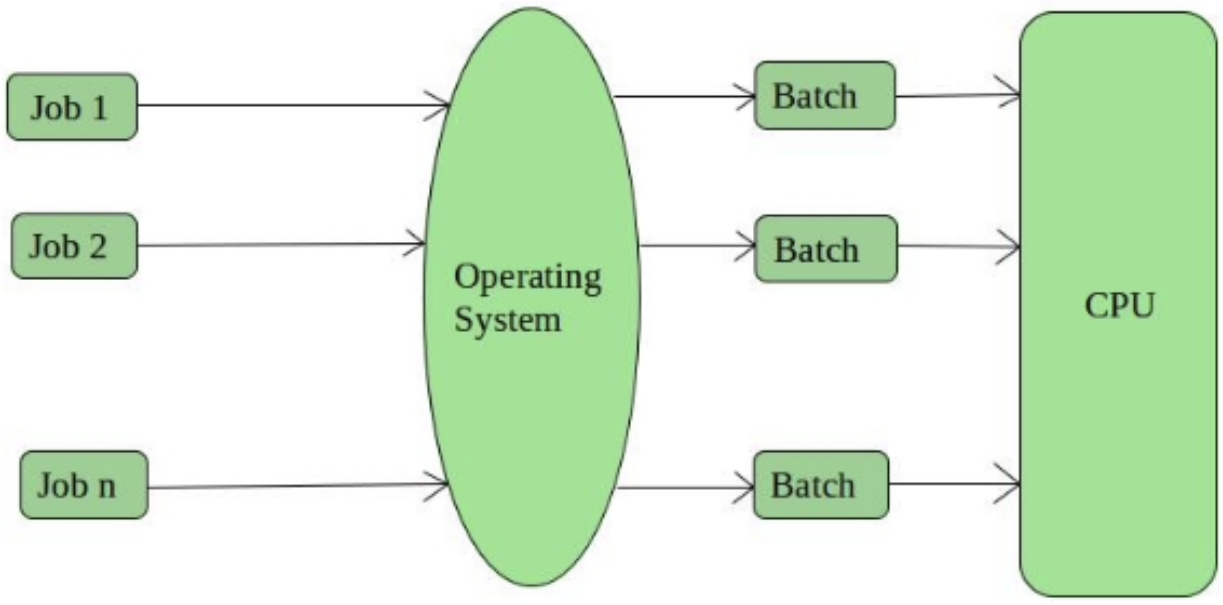
2.4 OS Types

An **Operating System** performs all the basic tasks like managing file,process, and memory. Thus operating system acts as manager of all the resources, i.e. **resource manager**. Thus operating system becomes an interface between user and machine.

Types of Operating Systems: Some of the widely used operating systems are as follows-

1. Batch Operating System

This type of operating system does not interact with the computer directly. There is an operator which takes similar jobs having same requirement and group them into batches. It is the responsibility of operator to sort the jobs with similar needs.



Advantages of Batch Operating System:

It is very difficult to guess or know the time required by any job to complete. Processors of the batch systems know how long the job would be when it is in queue

Multiple users can share the batch systems

The idle time for batch system is very less

It is easy to manage large work repeatedly in batch systems

Disadvantages of Batch Operating System:

The computer operators should be well known with batch systems Batch systems are hard to debug

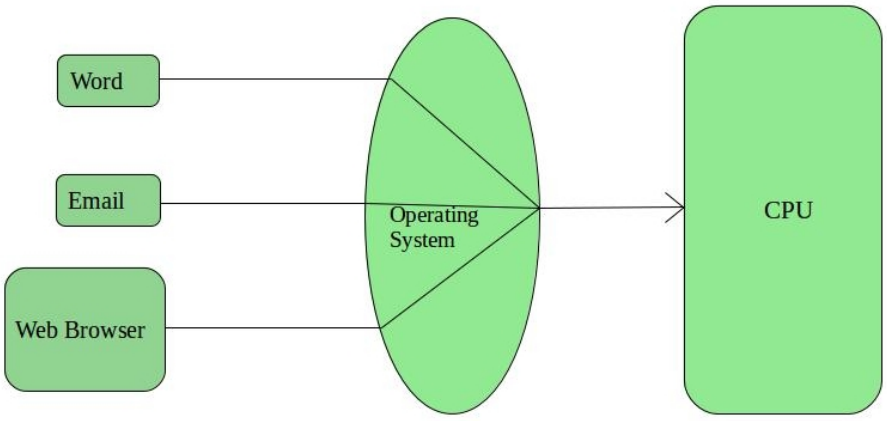
It is sometime costly

The other jobs will have to wait for an unknown time if any job fails

Examples of Batch based Operating System: Payroll System, Bank Statements etc.

2. Time-Sharing Operating Systems

Each task is given some time to execute, so that all the tasks work smoothly. Each user gets time of CPU as they use single system. These systems are also known as Multitasking Systems. The task can be from single user or from different users also. The time that each task gets to execute is called quantum. After this time interval is over OS switches over to next task.



Advantages of Time-Sharing OS:

Each task gets an equal opportunity Less chances of duplication of software CPU idle time can be reduced

Disadvantages of Time-Sharing OS:

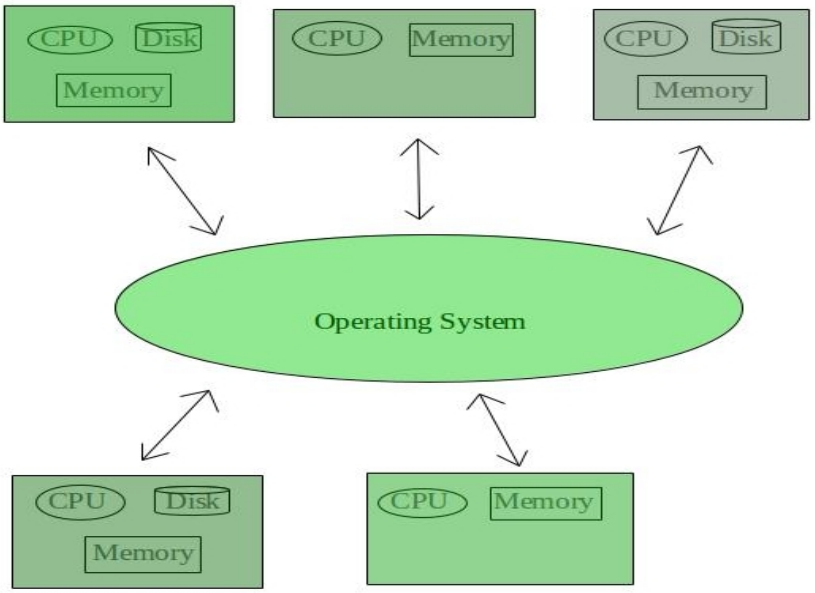
Reliability problem

One must have to take care of security and integrity of user programs and data Data communication problem

Examples of Time-Sharing OSs are: Linux, Unix etc.

3. Distributed Operating System

Various autonomous interconnected computers communicate each other using a shared communication network. Independent systems possess their own memory unit and CPU. These are referred as **loosely coupled systems** or distributed systems. These system's processors differ in size and function. The major benefit of working with these types of operating system is that it is always possible that one user can access the files or software which are not actually present on his system but on some other system connected within this network i.e., remote access is enabled within the devices connected in that network.



Advantages of Distributed Operating System:

Failure of one will not affect the other network communication, as all systems are independent from each other Electronic mail increases the data exchange speed

Since resources are being shared, computation is highly fast and durable

Load on host computer reduces

These systems are easily scalable as many systems can be easily added to the network Delay in data processing reduces

Disadvantages of Distributed Operating System:

Failure of the main network will stop the entire communication

To establish distributed systems the language which are used are not well defined yet

These types of systems are not readily available as they are very expensive. Not only that the underlying software is highly complex and not understood well yet

Examples of Distributed Operating System are- LOCUS .

4. Real-Time Operating System

These types of OSs are used in real-time systems. The time interval required to process and respond to inputs is very small. This time interval is called **response time**.

Real-time systems are used when there are time requirements are very strict like missile systems, air traffic control systems, robots etc.

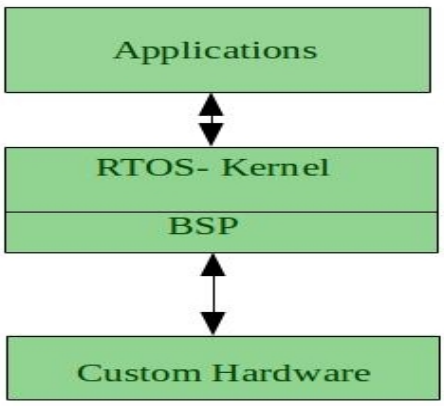
Two types of Real-Time Operating System which are as follows:

Hard Real-Time Systems:

These OSs are meant for the applications where time constraints are very strict and even the shortest possible delay is not acceptable. These systems are built for saving life like automatic parachutes or air bags which are required to be readily available in case of any accident. Virtual memory is almost never found in these systems.

Soft Real-Time Systems:

These OSs are for applications where for time-constraint is less strict.



Advantages of RTOS:

Maximum Consumption: Maximum utilization of devices and system,thus more output from all the resources

Task Shifting: Time assigned for shifting tasks in these systems are very less. For example in older systems it takes about 10 micro seconds in shifting one task to another and in latest systems it takes 3 micro seconds.

Focus on Application: Focus on running applications and less importance to applications which are in queue.

Real time operating system in embedded system: Since size of programs are small, RTOS can also be used in embedded systems like in transport and others.

Error Free: These types of systems MUST be able to deal with any exceptions, so they are not really error free, but handle error conditions without halting the system.

Memory Allocation: Memory allocation is best managed in these type of systems.

Disadvantages of RTOS:

Limited Tasks: Very few tasks run at the same time and their concentration is very less on few applications to avoid errors. **Use heavy system resources:** Sometimes the system resources are not so good and they are expensive as well.

Complex Algorithms: The algorithms are very complex and difficult for the designer to write on.

Device driver and interrupt signals: It needs specific device drivers and interrupt signals to response earliest to interrupts. **Thread Priority:** It is not good to set thread priority as these systems are very less prone to switching tasks.

Examples of Real-Time Operating Systems are: Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

2.5 Interrupts

An interrupt is a signal emitted by hardware or software when a process or an event needs immediate attention. It alerts the processor to a high priority process requiring interruption of the current working process. In I/O devices one of the bus control lines is dedicated for this purpose and is called the *Interrupt Service Routine (ISR)*.

When a device raises an interrupt at lets say process i, the processor first completes the execution of instruction i. Then it loads the Program Counter (PC) with the address of the first instruction of

the ISR. Before loading the Program Counter with the address, the address of the interrupted instruction is moved to a temporary location. Therefore, after handling the interrupt the processor can continue with process i+1.

While the processor is handling the interrupts, it must inform the device that its request has been recognized so that it stops sending the interrupt request signal. Also, saving the registers so that the interrupted process can be restored in the future, increases the delay between the time an interrupt is received and the start of the execution of the ISR. This is called Interrupt Latency.

Hardware Interrupts:

In a hardware interrupt, all the devices are connected to the Interrupt Request Line. A single request line is used for all the n devices. To request an interrupt, a device closes its associated switch. When a device requests an interrupts, the value of INTR is the logical OR of the requests from individual devices.

Sequence of events involved in handling an IRQ:

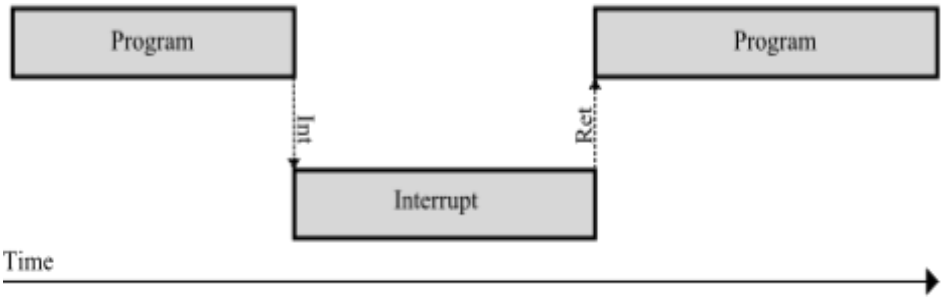
1. Devices raise an IRQ.

2. Processor interrupts the program currently being executed.

3. Device is informed that its request has been recognized and the device deactivates the request signal. 4. The requested action is performed.

5. Interrupt is enabled and the interrupted program is resumed.

Conceptually an interrupt causes the following to happen:



The grey bars represent the control flow. The top line is the program that is currently running, and the bottom bar is the interrupt service routine (ISR). Notice that when the interrupt (**Int**) occurs, the program stops executing and the microcontroller begins to execute the ISR. Once the ISR is complete, the microcontroller returns to processing the program where it left off.

Handling Multiple Devices:

When more than one device raises an interrupt request signal, then additional information is needed to decide which which device to be considered first. The following methods are used to decide which device to select: Polling, Vectored Interrupts, and Interrupt Nesting. These are explained as following below.

1. **Polling:**

In polling, the first device encountered with with IRQ bit set is the device that is to be serviced first. Appropriate ISR is called to service the same. It is easy to implement but a lot of time is wasted by interrogating the IRQ bit of all devices.

2. **Vectored Interrupts:**

In vectored interrupts, a device requesting an interrupt identifies itself directly by sending a special code to the processor over the bus. This enables the processor to identify the device that generated the interrupt. The special code can be the starting address of the ISR or where the ISR is located in memory, and is called the interrupt vector.

3. **Interrupt Nesting:**

In this method, I/O device is organized in a priority structure. Therefore, interrupt request from a higher priority device is

recognized where as request from a lower priority device is not. To implement this each process/device (even the processor). Processor accepts interrupts only from devices/processes having priority more than it.

What happens when external hardware requests another interrupt while the processor is already in the middle of executing the ISR for a previous interrupt request?

When the first interrupt was requested, hardware in the processor causes it to finish the current instruction, disable further interrupts, and jump to the interrupt handler.

The processor ignores further interrupts until it gets to the part of the interrupt handler that has the "return from interrupt" instruction, which re-enables interrupts.

If an interrupt request occurs while interrupts were turned off, some processors will immediately jump to that interrupt handler as soon as interrupts are turned back on. With this sort of processor, an interrupt storm "starves" the main loop background task. Other processors execute at least one instruction of the main loop before handling the interrupt, so the main loop may execute extremely slowly, but at least it never "starves".

A few processors have an interrupt controller that supports "round robin scheduling", which can be used to prevent a different kind of "starvation" of low-priority interrupt handlers.
Processors priority is encoded in a few bits of PS (Process Status register). It can be changed by program instructions that write into the PS. Processor is in supervised mode only while executing OS routines. It switches to user mode before executing application programs.

EXERCISES

1. List five services provided by an operating system that are designed to make it more convenient for users to use the computer system. In what cases it would be impossible for user-level programs to provide these services?

2. In a multiprogramming and time-sharing environment, several users share the system simultaneously. This situation can result in various security problems.
   a. What are two such problems?
   b. Can we ensure the same degree of security in a time-shared machine as in a dedicated machine?

3. Which of the functionalities listed below need to be supported by the operating system for the following two settings: (a) handheld devices and (b) real-time systems.
   a. Batch programming
   b. Virtual memory
   c. Time sharing

4. Describe the differences between symmetric and asymmetric multiprocessing. What are three advantages and one disadvantage of multiprocessor systems?

5. What is the purpose of interrupts? What are the differences between a trap and an interrupt? Can traps be generated intentionally by a user program? If so, for what purpose?

Chapter 3

Process Description & Control

While a computer program is a passive collection of instructions, a process is the actual execution of those instructions. Several processes may be associated with the same program; for example, opening up several instances of the same program often results in more than one process being executed.

3.1 Definition

In computing, a process is the instance of a computer program that is being executed by one or many threads. It contains the program code and its activity. Depending on the operating system (OS), a process may be made up of multiple threads of execution that execute instructions concurrently.

Multitasking is a method to allow multiple processes to share processors (CPUs) and other system resources. Each CPU (core) executes a single task at a time. However, multitasking allows each processor to switch between tasks that are being executed without having to wait for each task to finish (preemption). Depending on the operating system implementation, switches could be performed when tasks initiate and wait for completion of input/output operations, when a task voluntarily yields the CPU, on hardware interrupts, and when the operating system scheduler decides that a process has expired its fair share of CPU time (e.g, by the Completely Fair Scheduler of the Linux kernel).

A common form of multitasking is provided by CPU's time-sharing that is a method for interleaving the execution of users processes and threads, and even of independent kernel tasks - although the latter feature is feasible only in preemptive kernels such as Linux. Preemption has an important side effect for interactive process that are given higher priority with respect to CPU bound processes, therefore users are immediately assigned computing resources at the simple pressing of a key or when moving a mouse. Furthermore, applications like video and music reproduction are given some kind of real-time priority, preempting any other lower priority process. In time-sharing systems, context switches are performed rapidly, which makes it seem like multiple processes are being executed simultaneously on the same processor. This simultaneous execution of multiple processes is called concurrency.

For security and reliability, most modern operating systems prevent direct communication between independent processes, providing strictly mediated and controlled inter-process communication functionality.

A child process in computing is a process created by another process (the parent process). This technique pertains to multitasking operating systems, and is sometimes called a subprocess or traditionally a subtask.

A child process inherits most of its attributes (described above), such as file descriptors, from its parent. In Linux, a child process is typically created as a copy of the parent. The child process can then overlay itself with a different program as required.

Each process may create many child processes but will have at most one parent process; if a process does not have a parent this usually indicates that it was created directly by the kernel. In some systems, including Linux-based systems, the very first process is started by the kernel at booting time and never terminates; other parentless processes may be launched to carry out various tasks

in userspace. Another way for a process to end up without a parent is if its parent dies, leaving an orphan process; but in this case it will shortly be adopted by the main process.

In general, a computer system process consists of (or is said to own) the following resources:

Operating system descriptors of resources that are allocated to the process, such as file descriptors (Unix terminology) or handles (Windows), and data sources and sinks.
Security attributes, such as the process owner and the process' set of permissions (allowable operations).

Processor state (context), such as the content of registers and physical memory addressing. The state is typically stored in computer registers when the process is executing, and in memory otherwise.

The operating system holds most of this information about active processes in data structures called process control blocks. Any subset of the resources, typically at least the processor state, may be associated with each of the process' threads in operating systems that support threads or child processes.

The operating system keeps its processes separate and allocates the resources they need, so that they are less likely to interfere with each other and cause system failures (e.g., deadlock or thrashing). The operating system may also provide mechanisms for inter- process communication to enable processes to interact in safe and predictable ways.

3.2 Attributes of a Process (Process Context)

A process has following attributes:

1. Process Id: A unique identifier assigned by the operating system
2. Process State: Can be ready, running, etc.
3. CPU registers: Like the Program Counter (CPU registers must be saved and restored)
4. I/O status information: For example, devices allocated to the process, open files, etc.
5. CPU scheduling information: For example, Priority (Different processes may have different times)
6. Various accounting information

Context Switching

The process of saving the context of one process and loading the context of another process is known as Context Switching. In simple terms, it is like loading and unloading the process from running state to ready state. The cases of occurring context switching:

1. When a high-priority process comes to ready state (i.e. with higher priority than the running process)
2. An Interrupt occurs
3. User and kernel mode switch (It is not necessary though)
4. Preemptive CPU scheduling used.

A mode switch occurs when CPU privilege level is changed, for example when a system call is made or a fault occurs. The kernel works in more a privileged mode than a standard user task. If a user process wants to access things which are only accessible to the kernel, a mode switch must occur. The currently executing process need not be changed during a mode switch. A mode switch typically occurs for a process context switch to occur. Only the kernel can cause a context switch.

3.3 Process States

In a multitasking computer system, processes may occupy a variety of states. These distinct states may not be recognized as such by the operating system kernel. However, they are a useful abstraction for the understanding of processes.

If we look at the following diagram there are several things to note: Memory:

We have some states of a process where the process is kept in main memory - RAM. We have some states that are stored in secondary memory - that is what we call swap space and is actually part of the hard disk set aside for use by the operating system.

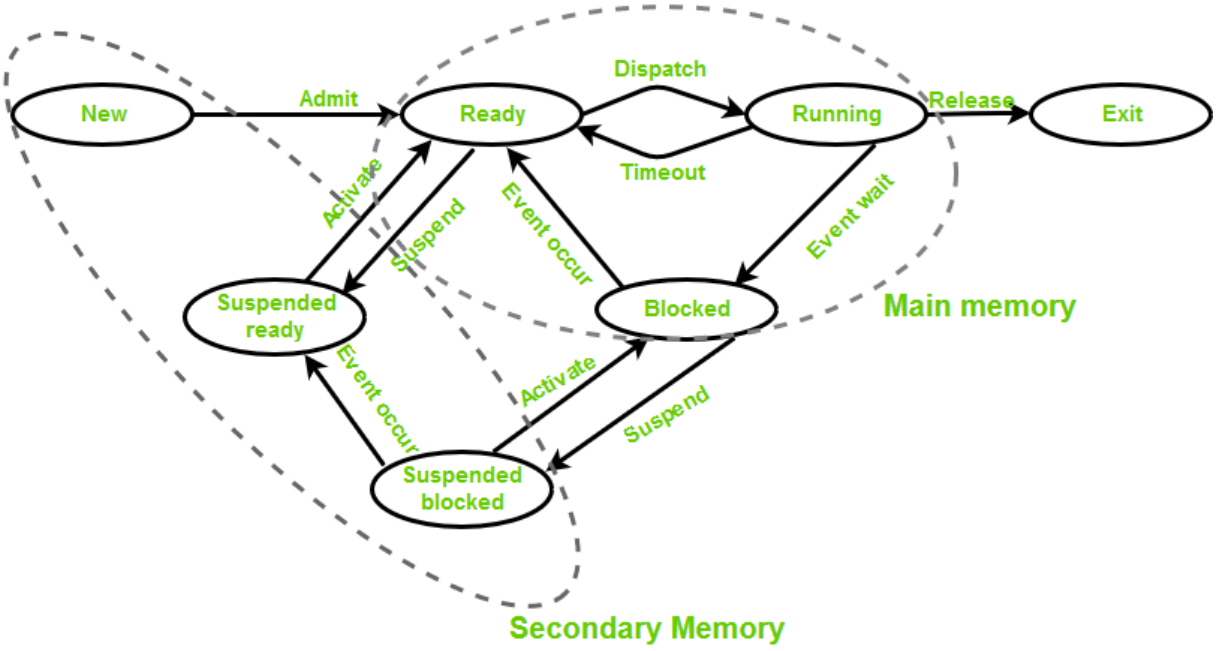
There are numerous actions: Admint, dispatch, Event wait, Event occur, Suspend, Activate, Timeout, and Release. These all have an impact on the process during its lifetime. In the following diagram there are 7 states. Depending on who the author is there may be 5, 6 or 7. Sometimes the 2 Suspended states are not shown, but we shown them for clarity here.

Let's follow a process through a lifecycle.

1. A new process gets created. For example, a user opens up a word processor, this requires a new process.
2. Once the process has completed its initialization, it is placed in a Ready state with all of the other processes waiting to take its
   turn on the processor.
3. When the process's turn comes, it is dispatched to the Running state and executes on the processor. From here one of 3 things
   happens:
   1. the process completes and is Released and moves to the Exit state
   2. it uses up it turn and so Timeout and is returned to the Ready state
   3. some event happens, such as waiting for user input, and it is moved to the Blocked state.
      1. In the Blocked state, once the event it is waiting on occurs, it can return to the Ready state.
      2. If the event doesn't occur for an extended period of time, the process can get moved to the Suspended blocked state.
      Since it is suspended it is now in virtual memory - which is actually disk space set aside for temporary storage.
4. Once the event this process is waiting on does occur it is moved to the Suspended ready state, then waits to get moved from secondary storage, into main memory in the Ready state.
   1. On very busy systems processes can get moved from the Ready state to the Suspended ready state as well.

5. Eventually every process will Exit.

The following typical process states are possible on computer systems of all kinds. In most of these states, processes are "stored" on main memory.



New or Created

When a process is first created, it occupies the "created" or "new" state. In this state, the process awaits admission to the "ready" state. Admission will be approved or delayed by a long-term, or admission, scheduler. Typically in most desktop computer systems, this admission will be approved automatically. However, for real-time operating systems this admission may be delayed. In a realtime system, admitting too many processes to the "ready" state may lead to oversaturation and overcontention of the system's resources, leading to an inability to meet process deadlines.

Operating systems need some ways to create processes. In a very simple system designed for running only a single application (e.g., the controller in a microwave oven), it may be possible to have all the processes that will ever be needed be present when the system comes up. In general-purpose systems, however, some way is needed to create and terminate processes as needed during operation.

There are four principal events that cause a process to be created:

1. System initialization.
2. Execution of process creation system call by a running process.

3. A user request to create a new process.
4. Initiation of a batch job.

When an operating system is booted, typically several processes are created. Some of these are foreground processes, that interact with a (human) user and perform work for them. Others are background processes, which are not associated with particular users, but instead have some specific function. For example, one background process may be designed to accept incoming e-mails, sleeping most of the day but suddenly springing to life when an incoming e-mail arrives. Another background process may be designed to accept an incoming request for web pages hosted on the machine, waking up when a request arrives to service that request.

There are several steps involved in process creation. The first step is the validation of whether the parent process has sufficient authorization to create a process. Upon successful validation, the parent process is copied almost entirely, with changes only to the unique process id, parent process, and user-space. Each new process gets its own user space.

Ready

A "ready" or "waiting" process has been loaded into main memory and is awaiting execution on a CPU (to be context switched onto the CPU by the dispatcher, or short-term scheduler). There may be many "ready" processes at any one point of the system's execution—for example, in a one-processor system, only one process can be executing at any one time, and all other "concurrently executing" processes will be waiting for execution.

A ready queue or run queue is used in computer scheduling. Modern computers are capable of running many different programs or processes at the same time. However, the CPU is only capable of handling one process at a time. Processes that are ready for the CPU are kept in a queue for "ready" processes. Other processes that are waiting for an event to occur, such as loading information from a hard drive or waiting on an internet connection, are not in the ready queue.

Running

A process moves into the running state when it is chosen for execution. The process's instructions are executed by one of the CPUs (or cores) of the system. There is at most one running process per CPU or core. A process can run in either of the two modes, namely kernel mode or user mode.

Blocked

A process transitions to a blocked state when it is waiting for some event, such as a resource becoming available or the completion of an I/O operation. In a multitasking computer system,

individual processes, must share the resources of the system. Shared resources include: the CPU, network and network interfaces, memory and disk.

For example, a process may block on a call to an I/O device such as a printer, if the printer is not available. Processes also commonly block when they require user input, or require access to a critical section which must be executed atomically.

Suspend ready

Process that was initially in the ready state but were swapped out of main memory(refer Virtual Memory topic) and placed onto external storage by scheduler are said to be in suspend ready state. The process will transition back to ready state whenever the process is again brought onto the main memory.

Suspend wait or suspend blocked

Similar to suspend ready but uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory. When work is finished it may go to suspend ready.

Terminated

A process may be terminated, either from the "running" state by completing its execution or by explicitly being killed. In either of these cases, the process moves to the "terminated" state. The underlying program is no longer executing, but the process remains in the process table as a zombie process until its parent process calls the wait system call to read its exit status, at which point the process is removed from the process table, finally ending the process's lifetime. If the parent fails to call wait, this continues to consume the process table entry (concretely the process identifier or PID), and causes a resource leak.

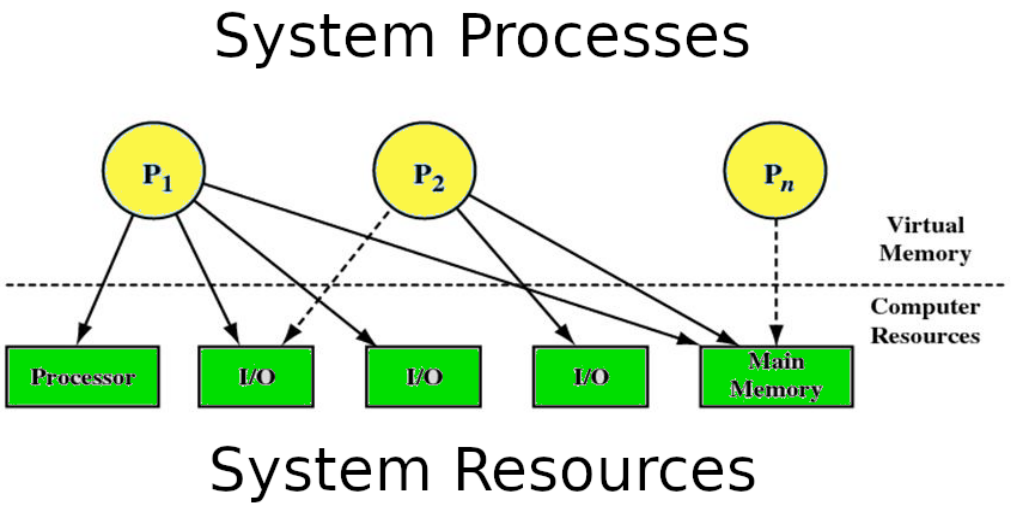
There are many reasons for process termination:

- Batch job issues halt instruction
- User logs off
- Process executes a service request to terminate
- Error and fault conditions
- Normal completion
- Time limit exceeded
- Memory unavailable

- Bounds violation; for example: attempted access of (non-existent) 11th element of a 10-element array.
- Protection error; for example: attempted write to read-only file
- Arithmetic error; for example: attempted division by zero
- Time overrun; for example: process waited longer than a specified maximum for an event I/O failure
- Invalid instruction; for example: when a process tries to execute data (text)
- Privileged instruction
- Data misuse
- Operating system intervention; for example: to resolve a deadlock
- Parent terminates so child processes terminate (cascading termination) Parent request

3.4 System Processes / System Resources

In a multiprocessing system, there are numerous processes competing to the system's resources. As each process takes its turn executing in the processor, the state of the other processes must be kept in the state that they were interrupted at so as to restore the next process to execute. Processes run, make use of I/O resources, consume memory. At times processes block waiting for I/O, allowing other processes to run on the processors. Some processes are swapped out in order to make room in physical memory for other processes' needs. P_1 is running, accessing I/O and memory. P_2 is blocked waiting for P1 to complete I/O. P_n is swapped out waiting to return to main memory and further processing.



3.5 Process Management and Control

As the operating system manages processes and resources, it must maintain information about the current status of each process and the resources in use. The approach to maintaining this information is for the operating system to construct and maintain various tables of information about each entity to be managed.

Operating system maintains four internal components: 1) memory; 2) devices; 3) files; and 4) processes. Details differ from one operating system to another, but all operating systems maintain information in these four categories.

Memory tables: Memory is central to the operation of a modern computer system. Memory is a large array of words or bytes, each with its own address. Interaction is achieved through a sequence of reads or writes of specific memory address. The CPU fetches from and stores in memory.

In order to improve both the utilization of CPU and the speed of the computer's response to its users, several processes must be kept in memory. There are many different algorithms depends on the particular situation. Selection of a memory management scheme for a specific system depends upon many factor, but especially upon the hardware design of the system. Each algorithm requires its own hardware support.

The operating system is responsible for the following activities in connection with memory management.

Keep track of which parts of memory are currently being used and by whom. Decide which processes are to be loaded into memory when memory space becomes available. Allocate and deallocate memory space as needed.

Device tables: One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. For example, in Linux, the peculiarities of I/O devices are hidden from the bulk of the operating system itself by the I/O system. The I/O system consists of:

- A buffer caching system
- A general device driver code
- Drivers for specific hardware devices.

Only the device driver knows the peculiarities of a specific device. We will cover the details later in this course

File tables: File management is one of the most visible services of an operating system. Computers can store information in several different physical forms; disk - both magnetic disks and newer SSD devices, various USB devices, and to the cloud. Each of these devices has it own characteristics and physical organization.

For convenient use of the computer system, the operating system provides a uniform logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file. Files are mapped, by the operating system, onto physical devices.

A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic or alphanumeric. Files may be free-form, such as text files, or may be rigidly formatted. In general a files is a sequence of bits, bytes, lines or records whose meaning is defined by its creator and user. It is a very general concept.

The operating system implements the abstract concept of the file by managing mass storage device, such as types and disks. Also files are normally organized into directories to ease their use. Finally, when multiple users have access to files, it may be desirable to control by whom and in what ways files may be accessed.

The operating system is responsible for the following activities in connection with file management:

- The creation and deletion of files

- The creation and deletion of directory
- The support of primitives for manipulating files and directories The mapping of files onto disk storage.
- Backup of files on stable (non volatile) storage.

This portion of the operating system will also be dealt with later.

Processes: The CPU executes a large number of programs. While its main concern is the execution of user programs, the CPU is also needed for other system activities. These activities are called processes. A process is a program in execution. Typically, a batch job is a process. A time-shared user program is a process. A system task, such as spooling, is also a process. For now, a process may be considered as a job or a time-shared program, but the concept is actually more general.

In general, a process will need certain resources such as CPU time, memory, files, I/O devices, etc., to accomplish its task. These resources are given to the process when it is created. In addition to the various physical and logical resources that a process obtains when its is created, some initialization data (input) may be passed along. For example, a process whose function is to display on the screen of a terminal the status of a file, say F1, will get as an input the name of the file F1 and execute the appropriate program to obtain the desired information.

We emphasize that a program by itself is not a process; a program is a passive entity, while a process is an active entity. It is known that two processes may be associated with the same program, they are nevertheless considered two separate execution sequences.

A process is the unit of work in a system. Such a system consists of a collection of processes, some of which are operating system processes, those that execute system code, and the rest being user processes, those that execute user code. All of those processes can potentially execute concurrently.

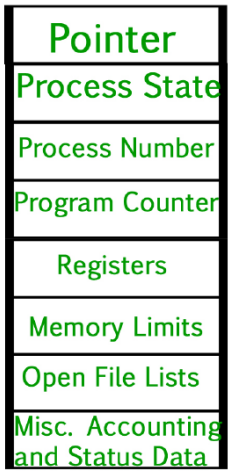
The operating system is responsible for the following activities in connection with processes managed.

Process Table and Process Control Block (PCB)

While creating a process the operating system performs several operations. To identify the processes, it assigns a process identification number (PID) to each process. As the operating system supports multi-programming, it needs to keep track of all the processes. For this task, the process control block (PCB) is used to track the process's execution status. Each block of memory contains information about the process state, program counter, stack pointer, status of opened files,

scheduling algorithms, etc. All these information is required and must be saved when the process is switched from one state to another. When the process makes a transition from one state to another, the operating system must update information in the process's PCB.

A process control block (PCB) contains information about the process, i.e. registers, quantum, priority, etc. The process table is an array of PCB's, that means logically contains a PCB for all of the current processes in the system.



Process scheduling state –The state of the process in terms of "ready", "suspended", etc., and other scheduling information as well, such as priority value, the amount of time elapsed since the process gained control of the CPU or since it was suspended. Also, in case of a suspended process, event identification data must be recorded for the event the process is waiting for. **Process structuring information** – the process's children id's, or the id's of other processes related to the current one in some functional way, which may be represented as a queue, a ring or other data structures

Pointer – It is a stack pointer which is required to be saved when the process is switched from one state to another to retain the current position of the process.

Process number – Every process is assigned with a unique id known as process ID or PID which stores the process identifier.

Program counter – It stores the counter which contains the address of the next instruction that is to be executed for the process.

Register – These are the CPU registers which includes: accumulator, base, registers and general purpose registers.

Memory Management Information – This field contains the information about memory management system used by operating system. This may include the page tables, segment tables etc.

Open files list – This information includes the list of files opened for a process.

Interprocess communication information – flags, signals and messages associated with the communication among independent processes

Process Privileges – allowed/disallowed access to system resources

Process State – new, ready, running, waiting, dead

Process Number (PID) – unique identification number for each process (also known as Process ID).

Program Counter (PC) – A pointer to the address of the next instruction to be executed for this process.

CPU Scheduling Information – information scheduling CPU time

Accounting Information – amount of CPU used for process execution, time limits, execution ID etc.

I/O Status Information – list of I/O devices allocated to the process.

3.6 kernel & System Call

The kernel is a computer program at the core of a computer's operating system that has complete control over everything in the system. It is the "portion of the operating system code that is always resident in memory", and facilitates interactions between hardware and software components. On most systems, the kernel is one of the first programs loaded on startup (after the bootloader). It handles the rest of startup as well as memory, peripherals, and input/output (I/O) requests from software, translating them into data-processing instructions for the central processing unit.

In computing, a **system call** when a program program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to interact with the operating system. Application programs are NOT allowed to perform certain tasks, such as open a file, or create a new process. System calls provide the services of the operating system to the application programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes, that is the applications that users are running on the system, to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

System Calls provide many services such as

1. Process creation and management

2. Main memory management

3. File Access, Directory and File system management 4. Device handling(I/O)

5. Protection

6. Networking, etc.

Example:

If a user is running a word processing tool, and wants to save the document - the word processor asks the operating system to create a file, or open a file, to save the current set of changes. If the application has permission to write to the requested file then the operating system performs the task. Otherwise, the operating system returns a status telling the user they do not have permission to write to the requested file. This concept of user versus kernel allows the operating system to maintain a certain level of control.

3.7 OS Execution Modes

An error in one program can adversely affect many processes, it might modify data of another program, or also can affect the operating system. For example, if a process stuck in infinite loop then this infinite loop could affect correct operation of other processes. So to ensure the proper execution of the operating system, there are two modes of operation:

User mode –

When the computer system run user applications like creating a text document or using any application program, then the system is in the user mode. When the user application requests for a service from the operating system or an interrupt occurs or **system call**, then there will be a transition from user to kernel mode to fulfill the requests.

Given below image describes what happen when an interrupt occurs: (do not worry about the comments about the mode bit)

Kernel Mode –

While the system is running the certain processes operate in kernel mode because the processes needs access to operating system calls. This provides protection by controlling which processes can access kernel mode operations. As shown in the diagram above, the system will allow certain user mode processes to execute system calls by allowing the process to temporarily run in kernel mode. While in kernel mode the process is allowed to have direct access to all hardware and

memory in the system (also called privileged mode). If a user process attempts to run privileged instructions in user mode then it will treat instruction as illegal and traps to OS. Some of the privileged instructions are:

1. Handling system interrupts
2. To switch from user mode to kernel mode.
3. Management of I/O devices.

EXERCISES

1. The following state transition table is a simplified model of process management with the labels representing transitions between states of READY, RUN, BLOCKED, and NONRESIDENT.

	READY	RUN	BLOCKED	NONRESIDENT
READY	-	1	-	5
RUN	2	-	3	-
BLOCKED	4	-	-	6

Draw the diagram and give an example of each event of the above transitions.

2. Suppose that we have a multi-programmed computer in which each job has identical characteristics. In one computation period T for a job, half the time is spent in I/O and the other half in processor activity. Each job runs for a total of N periods. Assume that a simple time-sharing scheduling is used and that I/O operations can overlap with processor operation. Considering these quantities:

Turnaround Time: actual time to complete a job.

Throughput: average number of jobs completed per time period T.

Processor Utilization: percentage of time that the processor is active and not waiting.

Compute these quantities for one, two, and four simultaneous jobs assuming that the period T is distributed in each of the following ways:

a. I/O first half, processor second Half.
b. I/O first and fourth quarters, processor second and third quarters.

3. Consider the following figure and assume that at time 5 no system resources are being used except for the processor and memory, now consider the following events:-

At time 5: P1 executes a command to read from disk unit 3.
At time 15: P5's time slice expires.
At time 18: P7 executes a command to write to disk unit 3.
At time 20: P3 executes a command to read from disk unit 2.
At time 24: P5 executes a command to write from disk unit 3.
At time 28: P5 is swapped out.
At time 33: An interrupt occurs from disk unit 2: P3's read is complete.
At time 36: An interrupt occurs from disk unit 3: P1's read is complete.
At time 38: P8 terminates.
At time 40: An interrupt occurs from disk unit 3: P5's write is complete.
At time 44: P5 is swapped back in.
At time 48: An interrupt occurs from disk unit 3: P7's write is complete

For each time 22, 37 and 47 identify which state each process is in. if a process is blocked further identify the event on which is it blocked.

Chapter 4

Process Scheduling

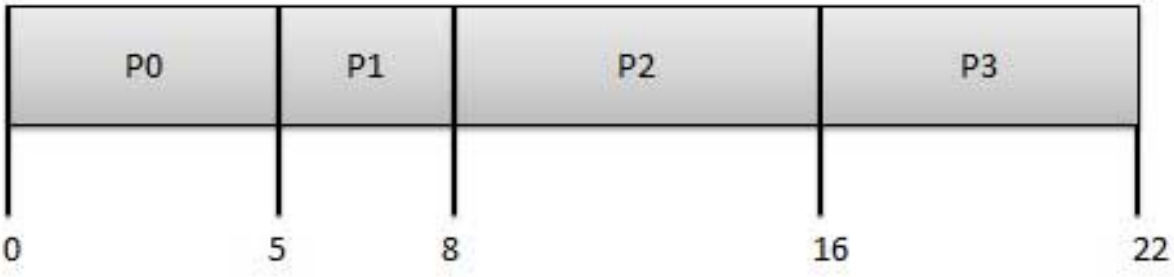
Process Scheduling Algorithms

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are popular process scheduling algorithms which we are going to discuss. These algorithms are either non-preemptive or preemptive. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

4.1 First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

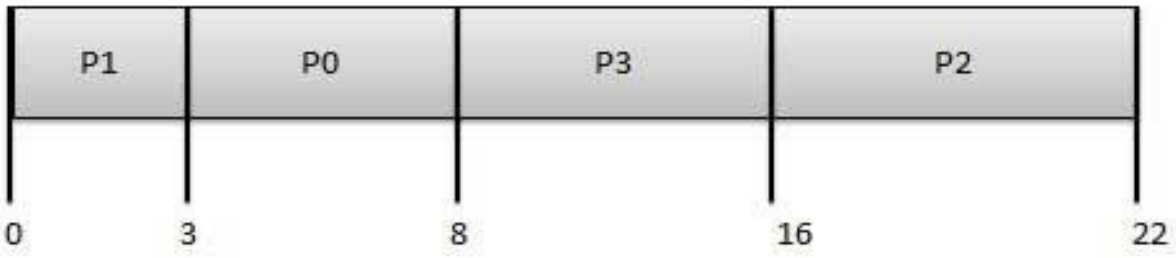
4.2 Shortest Job First (SJF)

- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Given: Table of processes, and their Arrival time, Execution time

Process	Arrival Time	Execution Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	14
P3	3	6	8

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Waiting time of each process is as follows −

Process	Waiting Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	14 - 2 = 12
P3	8 - 3 = 5

Average Wait Time: (0 + 4 + 12 + 5)/4 = 21 / 4 = 5.25

4.3 Priority Based Scheduling

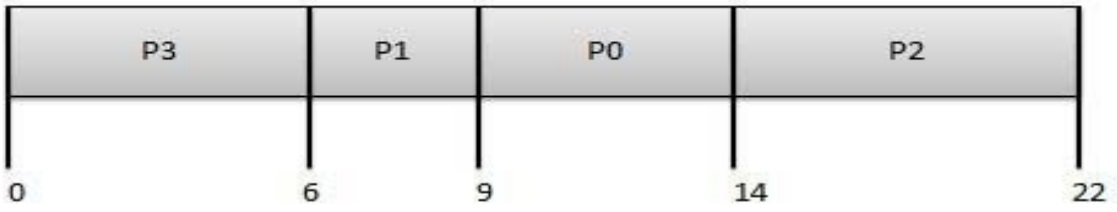
- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.

- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Given: Table of processes, and their Arrival time, Execution time, and priority. Here we are considering 1 is the lowest priority.

Process	Arrival Time	Execution Time	Priority	Service Time
P0	0	5	1	0
P1	1	3	2	11
P2	2	8	1	14
P3	3	6	3	5

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Waiting time of each process is as follows −

Process	Waiting Time
P0	0 - 0 = 0
P1	11 - 1 = 10
P2	14 - 2 = 12

P3	5 - 3 = 2

Average Wait Time: (0 + 10 + 12 + 2)/4 = 24 / 4 = 6

4.4 Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.

Given: Table of arrival time and burst time for five processes P1, P2, P3, P4 and P5.

Process	Burst Time	Arrival Time
P1	6 ms	2 ms
P2	2 ms	5 ms
P3	8 ms	1 ms
P4	3 ms	0 ms
P5	4 ms	4 ms

The Shortest Job First CPU Scheduling Algorithm will work on the basis of steps as mentioned below:

At time = 0,

- Process P4 arrives and starts executing

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
0-1ms	P4	0ms		1ms	3ms	2ms

The Shortest Job First CPU Scheduling Algorithm will work on the basis of steps as mentioned below:

At time = 0,

- Process P4 arrives and starts executing

At time= 1,

- Process P3 arrives.
- But, as P4 has a shorter burst time. It will continue execution.
- Thus, P3 will wait till P4 gets executed.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
1-2ms	P4	0ms	P3	1ms	2ms	1ms
	P3	1ms		0ms	8ms	8ms

At time =2,

- Process P1 arrives with burst time = 6
- As the burst time of P1 is more than that of P4
- Thus, P4 will continue its execution.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
2-3ms	~~P4~~	~~0ms~~	P3, P1	~~1ms~~	~~1ms~~	~~0ms~~
	P3	1ms		0ms	8ms	8ms
	P1	2ms		0ms	6ms	6ms

At time = 3,

- Process P4 will finish its execution.
- Then, the burst time of P3 and P1 is compared.
- Process P1 is executed because its burst time is less as compared to P3.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
3-4ms	P3	1ms	P3	0ms	8ms	8ms
	P1	2ms		1ms	6ms	5ms

At time = 4,

- Process P5 arrives.
- Then the burst time of P3, P5, and P1 is compared.
- Process P5 gets executed first among them because its burst time is lowest, and process P1 is preempted.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
4-5ms	P3	1ms	P3, P1	0ms	8ms	8ms
	P1	2ms		0ms	5ms	5ms
	P5	4ms		1ms	4ms	3ms

At time = 5,

- Process P2 arrives.
- The burst time of all processes are compared,
- Process P2 gets executed as its burst time is lowest among all.
- Process P5 is preempted.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
5-6ms	P3	1ms	P3, P5, P1	0ms	8ms	8ms
	P1	2ms		0ms	5ms	5ms
	P5	4ms		0ms	3ms	3ms
	P2	5ms		1ms	2ms	1ms

At time = 6,

- Process P2 will keep executing.
- It will execute till time = 8 as the burst time of P2 is 2ms

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
6-7ms	P3	1ms	P3, P5, P1	0ms	8ms	8ms
	P1	2ms		0ms	5ms	5ms
	P5	4ms		0ms	3ms	3ms
	~~P2~~	~~5ms~~		~~1ms~~	~~1ms~~	~~0ms~~

At time=7,

- The process P2 finishes its execution.
- Then again the burst time of all remaining processes is compared.
- The Process P5 gets executed because its burst time is lesser than the others.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
7-10ms	P3	1ms	P3, P1	0ms	8ms	8ms
	P1	2ms		0ms	5ms	5ms
	~~P5~~	~~4ms~~		~~3ms~~	~~3ms~~	~~0ms~~

At time = 10,

- The process P5 will finish its execution.
- Then the burst time of the remaining processes P1 and P3 is compared.

- Thus, process P1 is executed as its burst time is less than P3

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
10-15ms	P3	1ms	P3	0ms	8ms	8ms
	~~P1~~	~~4ms~~		~~4ms~~	~~5ms~~	~~0ms~~

At time = 15,

- The process P1 finishes its execution and P3 is the only process left.
- P3 will start executing.

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
15-23ms	~~P3~~	~~1ms~~		~~8ms~~	~~8ms~~	~~0ms~~

At time = 23,

- Process P3 will finish its execution.

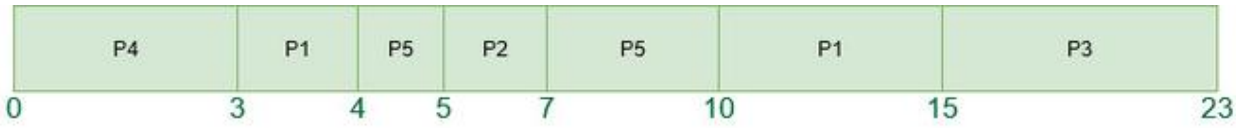
The overall execution of the processes will be as shown below:

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
0-1ms	P4	0ms		1ms	3ms	2ms

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
1-2ms	P4	0ms	P3	1ms	2ms	1ms
	P3	1ms		0ms	8ms	8ms
2-3ms	~~P4~~	~~0ms~~	P3, P1	~~1ms~~	~~1ms~~	~~0ms~~
	P3	1ms		0ms	8ms	8ms
	P1	2ms		0ms	6ms	6ms
3-4ms	P3	1ms	P3	0ms	8ms	8ms
	P1	2ms		1ms	6ms	5ms
4-5ms	P3	1ms	P3, P1	0ms	8ms	8ms
	P1	2ms		0ms	5ms	5ms
	P5	4ms		1ms	4ms	3ms
5-6ms	P3	1ms	P3, P5, P1	0ms	8ms	8ms
	P1	2ms		0ms	5ms	5ms
	P5	4ms		0ms	3ms	3ms

Time Instance	Process	Arrival Time	Waiting Table	Execution Time	Initial Burst Time	Remaining Burst Time
	P2	5ms		1ms	2ms	1ms
6-7ms	P3	1ms	P3, P5, P1	0ms	8ms	8ms
	P1	2ms		0ms	5ms	5ms
	P5	4ms		0ms	3ms	3ms
	~~**P2**~~	~~**5ms**~~		~~**1ms**~~	~~**1ms**~~	~~**0ms**~~
7-10ms	P3	1ms	P3, P1	0ms	8ms	8ms
	P1	2ms		0ms	5ms	5ms
	~~**P5**~~	~~**4ms**~~		~~**3ms**~~	~~**3ms**~~	~~**0ms**~~
10-15ms	P3	1ms	P3	0ms	8ms	8ms
	~~**P1**~~	~~**4ms**~~		~~**4ms**~~	~~**5ms**~~	~~**0ms**~~
15-23ms	~~**P3**~~	~~**1ms**~~		~~**8ms**~~	~~**8ms**~~	~~**0ms**~~

- Gantt chart for above execution:



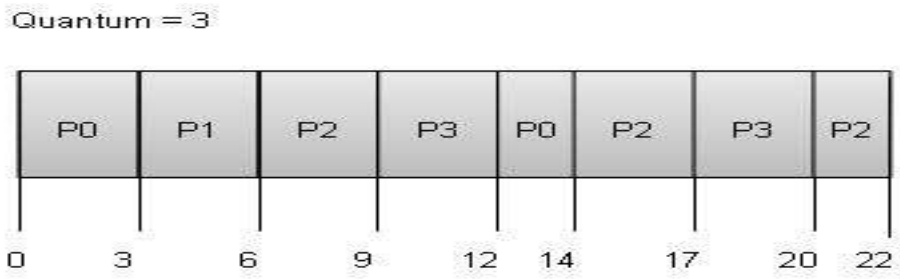
- Turn Around time = Completion time – arrival time
- Waiting Time = Turn around time – burst time

Process	Completion Time	Turn Around Time	Waiting Time
P1	15	15-2 = 13	13-6 = 7
P2	7	7-5 = 2	2-2 = 0
P3	23	23-1 = 22	22-8 = 14
P4	3	3-0 = 3	3-3 = 0
P5	10	10-4 = 6	6-4 = 2

- Average Turn around time = (13 + 2 + 22 + 3 + 6)/5 = 9.2
- Average waiting time = (7 + 0 + 14 + 0 + 2)/5 = 23/5 = 4.6

4.5 Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a quantum.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

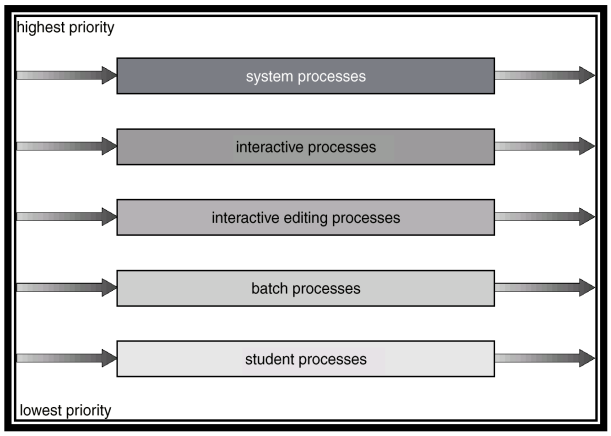
Average Wait Time: (9+2+12+11) / 4 = 8.5

4.6 Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

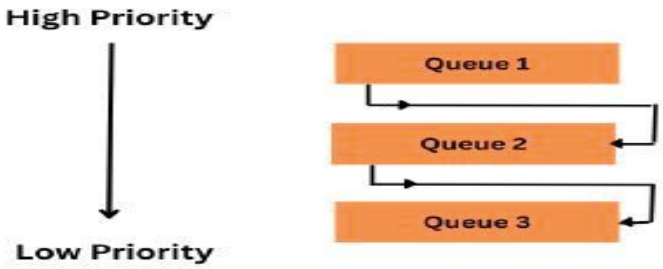


4.7 A Multilevel Feedback Queue (MLFQ)

It uses multiple queues with varying priorities to manage process execution. It dynamically adjusts priorities based on process behavior, promoting or demoting processes between queues. The MLFQ scheduling algorithm efficiently manages tasks, prioritizing responsiveness and resource utilization, which helps reduce starvation. Its dynamic nature is well-suited for systems with various processes, enhancing overall performance and user experience.

he steps involved in MLFQ scheduling when a process enters the system.

1. When a process enters the system, it is initially assigned to the highest priority queue.
2. The process can execute for a specific time quantum in its current queue.
3. If the process completes within the time quantum, it is removed from the system.
4. If the process does not complete within the time quantum, it is demoted to a lower priority queue and given a shorter time quantum.
5. This promotion and demotion process continues based on the behavior of the processes.
6. The high-priority queues take precedence over low-priority queues, allowing the latter processes to run only when high-priority queues are empty.
7. The feedback mechanism allows processes to move between queues based on their execution behavior.
8. The process continues until all processes are executed or terminated.



EXERCISES

For the following problems, draw the Gantt graph and determine the waiting tine and finish time for each process:-

a. Apply the first come first serve scheduling algorithm:-

Process	Arrival Time	Burst Time
P1	0	50
P2	20	20
P3	25	10
P4	40	24

b. Apply the Non-preemptive and preemptive shortest job first scheduling algorithm:-

Process	Arrival Time	Burst Time
P1	0	50
P2	20	20
P3	25	10
P4	40	24

c. Apply the Non-preemptive and preemptive priority scheduling algorithm:-

Process	Arrival	Burst	Priority
P_1	0	6	5
P_2	2	2	3
P_3	3	3	4
P_4	9	3	2
P_5	10	1	1

d. Apply the round robin scheduling algorithm with time quantum = 20 msec:-

Process	Arrival	Burst
P_1	0	53
P_2	25	17
P_3	50	68
P_4	75	24

e. Apply the Non-preemptive and preemptive multilevel queue scheduling algorithm with 2 levels, RR gets priority over FCFS, and RR with Quantum = 10 msec

Process	Arrival	Burst	Queue
P_1	0	12	FCFS
P_2	4	12	RR
P_3	8	8	FCFS
P_4	20	10	RR

f. Apply the Non-preemptive and preemptive multilevel feedback queue scheduling with three levels RR at 8 units, RR at 16 units and FCFS

Process	Arrival	Burst
P_1	0	32
P_2	10	12
P_3	30	10

Chapter 5

Concurrency: Deadlock and Starvation

In computer science, a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system. The implementation of threads and processes differs between operating systems, but in most cases a thread is a component of a process. Multiple threads can exist within one process, executing concurrently and sharing resources such as memory, while different processes do not share these resources. In particular, the threads of a process share its executable code and the values of its dynamically allocated variables and non-thread-local global variables at any given time.

Difference between Process and Thread.

5.1 Thread

Thread is the segment of a process means a process can have multiple threads and these multiple threads are contained within a process. A thread have 3 states: running, ready, and blocked.

Thread takes less time to terminate as compared to process and like process threads do not isolate.

Difference between Process and Thread:

	Process	Thread
1.	Process means any program is in execution.	Thread means segment of a process.
2.	Process takes more time to terminate.	Thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	Process is less efficient in term of communication.	Thread is more efficient in term of communication.
6.	Process consume more resources.	Thread consume less resources.
7.	Process is isolated.	Threads share memory.
8.	Process is called heavy weight process.	Thread is called light weight process.
9.	Process switching uses interface in operating system.	Thread switching does not require to call a operating system and cause an interrupt to the kernel.
10.	If one server process is blocked no other server process can execute until the first process unblocked.	Second thread in the same task could run, while one server thread is blocked.
11.	Process has its own Process Control Block, Stack and Address Space.	Thread has Parents' PCB, its own Thread Control Block and Stack and common Address space.

Multithreading

A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads.

For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

Threads differ from traditional multitasking operating-system processes in several ways:

- processes are typically independent, while threads exist as subsets of a process
- processes carry considerably more state information than threads, whereas multiple threads within a process share process state as well as memory and other resources.
- processes have separate address spaces, whereas threads share their address space.
- processes interact only through system-provided inter-process communication mechanisms.
- context switching between threads in the same process typically occurs faster than context switching between processes.

Benefits of Multithreading in Operating System

The benefits of multi threaded programming can be broken down into four major categories:

1. **Responsiveness –**

Multithreading in an interactive application may allow a program to continue running even if a part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.

In a non multi threaded environment, a server listens to the port for some request and when the request comes, it processes the request and then resume listening to another request. The time taken while processing of request makes other users wait unnecessarily. Instead a better approach would be to pass the request to a worker thread and continue listening to port.

For example, a multi threaded web browser allow user interaction in one thread while an video is being loaded in another thread. So instead of waiting for the whole web-page to load the user can continue viewing some portion of the web-page.

2. **Resource Sharing –**

Processes may share resources only through techniques such as-

Message Passing Shared Memory Such techniques must be explicitly organized by programmer. However, threads share the memory and the resources of the process to which they belong by default.The benefit of sharing code and data is that it allows an application to have several threads of activity within same address space.

3. **Economy –**

Allocating memory and resources for process creation is a costly job in terms of time and space. Since, threads share memory with the process it belongs, it is more economical to create and context switch threads. Generally much more time is consumed in creating and managing processes than in threads.

4. **Scalability –**

The benefits of multi-programming greatly increase in case of multiprocessor architecture, where threads may be running parallel on multiple processors. If there is only one thread then it is not possible to divide the processes into smaller tasks that different processors can perform.

Single threaded process can run only on one processor regardless of how many processors are available.

Multi-threading on a multiple CPU machine increases parallelism.

5.2 Process Synchronization

When we discuss the concept of synchronization, processes are categorized as one of the following two types:

1. **Independent Process** : Execution of one process does not affects the execution of other processes.

2. **Cooperative Process** : Execution of one process affects the execution of other processes. Process synchronization problems are most likely when dealing with cooperative processes because the process resources are shared between the multiple processes/threads.

Race Condition

When more than one processes is executing the same code, accessing the same memory segment or a shared variable there is the possibility that the output or the value of the shared variable is incorrect. This can happen when multiple processes are attempting to alter a memory location, this can create a race condition - where multiple processes have accessed the current value at a memory location, each process has changed that value, and now they need to write the new

back...BUT...each process has a different new value. So, which one is correct? Which one is going to be the new value.

Operating system need to have a process to manage these shared components/memory segments. This is called synchronization, and is a critical concept in operating system.

Usually race conditions occur inside what is known as a critical section of the code. Race conditions can be avoided if the critical section is treated as an atomic instruction, that is an operation that run completely independently of any other processes, making use of software locks or atomic variables which will prevent race conditions. We will take a look at this concept below.

Critical Section Problem

In concurrent programming, concurrent accesses to shared resources can lead to unexpected or erroneous behavior, so parts of the program where the shared resource is accessed need to be protected in ways that avoid the concurrent access. This protected section is the critical section or critical region. It cannot be executed by more than one process at a time. Typically, the critical section accesses a shared resource, such as a data structure, a peripheral device, or a network connection, that would not operate correctly in the context of multiple concurrent accesses.

Different codes or processes may consist of the same variable or other resources that need to be read or written but whose results depend on the order in which the actions occur. For example, if a variable x is to be.

Mutual Exclusion

The problem which mutual exclusion addresses is a problem of resource sharing: how can a software system control multiple processes' access to a shared resource, when each process needs exclusive control of that resource while doing its work? The mutual-exclusion solution to this makes the shared resource available only while the process is in a specific code segment called the critical section. It controls access to the shared resource by controlling each mutual execution of that part of its program where the resource would be used.

A successful solution to this problem must have at least these two properties:

It must implement mutual exclusion: only one process can be in the critical section at a time.

It must be free of deadlocks: if processes are trying to enter the critical section, one of them must eventually be able to do so successfully, provided no process stays in the critical section permanently.

Hardware solutions

On single-processor systems, the simplest solution to achieve mutual exclusion is to disable interrupts when a process is in a critical section. This will prevent any interrupt service routines (such as the system timer, I/O interrupt request, etc) from running (effectively preventing a process from being interrupted). Although this solution is effective, it leads to many problems. If a critical section is long, then the system clock will drift every time a critical section is executed because the timer interrupt (which keeps the system clock in sync) is no longer serviced, so tracking time is impossible during the critical section. Also, if a process halts during its critical section, control will never be returned to another process, effectively halting the entire system. A more elegant method for achieving mutual exclusion is the busy-wait.

Busy-waiting is effective for both single-processor and multiprocessor systems. The use of shared memory and an atomic (remember - we talked about atomic) test-and-set instruction provide the mutual exclusion. A process can test-and-set on a variable in a section of shared memory, and since the operation is atomic, only one process can set the flag at a time. Any process that is unsuccessful in setting the flag (it is unsuccessful because the process can NOT gain access to the variable until the other process releases it) can either go on to do other tasks and try again later, release the processor to another process and try again later, or continue to loop while checking the flag until it is successful in acquiring it. Preemption is still possible, so this method allows the system to continue to function—even if a process halts while holding the lock.

Software solutions

In addition to hardware-supported solutions, some software solutions exist that use busy waiting to achieve mutual exclusion.

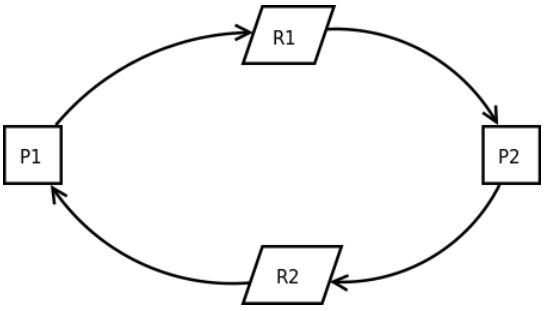
It is often preferable to use synchronization facilities provided by an operating system's multithreading library, which will take advantage of hardware solutions if possible but will use software solutions if no hardware solutions exist. For example, when the operating system's lock library is used and a thread tries to acquire an already acquired lock, the operating system could suspend the thread using a context switch and swap it out with another thread that is ready to be run, or could put that processor into a low power state if there is no other thread that can be run.

5.3 Deadlock

In concurrent computing, a deadlock is a state in which each member of a group waits for another member, including itself, to take action, such as sending a message or more commonly releasing a lock. Deadlocks are a common problem in multiprocessing systems, parallel computing, and distributed systems, where software and hardware locks are used to arbitrate shared resources and implement process synchronization.

In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock.

In a communications system, deadlocks occur mainly due to lost or corrupt signals rather than resource contention. As in the following figure, both processes need resources to continue execution. *P1* requires additional resource *R1* and is in possession of resource *R2*, *P2* requires additional resource *R2* and is in possession of *R1*; neither process can continue.



The previous image show a simple instance of deadlock. Two resources are "stuck", because the other process has control of the resource that the process needs to continue to process. While this can occure quite easily, there is usually code in place to keep this from happening. As we discussed in the previous module there are various inter-process communication techniques that can actually keep processes from becoming deadlocked due to resource contention. So, often times when we hit a deadlock like this it is something to do with the IPC that is not handling this situation properly.

5.3.1 Necessary conditions

A deadlock situation on a resource can arise if and only if all of the following conditions hold simultaneously in a system:

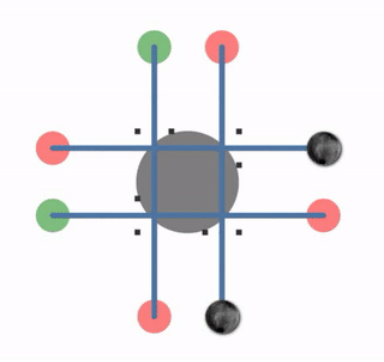
Mutual exclusion: At least one resource must be held in a non-shareable mode. Otherwise, the processes would not be prevented from using the resource when necessary. Only one process can use the resource at any given instant of time. **Hold and wait or resource holding**: a process is currently holding at least one resource and requesting additional resources which are being held by other processes.

No preemption: a resource can be released only voluntarily by the process holding it. **Circular wait**: each process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource. In general, there is a set of waiting processes, P = {P1, P2, ..., PN}, such that P1 is waiting for a resource held by P2, P2 is waiting for a resource held by P3 and so on until PN is waiting for a resource held by P1.

These four conditions are known as the Coffman conditions from their first description in a 1971 article by Edward G. Coffman, Jr.

While these conditions are sufficient to produce a deadlock on single-instance resource systems, they only indicate the possibility of deadlock on systems having multiple instances of resources.

The following image shows 4 processes and a single resource. The image shows 2 processes contending for the single reource (the grey circl in the middle), the it shows 3 processes contending for that resource, then finally how it looks when 4 processes contend for the same resource. The image depicts the processes waiting to gain access to the resource - only one resource at a time can have access.



5.3.2 Deadlock Detection and Prevention

Most current operating systems cannot prevent deadlocks. When a deadlock occurs, different operating systems respond to them in different non-standard manners. Most approaches work by preventing one of the four Coffman conditions from occurring, especially the fourth one. Major approaches are as follows.

Detection

Under the deadlock detection, deadlocks are allowed to occur. Then the state of the system is examined to detect that a deadlock has occurred and subsequently it is corrected. An algorithm is employed that tracks resource allocation and process states, it rolls back and restarts one or more of the processes in order to remove the detected deadlock. Detecting a deadlock that has already occurred is easily possible since the resources that each process has locked and/or currently requested are known to the resource scheduler of the operating system.

After a deadlock is detected, it can be corrected by using one of the following methods

1. Process termination: one or more processes involved in the deadlock may be aborted. One could choose to abort all competing processes involved in the deadlock. This ensures that deadlock is resolved with certainty and speed. But the expense is high as partial computations will be lost. Or, one could choose to abort one process at a time until the deadlock is resolved. This approach has high overhead because after each abort an algorithm must determine whether the system is still in deadlock. Several factors must be considered while choosing a candidate for termination, such as priority and age of the process.
2. Resource preemption: resources allocated to various processes may be successively preempted and allocated to other processes until the deadlock is broken.

Prevention

Deadlock prevention works by preventing one of the four Coffman conditions from occurring.

1. Removing the mutual exclusion condition means that no process will have exclusive access to a resource. This proves impossible for resources that cannot be spooled. But even with spooled resources, the deadlock could still occur. Algorithms that avoid mutual exclusion are called non-blocking synchronization algorithms.
2. The hold and wait or resource holding conditions may be removed by requiring processes to request all the resources they will need before starting up (or before embarking upon a particular set of operations). This advance knowledge is frequently difficult to satisfy and, in any case, is an inefficient use of resources. Another way is to require processes to request resources only when it has none; First they must release all their currently held resources before requesting all the resources they will need from scratch. This too is often impractical. It is so because resources may be allocated and remain unused for long periods. Also, a process requiring a popular resource may

have to wait indefinitely, as such a resource may always be allocated to some process, resulting in resource starvation.

3. The no preemption condition may also be difficult or impossible to avoid as a process has to be able to have a resource for a certain amount of time, or the processing outcome may be inconsistent or thrashing may occur. However, the inability to enforce preemption may interfere with a priority algorithm. Preemption of a "locked out" resource generally implies a rollback, and is to be avoided since it is very costly in overhead. Algorithms that allow preemption include lock-free and wait-free algorithms and optimistic concurrency control. If a process holding some resources and requests for some another resource(s) that cannot be immediately allocated to it, the condition may be removed by releasing all the currently being held resources of that process.

4. The final condition is the circular wait condition. Approaches that avoid circular waits include disabling interrupts during critical sections and using a hierarchy to determine a partial ordering of resources. If no obvious hierarchy exists, even the memory address of resources has been used to determine ordering and resources are requested in the increasing order of the enumeration.

Avoidance

The banker algorithm used to **avoid deadlock** and **allocate resources** safely to each process in the computer system. The '**Safe-State'** examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes. The banker's algorithm is named because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate allocation resources. In this section, we will learn the **Banker's Algorithm** in detail. Also, we will solve problems based on the **Banker's Algorithm**. To understand the Banker's Algorithm first we will see a real word example of it.

Suppose the number of account holders in a particular bank is 'n', and the total money in a bank is 'T'. If an account holder applies for a loan; first, the bank subtracts the loan amount from full cash and then estimates the cash difference is greater than T to approve the loan amount. These steps are taken because if another person applies for a loan or withdraws some amount from the bank, it helps the bank manage and operate all things without any restriction in the functionality of the banking system.

Similarly, it works in an **operating system**. When a new process is created in a computer system, the process must provide all types of information to the operating system like upcoming processes, requests for their resources, counting them, and delays. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system. Therefore, it is also known as **deadlock avoidance algorithm** or **deadlock detection** in the operating system.

Advantages

Following are the essential characteristics of the Banker's algorithm:

1. It contains various resources that meet the requirements of each process.
2. Each process should provide information to the operating system for upcoming resource requests, the number of resources, and how long the resources will be held.
3. It helps the operating system manage and control process requests for each type of resource in the computer system.
4. The algorithm has a Max resource attribute that represents indicates each process can hold the maximum number of resources in a system.

Disadvantages

1. It requires a fixed number of processes, and no additional processes can be started in the system while executing the process.
2. The algorithm does no longer allows the processes to exchange its maximum needs while processing its tasks.
3. Each process has to know and state their maximum resource requirement in advance for the system.
4. The number of resource requests can be granted in a finite time, but the time limit for allocating the resources is one year.

When working with a banker's algorithm, it requests to know about three things:

1. How much each process can request for each resource in the system. It is denoted by the [**MAX**] request.
2. How much each process is currently holding each resource in a system. It is denoted by the [**ALLOCATED**] resource.

3. It represents the number of each resource currently available in the system. It is denoted by the [**AVAILABLE**] resource.

Following are the important data structures terms applied in the banker's algorithm as follows:

Suppose n is the number of processes, and m is the number of each type of resource used in a computer system.

1. **Available**: It is an array of length 'm' that defines each type of resource available in the system. When Available[j] = K, means that 'K' instances of Resources type R[j] are available in the system.
2. **Max:** It is a [n x m] matrix that indicates each process P[i] can store the maximum number of resources R[j] (each type) in a system.
3. **Allocation:** It is a matrix of m x n orders that indicates the type of resources currently allocated to each process in the system. When Allocation [i, j] = K, it means that process P[i] is currently allocated K instances of Resources type R[j] in the system.
4. **Need:** It is an M x N matrix sequence representing the number of remaining resources for each process. When the Need[i] [j] = k, then process P[i] may require K more instances of resources type Rj to complete the assigned work.
   Nedd[i][j] = Max[i][j] - Allocation[i][j].
5. **Finish**: It is the vector of the order **m**. It includes a Boolean value (true/false) indicating whether the process has been allocated to the requested resources, and all resources have been released after finishing its task.

The Banker's Algorithm is the combination of the safety algorithm and the resource request algorithm to control the processes and avoid deadlock in a system:

Safety Algorithm

It is a safety algorithm used to check whether or not a system is in a safe state or follows the safe sequence in a banker's algorithm:

1. There are two vectors **Wok** and **Finish** of length m and n in a safety algorithm.

Initialize: Work = Available

Finish[i] = false; for I = 0, 1, 2, 3, 4… n - 1.

2. Check the availability status for each type of resources [i], such as:

Need[i] <= Work

Finish[i] == false

If the i does not exist, go to step 4.

3. Work = Work +Allocation(i) // to get new resource allocation

Finish[i] = true

Go to step 2 to check the status of resource availability for the next process.

4. If Finish[i] == true; it means that the system is safe for all processes.

Resource Request Algorithm

A resource request algorithm checks how a system will behave when a process makes each type of resource request in a system as a request matrix.

Let create a resource request array R[i] for each process P[i]. If the Resource Requesti [j] equal to 'K', which means the process P[i] requires 'k' instances of Resources type R[j] in the system.

1. When the number of **requested resources** of each type is less than the **Need** resources, go to step 2 and if the condition fails, which means that the process P[i] exceeds its maximum claim for the resource. As the expression suggests:

If Request(i) <= Need

Go to step 2;

2. And when the number of requested resources of each type is less than the available resource for each process, go to step (3). As the expression suggests:

If Request(i) <= Available

Else Process P[i] must wait for the resource since it is not available for use.

3. When the requested resource is allocated to the process by changing state:

Available = Available - Request

Allocation(i) = Allocation(i) + Request (i)

Need(i) = Need(i) – Request(i)

When the resource allocation state is safe, its resources are allocated to the process P(i). And if the new state is unsafe, the Process P (i) has to wait for each type of Request R(i) and restore the old resource-allocation state.

Example: Consider a system that contains five processes P1, P2, P3, P4, P5 and the three resource types A, B and C. Following are the resources types: A has 10, B has 5 and the resource type C has 7 instances.

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			
P3	3	0	2	9	0	2			
P4	2	1	1	2	2	2			
P5	0	0	2	4	3	3			

Answer the following questions using the banker's algorithm:

1. What is the reference of the need matrix?
2. Determine if the system is safe or not.
3. What will happen if the resource request (1, 0, 0) for process P1 can the system accept this request immediately?

Process	Need		
	A	B	C
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

Ans. 2: Context of the need matrix is as follows:

Need [i] = Max [i] - Allocation [i]

Need for P1: (7, 5, 3) - (0, 1, 0) = 7, 4, 3

Need for P2: (3, 2, 2) - (2, 0, 0) = 1, 2, 2

Need for P3: (9, 0, 2) - (3, 0, 2) = 6, 0, 0

Need for P4: (2, 2, 2) - (2, 1, 1) = 0, 1, 1

Need for P5: (4, 3, 3) - (0, 0, 2) = 4, 3, 1

Hence, we created the context of need matrix.

Ans. 2: Apply the Banker's Algorithm:

Available Resources of A, B and C are 3, 3, and 2.

Now we check if each type of resource request is available for each process.

Step 1: For Process P1:

Need <= Available

7, 4, 3 <= 3, 3, 2 condition is **false**.

So, we examine another process, P2.

Step 2: For Process P2:

Need <= Available

1, 2, 2 <= 3, 3, 2 condition **true**

New available = available + Allocation

(3, 3, 2) + (2, 0, 0) => 5, 3, 2

Similarly, we examine another process P3.

Step 3: For Process P3:

P3 Need <= Available

6, 0, 0 < = 5, 3, 2 condition is **false**.

Similarly, we examine another process, P4.

Step 4: For Process P4:

P4 Need <= Available

0, 1, 1 <= 5, 3, 2 condition is **true**

New Available resource = Available + Allocation

5, 3, 2 + 2, 1, 1 => 7, 4, 3

Similarly, we examine another process P5.

Step 5: For Process P5:

P5 Need <= Available

4, 3, 1 <= 7, 4, 3 condition is **true**

New available resource = Available + Allocation

7, 4, 3 + 0, 0, 2 => 7, 4, 5

Now, we again examine each type of resource request for processes P1 and P3.

Step 6: For Process P1:

P1 Need <= Available

7, 4, 3 <= 7, 4, 5 condition is **true**

New Available Resource = Available + Allocation

7, 4, 5 + 0, 1, 0 => 7, 5, 5

So, we examine another process P2.

Step 7: For Process P3:

P3 Need <= Available

6, 0, 0 <= 7, 5, 5 condition is true

New Available Resource = Available + Allocation

7, 5, 5 + 3, 0, 2 => 10, 5, 7

Hence, we execute the banker's algorithm to find the safe state and the safe sequence like P2, P4, P5, P1 and P3.

Ans. 3: For granting the Request (1, 0, 2), first we have to check that **Request <= Available**, that is (1, 0, 2) <= (3, 3, 2), since the condition is true. So the process P1 gets the request immediately.

EXERCISES

1. There are 5 processes, P0 through P4, and 4 types of resources. At T0 we have the following system state: Max Instances of Resources (A, B, C, D)→ (3, 17, 16, 12)

Given Matrices												
	Allocation Matrix (N0 of the allocated resources By a process)				**Max Matrix Max resources that may be used by a process**				**Available Matrix Not Allocated Resources**			
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P_0	0	1	1	0	0	2	1	0	1	5	2	0
P_1	1	2	3	1	1	6	5	2				
P_2	1	3	6	5	2	3	6	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				
Total	**2**	**12**	**14**	**12**								

a. Create the need matrix(max-allocation)

b. Use the safety algorithm to test if the system is in a safe state or not?

2. Consider the following snapshot of a system, Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the processes may complete.

	Allocation	Max
	ABCD	ABCD
P_0	3014	5117
P_1	2210	3211
P_2	3121	3321
P_3	0510	4512
P_4	4212	6325

a. Available = (0, 3, 0, 1)

b. Available = (1, 0, 0, 2)

3. Assume that there are three resources, A, B, and C. There are 4 processes P_0 to P_3. At T_0 we have the following snapshot of the system:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	1	0	1	2	1	1	2	1	1
P_1	2	1	2	5	4	4			
P_2	3	0	0	3	1	1			
P_3	1	0	1	1	1	1			

a. Create the need matrix.

b. Is the system in a safe state? Why or why not?

Chapter 6

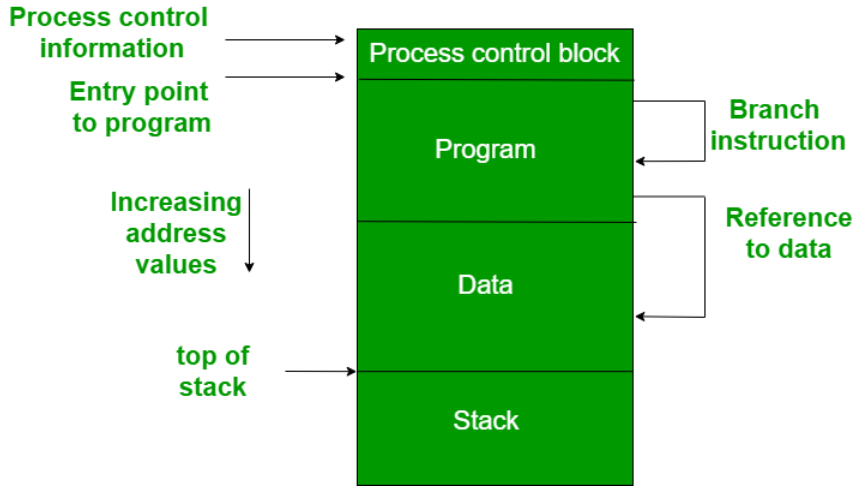
Memory Management

Memory management keeps track of the status of each memory location, whether it is allocated or free. It allocates the memory dynamically to the programs at their request and frees it for reuse when it is no longer needed.

6.1 Memory Management Requirements

1. **Relocation –** The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of his program. Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process.

When a program gets swapped out to a disk memory, then it is not always possible that when it is swapped back into main memory then it occupies the previous memory location, since the location may still be occupied by another process. We may need to **relocate** the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.



The figure depicts a process image. Every process looks like this in memory. Each process contains:

1) process control blocks;

2) a program entry point - this is the instruction where the program starts execution;

3) a program section;

4) a data section;

5) a stack. The process image is occupying a continuous region of main memory.

The operating system will need to know many things including the location of process control information, the execution stack, and the code entry. Within a program, there are memory references in various instructions and these are called logical addresses.

After loading of the program into main memory, the processor and the operating system must be able to translate logical addresses into physical addresses. Branch instructions contain the address of the next instruction to be executed. Data reference instructions contain the address of byte or word of data referenced.

2. **Protection –** There is always a danger when we have multiple programs executing at the same time - one program may write to the address space of another program. So every process must be protected against unwanted interference if one process tries to write into the memory space of another process - whether accidental or incidental. The operating system makes a trade-off between relocation and protection requirement: in order to satisfy the relocation requirement the difficulty of satisfying the protection requirement increases in difficulty.

It is impossible to predict the location of a program in main memory, which is why it is impossible to determine the absolute address at compile time and thereby attempt to assure protection. Most programming languages provide for dynamic calculation of address at run time. The memory protection requirement must be satisfied by the processor rather than the operating system because the operating system does not necessarily control a process when it occupies the processor. Thus it is not possible to check the validity of memory references.

3. **Sharing –** A protection mechanism must allow several processes to access the same portion of main memory. This must allow for each processes the ability to access the same copy of the program rather than have their own separate copy.

This concept has an advantage. For example, multiple processes may use the same system file and it is natural to load one copy of the file in main memory and let it shared by those processes. It is the task of memory management to allow controlled access to the shared areas of memory without compromising the protection. Mechanisms are used to support relocation supported sharing capabilities.

4. **Logical organization –** Main memory is organized as linear or it can be a one-dimensional address space which consists of a sequence of bytes or words. Most of the programs can be organized into modules, some of those are unmodifiable (read-only, execute only) and some of those contain data that can be modified. To effectively deal with a user program, the operating

system and computer hardware must support a basic module to provide the required protection and sharing. It has the following advantages:

- Modules are written and compiled independently and all the references from one module to another module are resolved by the system at run time.
- Different modules are provided with different degrees of protection.
- There are mechanisms by which modules can be shared among processes.
- Sharing can be provided on a module level that lets the user specify the sharing that is desired.

5. **Physical organization –** The structure of computer memory has two levels referred to as main memory and secondary memory. Main memory is relatively very fast and costly as compared to the secondary memory. Main memory is volatile. Thus secondary memory is provided for storage of data on a long-term basis while the main memory holds currently used programs. The major system concern between main memory and secondary memory is the flow of information and it is impractical for programmers to understand this for two reasons:

The programmer may engage in a practice known as overlaying when the main memory available for a program and its data may be insufficient. It allows different modules to be assigned to the same region of memory. One disadvantage is that it is time-consuming for the programmer. In a multiprogramming environment, the programmer does not know how much space will be available at the time of coding and where that space will be located inside the memory.

6.2 Memory Partitioning

In the world of computer operating system, there are four common memory management techniques. They are:

1. **Single contiguous allocation:** Simplest allocation method used by MS-DOS. All memory (except some reserved for OS) is available to a process.

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

3. **Paged memory management:** Memory is divided into fixed-sized units called page frames, used in a virtual memory environment.

4. **Segmented memory management:** Memory is divided into different segments (a segment is a logical grouping of the process' data or code).In this management, allocated memory doesn't have to be contiguous.

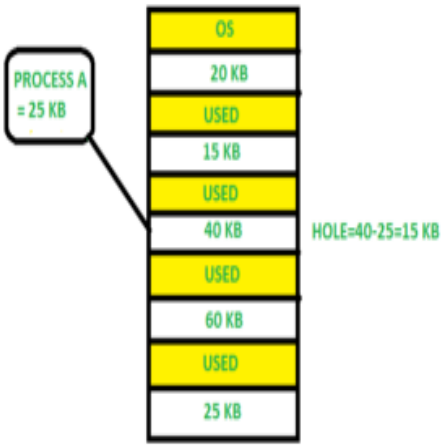
Most of the operating systems (for example Windows and Linux) use segmentation with paging. A process is divided into segments and individual segments have pages.

In p**artition allocation**, when there is more than one partition freely available to accommodate a process's request, a partition must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation.

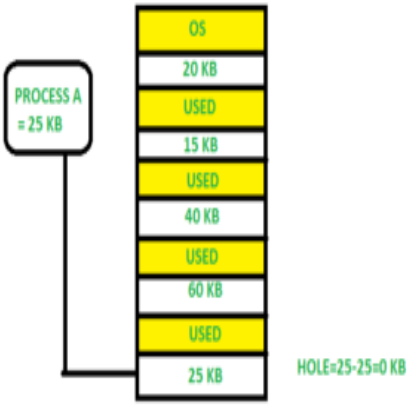
When it is time to load a process into the main memory and if there is more than one free block of memory of sufficient size then the OS decides which free block to allocate.

There are different Placement Algorithm:

1. **First Fit**: In the first fit, the partition is allocated which is the first sufficient block from the top of main memory. It scans memory from the beginning and chooses the first available block that is large enough. Thus it allocates the first hole that is large enough.



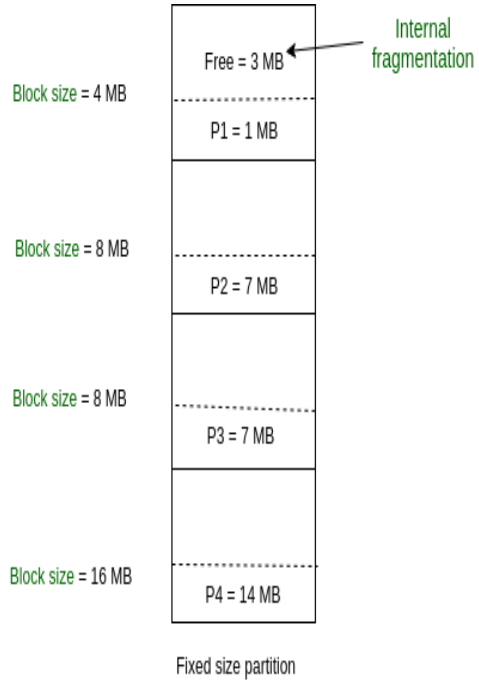
2. **Best Fit** Allocate the process to the partition which is the first smallest sufficient partition among the free available partition. It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.



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

6.2.1 Fixed Partitioning

This is the oldest and simplest technique that allows more than one processes to be loaded into main memory. In this partitioning method the number of partitions (non-overlapping) in RAM are all a fixed size, but they may or may not be same size. This method of partitioning provides for contiguous allocation, hence no spanning is allowed. The partition sizes are made before execution or during system configuration.



As illustrated in above figure, first process is only consuming 1MB out of 4MB in the main memory. Hence, Internal Fragmentation in first block is (4-1) = 3MB.
Sum of Internal Fragmentation in every block = (4-1)+(8-7)+(8-7)+(16-14)= 3+1+1+2 = 7MB.
Suppose process P5 of size 7MB comes. But this process cannot be accommodated inspite of available free space because of contiguous allocation (as spanning is not allowed). Hence, 7MB becomes part of External Fragmentation.

There are some advantages and disadvantages of fixed partitioning.

Advantages of Fixed Partitioning

- **Easy to implement:**

  Algorithms needed to implement Fixed Partitioning are easy to implement. It simply requires putting a process into certain partition without focussing on the emergence of Internal and External Fragmentation.

- **Little OS overhead:**

  Processing of Fixed Partitioning require lesser excess and indirect computational power.

Disadvantages of Fixed Partitioning

- **Internal Fragmentation:**

  Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This can cause internal fragmentation.

- **External Fragmentation:**

  The total unused space (as stated above) of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form (as spanning is not allowed).

- **Limit process size:**

  Process of size greater than size of partition in Main Memory cannot be accommodated. Partition size cannot be varied according to the size of incoming process's size. Hence, process size of 32MB in above stated example is invalid.

- **Limitation on Degree of Multiprogramming:**

  Partition in Main Memory are made before execution or during system configure. Main Memory is divided into fixed number

6.2.2 Dynamic Partitioning

Variable partitioning is part of the contiguous allocation technique. It is used to alleviate the problem faced by fixed partitioning. As opposed to fixed partitioning, in variable partitioning, partitions are not created until a process executes. At the time it is read into main memory, the process is given exactly the amount of memory needed. This technique, like the fixed partitioning scheme previously discussed have been replaced by more complex and efficient techniques.

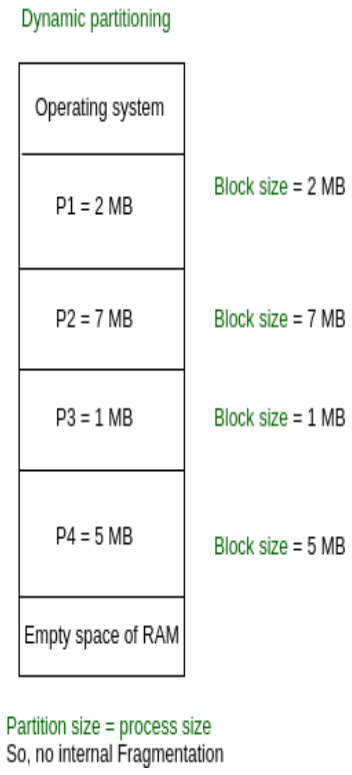
Various features associated with variable partitioning.

Initially RAM is empty and partitions are made during the run-time according to process's need instead of partitioning during system configure.

The size of partition will be equal to incoming process.

The partition size varies according to the need of the process so that the internal fragmentation can be avoided to ensure efficient utilisation of RAM.

Number of partitions in RAM is not fixed and depends on the number of incoming process and Main Memory's size.



Advantages of Variable Partitioning

- No Internal Fragmentation:

In variable Partitioning, space in main memory is allocated strictly according to the need of process, hence there is no case of internal fragmentation. There will be no unused space left in the partition.

- No restriction on Degree of Multiprogramming:

More number of processes can be accommodated due to absence of internal fragmentation. A process can be loaded until the memory is empty.

- No Limitation on the size of the process:

In Fixed partitioning, the process with the size greater than the size of the largest partition could not be loaded and process can not be divided as it is invalid in contiguous allocation technique. Here, In variable partitioning, the process size can't be restricted since the partition size is decided according to the process size.

Disadvantages of Variable Partitioning

- Difficult Implementation:

Implementing variable Partitioning is difficult as compared to Fixed Partitioning as it involves allocation of memory during run-time rather than during system configure.

- External Fragmentation:

There will be external fragmentation inspite of absence of internal fragmentation.

For example, suppose in above example- process P1(2MB) and process P3(1MB) completed their execution. Hence two spaces are left i.e. 2MB and 1MB. Let's suppose process P5 of size 3MB comes. The empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. The rule says that process must be contiguously present in main memory to get executed. Hence it results in External Fragmentation.

6.2.3 Memory Paging

A page, memory page, or virtual page is a fixed-length contiguous block of virtual memory, described by a single entry in the page table. It is the smallest unit of data for memory management in a virtual memory operating system. Similarly, a page frame is the smallest fixed-length contiguous block of physical memory into which memory pages are mapped by the operating system

In computer operating systems, memory paging is a memory management scheme by which a computer stores and retrieves data from secondary storage for use in main memory. In this scheme, the operating system retrieves data from secondary storage (usually the swap space on the disk) in same-size blocks called pages. Paging is an important part of virtual memory implementations in modern operating systems, using secondary storage to let programs exceed the size of available physical memory.

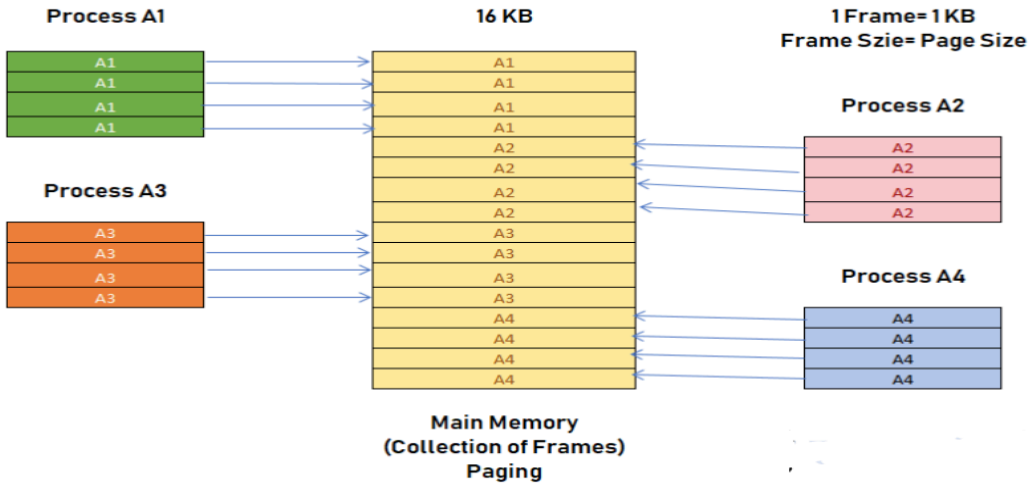
Page Table

Part of the concept of paging is the page table, which is a data structure used by the virtual memory system to store the mapping between virtual addresses and physical addresses. Virtual addresses are used by the executed program, while physical addresses are used by the hardware, or more specifically, by the RAM subsystem. The page table is a key component of virtual address translation which is necessary to access data in memory.

Role of the page table

In operating systems that use virtual memory, every process is given the impression that it is working with large, contiguous sections of memory. Physically, the memory of each process may be dispersed across different areas of physical memory, or may have been moved (paged out) to another storage, typically to a hard disk drive or solid state drive.

When a process requests access to data in its memory, it is the responsibility of the operating system to map the virtual address provided by the process to the physical address of the actual memory where that data is stored. The page table is where the operating system stores its mappings of virtual addresses to physical addresses, with each mapping also known as a *page table entry* (PTE).



Advantages of Paging

- Easy to use memory management algorithm
- No need for external Fragmentation
- Swapping is easy between equal-sized pages and page frames.

Disadvantages of Paging

- May cause Internal fragmentation
- Page tables consume additonal memory.
- Multi-level paging may lead to memory reference overhead.

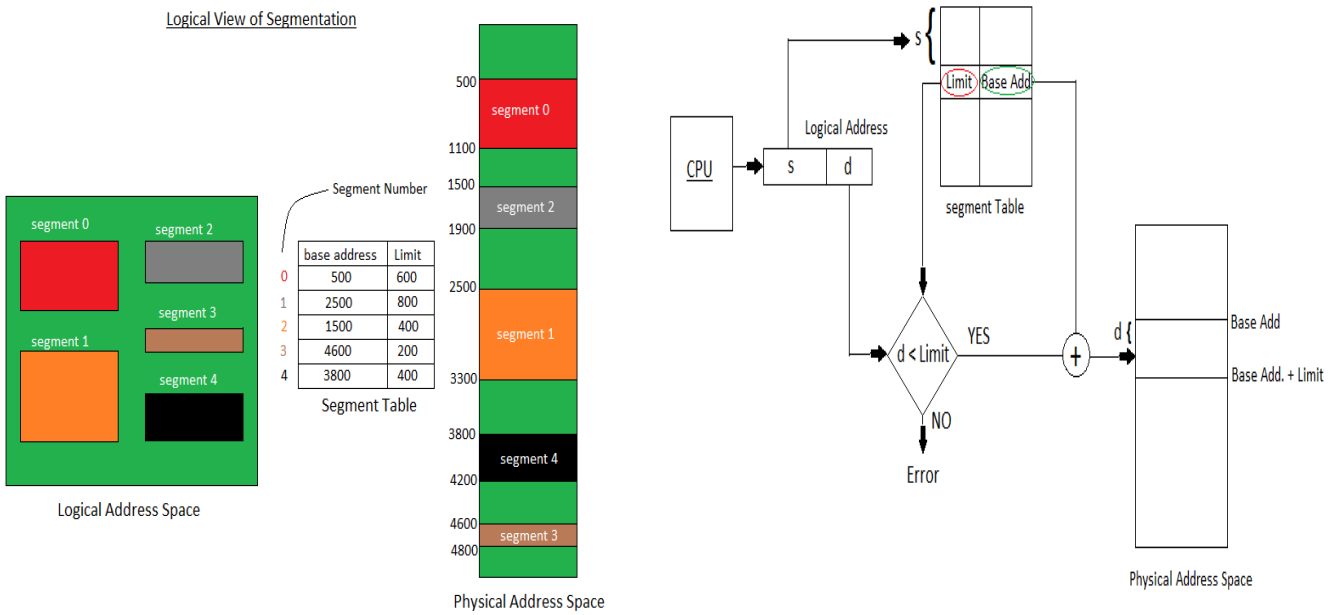
6.2.4 Segmentation

A process is divided into segments. The segments are not required to be of the same sizes.

Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously.

There is no simple relationship between logical addresses and physical addresses in segmentation.

A table stores the information about all such segments and is called Segment Table.



The Segment Table maps the logical address, made up of the base address and the limit, into one-dimensional physical address. It's each table entry has:

Base Address: It contains the starting physical address where the segments reside in memory.

Limit: It specifies the length of the segment.

Translation of a two dimensional Logical Address to one dimensional Physical Address. Address generated by the CPU is divided into:

Segment number (s): Number of bits required to represent the segment.

Segment offset (d): Number of bits required to represent the size of the segment. Walking through the diagram above:

1. CPU generates a 2 part logival address.

2. The segment number is used to get the Limit and the Base Address value from the segment table. 3. If the segment offset (d) is less than the Limit value from the segment table then The Base Address returned from the segment table, points to the beginning of the segment The Limit value points to the end of the segment in physcial memory.

Advantages of Segmentation

- No Internal fragmentation.
- Segment Table consumes less space in comparison to Page table in paging.

Disadvantage of Segmentation

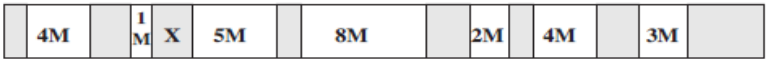
- As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.

EXERCISES

1. **Consider a fixed partitioning scheme** with equal size partitions of 2^{16} bytes and a total main memory size of 2^{24} bytes. A process table is maintained that includes a pointer to a partition for each resident process. How many bits are required for the pointer?

2. **Given six memory partitions** of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and next-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)? For each algorithm, draw a horizontal segment under the memory strip and label it clearly

3. **The following diagram** shows an example of memory configuration under dynamic partitioning, after a number of placement and swapping-out operations have been carried out. Addresses go from left to right; gray areas indicate blocks occupied by processes; white areas indicate free memory blocks. The last process placed is 2-Mbyte and is marked with an X. only one process was swapped out after that.



   a. What was the maximum size of the swapped out process?
   b. What was the size of the free block just before it was partitioned by X?
   c. A new 3-Mbyte allocation request must be satisfied next, indicate the intervals of memory where a partition will be created for the new process under the following placement algorithms the first-fit, best-fit, and next-fit. For each algorithm, draw a horizontal segment under the memory strip and label it clearly.

4. **Consider a logical address space** of 64 pages of 1,024 words each, mapped onto a physical memory of 32 frames.
   a. How many bits are there in the logical address?
   b. How many bits are there in the physical address?

5. **Consider a simple paging system** with the following parameters: 232 bytes of physical memory, paging size of s10bytes, 216 pages of logical address space.
   a. How many bits are in logical address?
   b. How many bytes in frame?
   c. How many bits in the physical address specify the frame?
   d. How many entries in the page table?
   e. How many bits in each page table entry? Assume each page table entry contains a valid/invalid bit.

6. **Write the binary translation** of the logical address 001010010111010 under the following memory management schemes, and explain your answer:
   a. A paging system with a 256-address page size, using a page table in which the frame number happens to be four times smaller the page number.
   b. A segmentation system with a 1K-address maximum segment size, using a segment table in which bases happen to be regularly placed at real addresses:22+4096*segment no.

7. **Consider a simple segmentation system** that has the following segment table:

Starting Address	Length (bytes)
660	248
1752	422
222	198
996	604

For each of the following logical addresses, determine the physical address or indicate if a segment fault occurs:

a. 0, 198
b. 2, 156
c. 1, 530
d. 3, 444
e. 0, 222

REFERENCES

[1] SILBERSCHATZ, A. (2021) Operating system concepts. S.l.: JOHN WILEY.

[2] CPU scheduling in operating systems (2023) GeeksforGeeks.

[3] Operating system scheduling algorithms (2023) Online Courses and eBooks Library.

[4] Banker's algorithm in operating system (OS) - javatpoint (no date) www.javatpoint.com.

[5] Banker's algorithm in operating system (no date) Studytonight.com.

[6] College, P.M.J.D. et al. (2021a) Operating system: The basics, Engineering LibreTexts.

[7] Stallings, W. (2015) Operating systems: Internals and design principles. Upper Saddle River: Pearson Education.