

An Introduction To The Operating System By Coding Ninjas

What is an Operating System?

An **operating system** is a piece of software that manages all the resources of a computer system, both hardware and software, and provides an environment in which the user can execute his/her programs in a convenient and efficient manner.

Operating system objectives:

- Execute user programs and make solving user problems easier
- Make the computer system convenient to use
- Use the computer hardware in an efficient manner

Design terms

- Easy (ease of use)
- Personal computers
- Efficiency (allocation of appropriate resources)
- High performance computers
- Energy saving
- Minor user interruptions
- Device is installed

Tasks offered by an Operating System

- **Program implementation**

Operating system loads a program into memory and executes the program. The program must be able to end its execution, either normally or abnormally.

- **I/O Performance**

I/O means any file or any specific I/O device. Program may require any I/O device while running. So the operating system must provide the required I/O.

- **File system manipulation**

Program needs to read a file or write a file. The operating system gives the permission to the program for operation on file.

- **Communication**

Data transfer between two processes is required for some lime. The both processes are on the one computer or on different computers but connected through computer network Communication may be implemented by two methods: shared memory and message passing.

- **Error detection**

Error may occur in CPU, in I/O device or in the memory hardware. The operating system constantly needs to be aware of possible errors. It should take the appropriate action to ensure correct and consistent computing.

Operating System Types

- **Single process operating system**

A computer system that allows only one user to use the computer at a given time is known as a single-user system. The goals of such systems are maximizing user convenience and responsiveness, instead of maximizing the utilization of the CPU and peripheral devices.

Example: Windows, Apple Mac OS,etc.

- **Batch-processing system**

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

Example: Payroll System,Bank Statements

- **Time-Sharing Operating System**

The **Time Shared Operating System** is also known as the Multitasking Operating System.Time-sharing operating systems implements CPU scheduling and multi programming systems which deliver to every user a small piece of operating time.

Example: UNIX, Multics, Linux, Windows 2000 server,etc.

- **Distributed Operating System**

Distributed operating system allows distributing of entire systems on the couples of center processors, and it serves on the multiple real time products as well as multiple users.

Example: Windows Server 2003, Windows Server 2008, Windows Server 2012, Ubuntu, Linux(Apache Server),etc.

- **Real Time Operating System**

Real time systems are used when strict time requirements are placed on the operation of a processor or the flow of data. These are used to control a device in a dedicated application.

Example: Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

- **Network Operating System**

Network Operating System is a computer operating system that facilitates to connect and communicate various autonomous computers over a network. An Autonomous computer is an independent computer that has its own local memory, hardware, and O.S. It is self capable to perform operations and processing for a single user. They can either run the same or different O.S.

Example: Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD, etc

Previous

Next

Interview Questions

Interview Questions

Q 1. What is an operating system? Explain in one sentence.

It is the interface between the user and the computer hardware.

Q 2. What are payroll systems? Payroll systems are an example of which type of operating system? (Zomato)

Payroll systems manage everything having to do with the process of paying employees and filing employment taxes. They are an example of a batch processing system.

Q 3. Explain about real time and distributed operating systems. (IBM)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “Types of operating systems”.

Q 4. Suppose you have to do an experiment which requires accuracy upto 1/1000000 of a second. Which system will you choose for the part? (Amazon)

One should use “Real time operating system” in this case, because the operation has time constraints in microseconds and we will get the desired accuracy using this operating system.

Previous

Next

Process Notes

Process

What is a process?

The process is an example of a computer program used. Contains the program code and its current function. Depending on the operating system (OS), a process can be performed with multiple configurations that issue commands simultaneously. Each process has a complete set of variations.

Process Conditions:

- **New State:** This is the state when the process is just created. It is the first state of a process.
- **Ready State:** After the creation of the process, when the process is ready for its execution then it goes into the ready state. In a ready state, the process is ready for its execution by the CPU but it is waiting for its turn to come. There can be more than one process in the ready state.
- **Ready Suspended State:** There can be more than one process in the ready state but due to memory constraint, if the memory is full then some process from the ready state gets placed in the ready suspended state.
- **Running State:** Amongst the process present in the ready state, the CPU chooses one process amongst them by using some CPU scheduling algorithm. The process will now be executed by the CPU and it is in the running state.
- **Waiting or Blocked State:** During the execution of the process, the process might require some I/O operation like writing on file or some more priority process might come. In these situations, the running process will have to go into the waiting or blocked state and the other process will come for its execution. So, the process is waiting for something in the waiting state.
- **Waiting Suspended State:** When the waiting queue of the system becomes full then some of the processes will be sent to the waiting suspended state.
- **Terminated State:** After the complete execution of the process, the process comes into the terminated state and the information related to this process is deleted.

Process Control Block(PCB)

Each process is represented in the operating system by a process control block (PCB) also called a task control block. It contains many pieces of information associated with a specific process, including these:

- **Process:** The state may be new, ready, running, and so on
- **Program counter:** It indicates the address of the next instruction to be executed for this program.
- **CPU registers:** These vary in number and type based on architecture. They include accumulators, stack pointers, general-purpose registers, etc.
- **CPU scheduling:** This includes process priority, pointers to scheduling queues, and any scheduling parameters.
- **Memory-management:** This includes the value of base and limit registers (protection) and page tables, segment tables depending on memory.
- **Accounting:** It includes the amount of CPU and real-time used, account numbers, process numbers, etc
- **I/O status information:** It includes a list of I/O devices allocated to this process, a list of open files, etc

Program vs Process

Multiprogramming

A computer running more than one program at a time (like running Excel and Firefox simultaneously).

Multiprogramming increases CPU utilization by keeping multiple jobs (code and data) in the memory so that the CPU always has one to execute.

Multitasking

Multitasking has the same meaning of multiprogramming but in a more general sense, as it refers to having multiple (programs, processes, tasks, threads) running at the same time. Multitasking is a logical extension of multiprogramming.

CPU executes multiple tasks by switching among them.

The switching is very fast.

Requires an interactive (hands-on) computer where the user can directly interact with the computer.

Multiprocessing

Multi Processing sometimes refers to executing multiple processes (programs) at the same time.

A system can be both multi programmed by having multiple programs running at the same time and multiprocessing by having more than one physical processor.

Previous

Next

Interview Questions

Interview Questions

Q 1. What are the types of processes? (Wipro)

CPU Bound Process, I/O Bound Process

Q 2. Explain the different sections of the process. (Infosys)

There are mainly four sections in a process. They are as below:

- Stack: contains local variables, returns address
- Heap: Dynamically allocated memory via malloc, calloc, realloc
- Data: contains global and static variables.
- Code or text: contains code, program counter, and content of processor's register.

Q 3. What is the Zombie process?

A zombie process is a process that has completed and in the terminated state but has its entry in the process table. It shows that the resources are held by the process and are not free.

Q 4. Name all the states of a Process. (Amazon)

- New
- Running
- Waiting

- Ready
- Terminate

Q 5. Give the advantages of the multiprocessor system. (Persistent)

Multi-processor, as its name says, it's clear that it uses more than one processor with the increase in processor number; the processing capability of the system increases too.

Q 6. Define a multi-programming system.

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Multiprogramming**”.

Q 7. Define multitasking systems.

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Multitasking**”.

Previous

Next

Threads

Threads

What is a thread?

Thread is an execution unit that consists of its own program counter, a stack, and a set of registers. Threads are also known as Lightweight processes. Threads are a popular way to improve the application through parallelism. The CPU switches rapidly back and forth among the threads giving the illusion that the threads are running in parallel. As each thread has its own independent resource for process execution, multiple processes can be executed parallelly by increasing the number of threads.

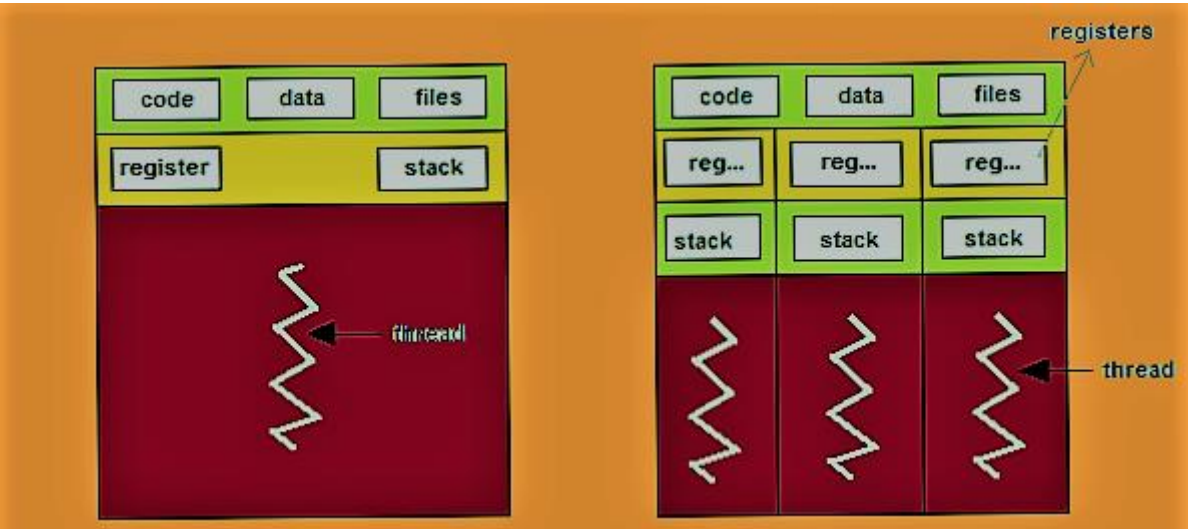
Types of Thread

- **User threads** are above the kernel and without kernel support. These are the threads that application programmers use in their programs.
- **Kernel threads** are supported within the kernel of the OS itself. All modern operating systems support kernel-level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

Program vs Process vs Thread

Multithreading

Multithreading is a phenomenon of executing multiple threads at the same time. 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.



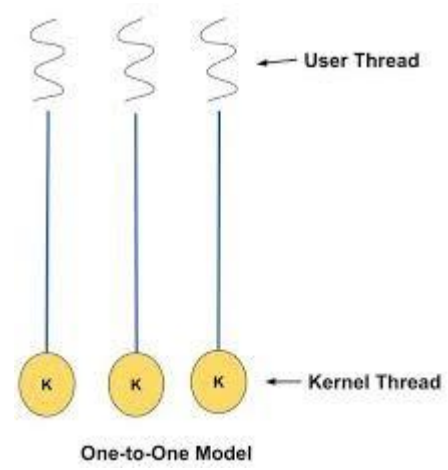
Multithreading Models

The user threads must be mapped to kernel threads, by one of the following strategies:

Many to One Model

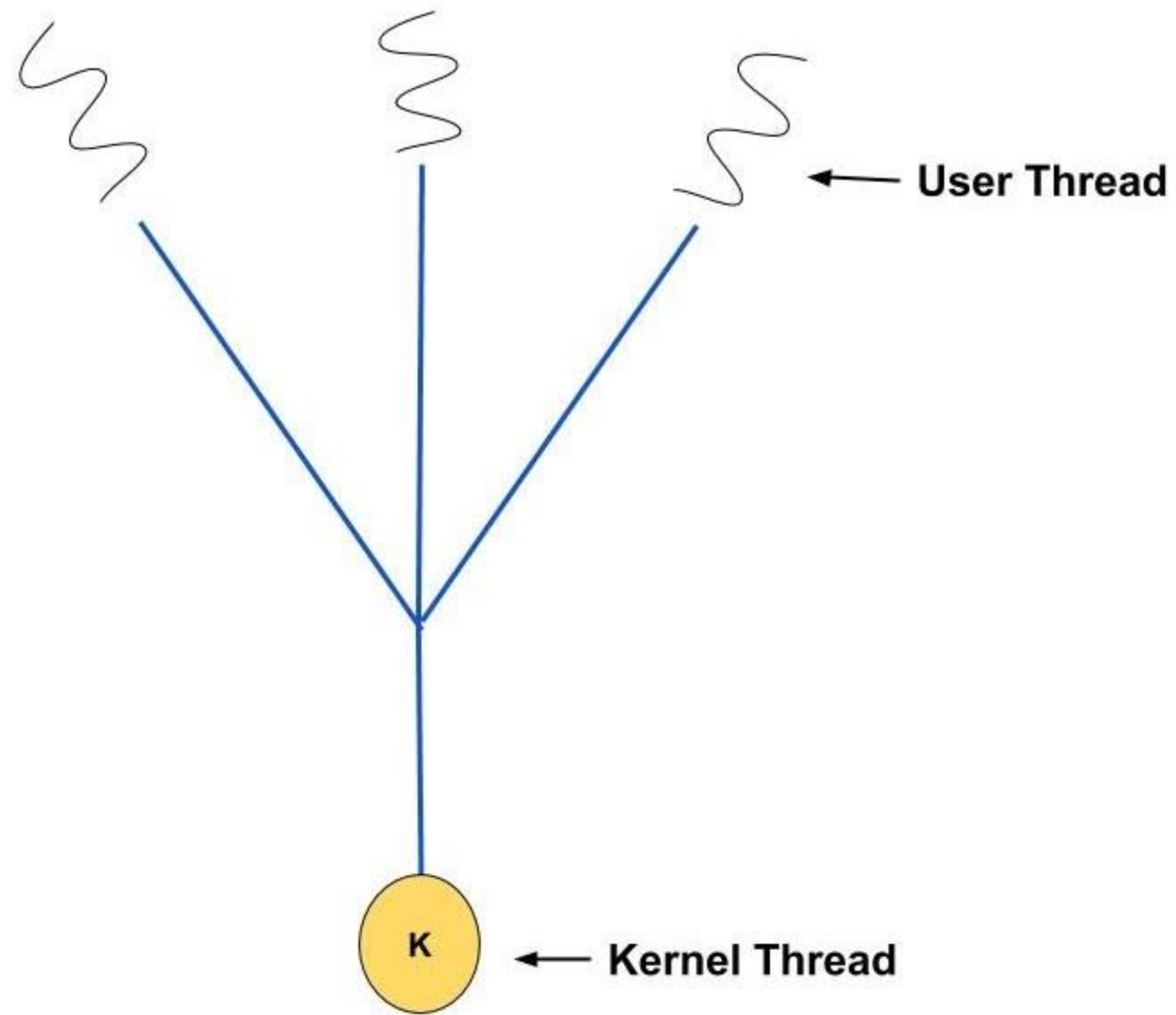
As the name suggests there is many to one relationship between threads. Here, multiple user threads are associated or mapped with one kernel thread. The thread management is done on the user level so it is more efficient.

Example: Initial implementation of Java threads on the Solaris system



Many to One Model

The one to one model creates a separate kernel thread to handle each and every user thread. Most implementations of this model place a limit on how many threads can be created. Linux and Windows from 95 to XP implement the one-to-one model for threads.

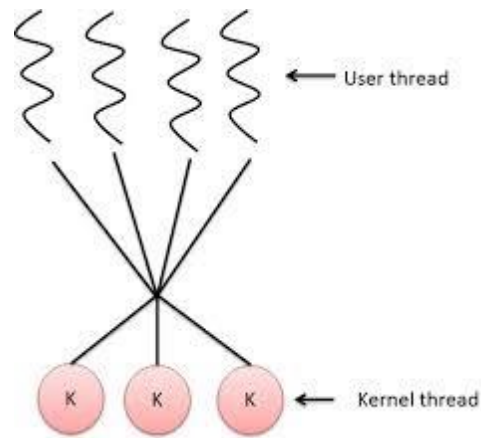


Many-to-One Model

Example: Windows NT and the OS/2 threads package

Many to Many Model

The many to many model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models. Blocking the kernel system calls does not block the entire process.



Example: Implementation of Java on an MT operating system.

[Previous](#)

[Next](#)

[Interview Questions](#)

Interview Questions

Q 1. What is Thread? (Infosys)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Thread**”.

Q 2. Types of thread? (TCS)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Types of thread**”.

Q 3. Explain the differences between multithreading and multiprocessing? (Capgemini, Wipro)

In the case of multithreading, the application processes and threads share the single or multiple core resources that include CPU caches, translation lookaside buffers, and computing units. But in the case of multiprocessing, the application implements multiple complete processing units in one or more cores. This is the basic difference between multithreading and multiprocessing since multithreading aims to implement more single-core utilization with the help of thread-level and instruction-level parallelism.

Q 4. Explain different types of multithreading? (TCS)

There are mainly three types of multithreading processes.

- Interleaved Multithreading: This type of multithreading is mainly processed to make the execution pipeline independent of any dependency by removing all the data dependency stalls from the same.
- Coarse-grained Multithreading: This is a multithreading type where one thread can continue to execute until it is blocked by a normal event that would process a long latency stall.
- Simultaneous Multithreading: This is the new and advanced type of multithreading process. In this case, the multithreading process is mainly implemented towards the superscalar processors.

Q 5.Explain different methods for threads? (Amazon)

Below are the few important methods for threads:

- Start(): This method is used to start the thread execution process. This is the basic method that should be used to start any multithreading processes.
- getName(): It is used to get the name of a thread
- setpoint(Int new priority): This method is used to change the priorities of the thread
- yield(): This method is used to pause the execution of the current thread while the other thread will carry on the normal execution phase.

Q 6. Explain the thread synchronization process? (Adobe)

In the case of multithreading, there is a presence of asynchronous behaviour in programs that can be balanced and manipulated properly through the synchronization process. Often, it has been found when multiple threads perform both read and write activities simultaneously, and then there is a chance of creating confusion. So, to prevent this inconsistency, the thread synchronization process has been implemented. In this process, when the thread gets inside the synchronized block, then it becomes unreachable for others, and no other thread can call that method.

Q 7. What do you mean by the lightweight process? (vmware)

The thread is also known as a lightweight process. It is basically a flow of information and its execution across the process code concerning its own integrated programs. It improves performance by using parallelism.

Q 8. What do you mean by Livelock? (Amazon)

When all the threads are in a blocked state and execution is stopped due to resource unavailability, then that situation is termed as livelock.

Q 9. Explain the pros and cons of multithreading. (Infosys)

Find below the pros and cons of multithreading.

Pros:

1. It can be easily extended to multiprocessors.
2. It provides independent action support to pass or fail accordingly.
3. The wait/Sleep method can be implemented more easily and cleanly.

Cons:

1. Careful synchronization is required.

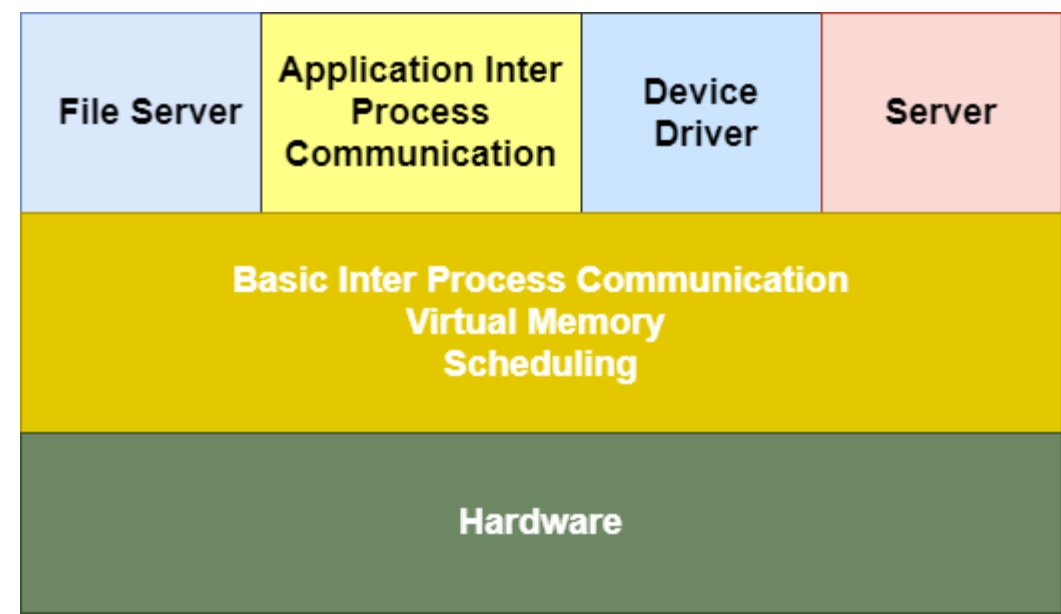
2. Sometimes create space-related issues in case of blocked threads.

[Previous](#)
[Next](#)
[Kernel](#)

Kernel

A **Kernel** is the central component of an operating system that manages operations of computers and hardware. It basically manages operations of memory and CPU time. It is the core component of an operating system. Kernel acts as a bridge between applications and data processing performed at the hardware level using inter-process communication and system calls.

Kernel Architecture



MicroKernel Architecture

Types of Kernel

1. Monolithic Kernel: Here, the OS and Kernel both run in the same memory space and are suitable where security is not a significant concern. It results in faster access, but if there is a bug in the device driver, the entire system crashes.

Example: Unix, Linux, Open VMS, XTS-400, etc.

2. Microkernel: It's a stripped-down version of Monolithic Kernel where the Kernel itself can do most of the job, and there is no need for an extra GUI. A microkernel is much smaller in size than a conventional kernel and supports only the core operating system functionalities.

Example: Mach, L4, AmigaOS, Minix, K42 etc.

3. Hybrid Kernel: This Kernel is what we see most. Windows, Apple's macOS. They are a mix of Monolithic Kernel and Microkernel. It moves out drivers but keeps system services inside the Kernel – similar to how drivers are loaded when Windows Starts the bootup process.

Example: Windows NT, Netware, BeOS etc.

4. Nano Kernel: If you need to have a kernel, but its majority of function is set up outside, then this comes into the picture.

Example: EROS etc.

5. Exo Kernel: This kernel only offers process protection and resource handling. However it is mostly used when you are testing out an inhouse project, and you upgrade to a better Kernel type.

Example: Nemesis, ExOS etc.

Shell

A **shell**, also known as a command interpreter, is that part of the operating system that receives commands from the users and gets them executed.

System calls

A system call is a mechanism using which a user program can request a service from the kernel for which it does not have permission to perform. User programs typically do not have permission to perform operations like accessing I/O devices and communicating with other programs.

A user program invokes system calls when it requires such services.

System calls provide an interface between a program and the operating system.

System calls are of different types.

Example - fork, exec, getpid, getppid, wait, exit.

[Previous](#)

[Next](#)

[Interview Questions](#)

Interview Questions

1.What is Kernel? (Wipro,Capgemini)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “Kernel”.

2.What are the main functions of the kernel? (tcs, infosys,amazon)

The main functions of a Kernel are:

- - Process management
- - Device management
- - Memory management
- - Interrupt handling
- - I/O communication
- - File system management

3.What are the different types of Kernel? (Veritas, Goldman Sachs)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “Types of Kernel”.

4. Disadvantages of kernel (vmware)

Complex process management

Debugging the messaging is complex.

Loss in performance because of the requirement of more software.

Previous

Next

Process Scheduling Intro Notes

Process Scheduling

Process Scheduling

The process scheduling is the job of the process manager that handles the removal of the current running process from the CPU and the selection of another process on the basis of a particular approach.It is an essential part of a multiprogramming operating system. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.In multiprogramming systems, one process can use CPU while another is waiting for I/O. This is possible only with process scheduling.

Scheduling Queue

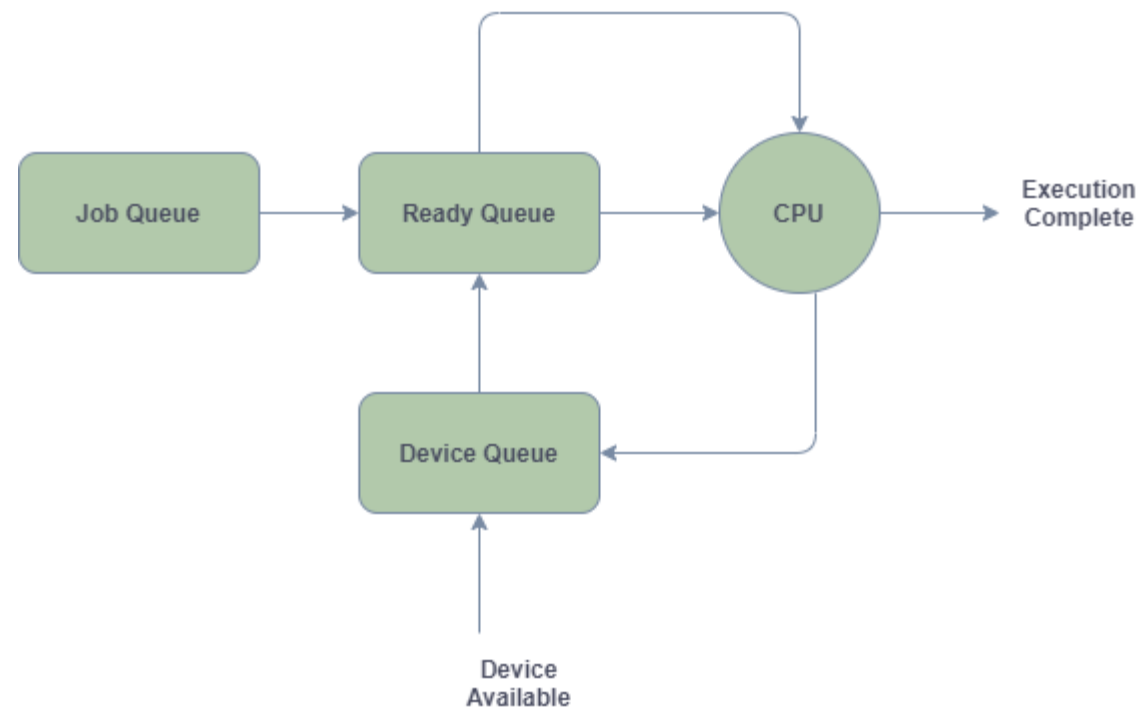
Scheduling queues refers to queues of processes or devices. When the process enters into the system, then this process is put into a job queue. This queue consists of all processes in the system. The operating system also maintains other queues such as device queue. Device queue is a queue for which multiple processes are waiting for a particular I/O device. Each device has its own device queue.

Queues are of three types:

Job Queue: As a process enters the system, it is put in a job queue that contains all the processes in the system.

Ready queue: The processes that are residing in the memory and are ready for execution are kept in the ready queue.The ready queue is implemented as a linked list of PCBs with a header containing pointers to the first and the last PCBs.

Device queue: The list of the processes waiting for a particular i/o device is called a device queue.



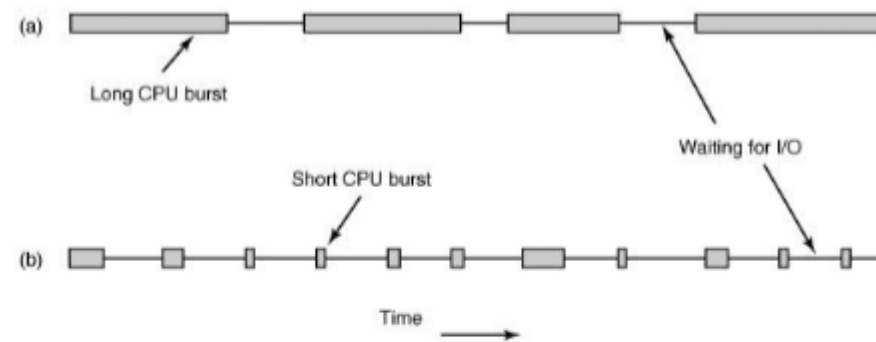
CPU Bound Process

- CPU Bound processes are those that use algorithms with a large number of statistics.
- They can be expected to hold the CPU as long as the editor allows it.
- Programs such as simulations may be tied to the CPU for most of the process life.
- Users do not expect an immediate response from a computer when using CPU-bound programs.
- They should be given the most important scheduled schedule.

I/O Bound Process

- The most awaiting processes for the completion of input or output (I / O) are I / O Bound.
- Interactive processes, such as office applications especially I / O are binding throughout the life of the process. Some processes may be bound by I / O in just a few short periods.
- The expected running time of bound I / O processes means that the process will not last very long.

- This should be provided very early by the editor.



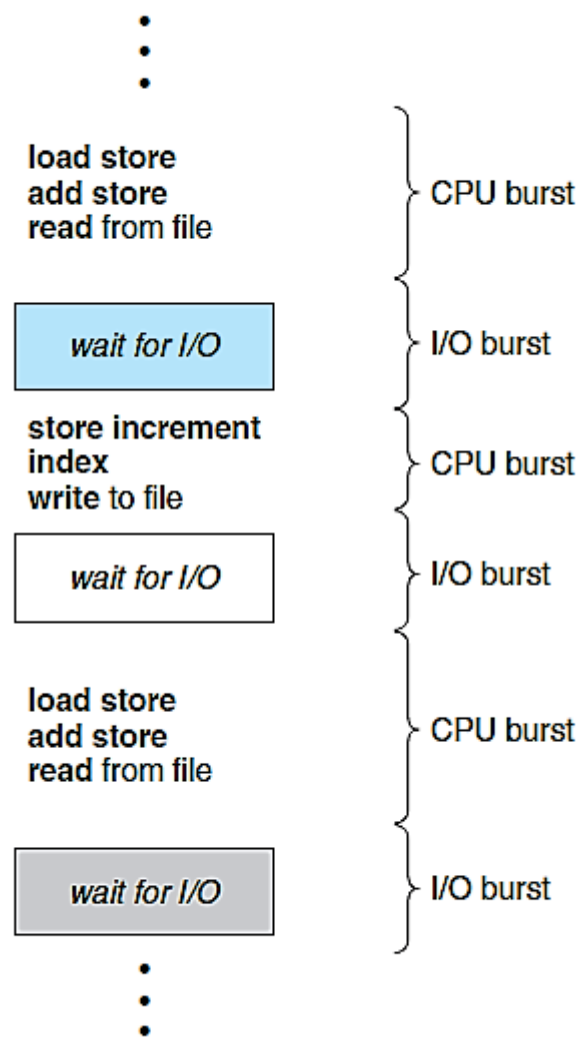
CPU bound vs. I/O bound processes

Context Switching

- Context switching is done to switch between processes.
- Switching the CPU to another process requires saving the state of the current process and reloading the state of another process. States are saved into and reloaded from PCBs.
- Context-switch time is a pure overhead as the system does not do any useful work during a control switch.
- Context-switch time depends highly on the hardware.
- Context switching is faster on RISC processors with overlapped register windows.

CPU-I/O Burst Cycle

- Process execution consists of a cycle of CPU execution and I/O wait. Processes alternate between these two states.
- Process execution begins with a CPU burst. That is followed by an I/O burst, then another CPU burst, then another I/O burst, and so on.
- Eventually, the last CPU burst will end with a system request to terminate execution, rather than with another I/O burst.



Alternate Sequence of CPU and I/O Burst

CPU Scheduler

When the CPU turns out to be idle then the operating system must select one of the essential processes in the ready queue to be executed. Short term scheduler is used for the selection process. There are several algorithms that are used for ready queue such as FIFO, LIFO priority queue and so on. Generally, PCB is used to keep a record of queues.

CPU scheduling decisions may take place when a process:

1. Switches from running to waiting for state
2. Switches from running to ready state
3. Switches from waiting to ready

4. Terminates

There are two types of scheduling:

Preemptive scheduling

In preemptive scheduling, a process can be forced to leave the CPU and switch to the ready queue. Example – Unix, Linux, Windows 95 and higher.

Non-preemptive scheduling

In non-preemptive scheduling or cooperating scheduling, a process keeps the CPU until it terminates or switches to the waiting state. Some machines support non-preemptive scheduling only. Example – Window 3.1x.

Dispatcher

A dispatcher is the module of the operating system that gives control of the CPU to the process selected by the CPU scheduler.

Steps :

- switching context
- switching to user mode
- jumping to the proper location in the user program

Dispatch latency: time it takes for the dispatcher to stop one process and start another running.

Scheduling Criteria

CPU utilization: keep the CPU as busy as possible

Throughput: No. of processes that complete their execution per time unit

Turnaround time: the amount of time to execute a particular process

Waiting time: the amount of time a process has been waiting in the ready queue

Response time: the amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Previous

Next

Interview Questions

Interview Questions

Q 1. What is Context Switching? (Amazon, TCS, Flipkart)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “Context Switching”.

Q 2. What are the different types of schedulers? (Infosys)

Long Term Scheduler

Medium Term Scheduler

Small Term Scheduler

Q 3. Explain different Scheduling Criteria? (Vmware, Microsoft)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “Scheduling Criteria”.

Q 4. What is the basic difference between preemptive and non-preemptive scheduling? (Adobe)

Preemptive scheduling allows interruption of a process while it is executing and taking the CPU to another process while non-preemptive scheduling ensures that a process keeps the CPU under control until it has completed execution.

Q 5. Is non-preemptive scheduling frequently used in a computer? Why? (Morgan Stanely)

No, it is rarely used for the reasons mentioned below:

It can not ensure that each user gets a share of the CPU regularly.

The idle time with this increases reducing the efficiency and overall performance of the system.

It allows programs to run indefinitely which means that other processes have to wait for very long.

Q 6. What is the dispatcher and what is dispatcher latency? (Phonepe, Tcs)

The dispatcher is a CPU-scheduling component that gives control of the CPU to the process selected by the short-term scheduler.

The function involves the following:

- *Switching Context

- *Switching to user mode

- *Jumping to the proper location in the user program to restart the program

Dispatcher Latency is the time taken by the dispatcher to stop one process and start another running.

Previous

Next

FCFS Notes

First-come-first-served (FCFS) scheduling

The process that asks for the CPU first is given to the CPU first. The implementation of the FCFS policy is easily handled with the FIFO queue. The average waiting time under the FCFS policy, however, is often quite long.

It is non-preemptive.

It has a high average waiting time.

The FCFS algorithm is particularly troublesome for time-sharing systems, where it is important that each user get a share of the cpu at regular intervals.

Previous

Next

Interview Questions

Interview Questions

1.What is the Convoy Effect? (Vmware)

Convoy Effect is a situation where many processes, which need to use a resource for a short time, are blocked by one process holding that resource for a long time.

This essentially leads to poor utilization of resources and hence poor performance.

2. Why is fcfs non-preemptive? (Amazon)

It is Non Preemptive algorithm, which means the process priority doesn't matter

3.Briefly explain FCFS. (TCS)

FCFS stands for First-come, first-served. It is one type of scheduling algorithm. In this scheme, the process that requests the CPU first is allocated to the CPU first. Implementation is managed by a FIFO queue

Previous

Next

SJFS NOTES

Shortest-Job-First Scheduling

This algorithm is associated with each process length of the next CPU storage. When the CPU is available, it is given a process with the next minimum CPU bust. If the two processes are the similar length for the next CPU explosion, the FCFS configuration is used to break the tire.

If a new process arrives with CPU burst length less than the remaining time of the current executing process, preempt,his scheme is known as the Shortest-Remaining-Time-First (SRTF).

SJF is optimal – gives a minimum average waiting time for a given set of processes.

Previous

Next

Interview Questions

Interview Questions

Q 1.How does the SJF algorithm work? (Flipkart)

Shortest Job First (SJF) is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

Q 2.What is the preemptive shortest job first? (Amazon)

In Preemptive Shortest Job First Scheduling, jobs are put into a ready queue as they arrive, but as a process with short burst time arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.

Q 3.What is the real difficulty with the SJF CPU scheduling algorithm? (TCS)

The real difficulty with SJF is knowing the length of the next CPU burst. For long-term (job) scheduling in a batch system, the length of process time could be specified by a user. However, at the level of short-term CPU scheduling, there is no way to know the length of the next CPU burst.

Q 4.What is the main problem with the shortest job scheduling and what is its solution? (Barclays)

The main problem with the shortest job first algorithm is starvation [1], [2]. If there is a steady supply of short processes, the long process may never get the chance to be executed by the processor. There are a variety of scheduling algorithms proposed in the past to solve the issue of starvation of SJF.

Previous

Next

Priority Scheduling Notes

Priority Scheduling

- CPU is allocated to the particular process with the highest priority.
- Priority range be 0 to 7 (say), with 0 representing the highest or the lowest priority
- Priority may depend on internal factors (time limit, memory requirement, number of open files, etc.) and external factors (user, department, etc.)
- Maybe preemptive or non-preemptive.
- SJF is an important case of priority scheduling, with priority inversely proportional to the predicted next CPU burst length.
- May cause starvation, i.e. indefinite blocking of processes
- Aging: gradually increase the priority of a process waiting for a long time

- Priority inversion: a low-priority process gets the priority of a high-priority process waiting for it

The problem with priority scheduling algorithms is indefinite blocking or starvation. A solution to the problem of indefinite blockage of low priority processes is aging.

Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time. For example if priorities range from 0 (low) to 127 (high), we could increment the priority of a waiting process by 1 every 15 mins.

Previous

Next

Interview Questions

Interview Questions

1 . What is Priority scheduling? (Paypal)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**priority scheduling**”.

2.What is the major problem with priority scheduling algorithms? (Vmware)

A major problem with priority scheduling is indefinite blocking or starvation. A solution to the problem of indefinite blockage of the low-priority process is aging. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long period of time.

3.What is the benefit of priority scheduling with aging? (Amazon)

Aging is used to ensure that jobs with lower priority will eventually complete their execution. This technique can be used to reduce starvation of low-priority tasks. There are many ways to implement aging, but all have the same principle that the priority of a process should increase as it waits in the ready queue.

4.What is starvation and aging? (Nagarro)

Starvation: Starvation is a resource management problem where a process does not get the resources it needs for a long time because the resources are being allocated to other processes. Aging: Aging is a technique to avoid starvation in a scheduling system.

[Previous](#)

[Next](#)

[Round Robin Notes](#)

Round-Robin Scheduling

The round-robin (RR) scheduling algorithm is designed especially for time-sharing systems. It is similar to FCFS scheduling, but preemption is added to switch between processes. A small unit of time called a time quantum (or time slice) is defined.

The ready queue is treated as a circular queue.

The performance of the RR algorithm depends heavily on the size of the time–quantum. If the time-quantum is very large(infinite) then RR policy is the same as FCFS policy. If time quantum is very small, the RR approach is called processor sharing and appears to the users as though each of n processes has its own processor running at 1/n the speed of a real processor.

[Previous](#)

[Next](#)

[Interview Questions](#)

Interview Questions

Q 1. Why is the round robin algorithm considered better than the first come first served algorithm? (Vmware)

The first come first served algorithm is the simplest scheduling algorithm known. The processes are assigned to the CPU on the basis of their arrival time in the ready queue. Since, it is non-preemptive once a process is assigned to the CPU, it will run till completion. Since a process takes the CPU till it is executed it is not very good in providing good response times. It can make other important processes wait unnecessarily.

On the other hand, the round-robin algorithm works on the concept of time slice or also known as quantum. In this algorithm, every process is given a predefined amount of time to complete the process. In case, a process is not completed in its predefined time then it is assigned to the next process waiting in the queue. In this way, continuous execution of processes is maintained which would not have been possible in case of FCFS algorithm

Q 2. What is Round Robin Scheduling? (TCS)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “Round Robin Scheduling”.

Q 3. What happens when we use round robin scheduling? (Samsung)

a round-robin editor usually uses time-sharing, giving each task time or quantum (its share of CPU time), and interrupting work if it is not completed at that time. The work is resumed the next time the time is given to perform that process.

[Previous](#)

[Next](#)

[Multilevel Queue Scheduling Notes](#)

Multilevel Queue Scheduling

A multilevel queue-scheduling algorithm partitions the ready queue into several separate queues. The processes are completely assigned to one queue, generally based on some unique property of the process, such as memory size, process priority, or process type.

There must be scheduled between the queues, which is commonly implemented as fixed-priority preemptive scheduling.

For example, the foreground queue may have absolute priority over the background queue.

Example: Five queens

1. System processes
2. Interactive processes
3. Interactive editing processes
4. Batch processes
5. Student processes

[Previous](#)

[Next](#)

Interview Questions

Q 1.How can multilevel queues be scheduled? (Vmware)

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

Q 2.What are the Advantages of Multilevel Queue Scheduling (Amazon)

With the help of this scheduling we can apply various kind of scheduling for different kind of processes:

For System Processes: First Come First Serve(FCFS) Scheduling.

For Interactive Processes: Shortest Job First (SJF) Scheduling.

For Batch Processes: Round Robin(RR) Scheduling

For Student Processes: Priority Scheduling

Q 3.Explain Disadvantages of Multilevel Queue Scheduling (Veritas, INTUIT)

The main disadvantage of Multilevel Queue Scheduling is the problem of starvation for lower-level processes.

Q 4.Explain Multilevel Queue Scheduling.

Please refer to the notes for definitions of the aforementioned operating systems under the heading “ Multilevel Queue Scheduling.”.

Previous

Next

Multilevel Feedback Scheduling Notes

Multilevel Feedback Queue Scheduling

In the multilevel queue-scheduling algorithm, the processes are permanently assigned to the system entry queue. Processes do not move between queues. This setup has the advantage of lower scheduling overhead, but the lack of consistency.

Multilevel Feedback queue scheduling, however, allows the process to move between queues. The idea is to split processes with different CPU-burst features. If the process consumes too much CPU time, it will be moved to a much lower line. Similarly, the longest waiting process in the lowest line can be delivered to the most important line. This form of aging prevents starvation.

- Allows processes to move between queues
- Inter-queue scheduling: preemptive priority scheduling
- A process waiting too long in a low-priority queue may be moved to a high-priority queue.

Example:

consider a multilevel feedback queue scheduler with three queues, numbered from 0 to 2 The scheduler first executes all processes in queue 0. Only when queue 0 is empty will it execute processes in queue 1. Similarly, processes in queue 2 will be executed only if queues 0 and 1 are empty. A process that arrives for queue 1 will preempt a process in queue 2. A process that arrives for queue 0 will, in turn, preempt a process in queue 1.

Previous

Next

Interview Questions

Interview Questions

Q 1. Explain Multilevel Feedback Queue Scheduling. (GS)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Multilevel Feedback Queues scheduling**”.

Q 2. Which of the following parameters are required to define a multilevel feedback queue scheduler? (Myntra)

In general, a multilevel feedback queue scheduler is defined by the following parameters: The number of queues. The scheduling algorithm for each queue which can be different from FIFO. The method used to determine when to promote a process to a higher priority queue.

Q 3.What is the difference between a multilevel queue and multilevel feedback queue?(Adobe)

[Previous](#)

[Next](#)

[Printer Spooler Problem Notes](#)

Printer Spooler Problem

As we know that a printer is a peripheral device, so it is slower in comparison to CPU and memory. So, if multiple users send some file to the printer to print then the spooler comes into play. The spooler is a program in the printer that stores all the files coming to print and when the printer is free it gives it to the printer in a sequential manner.

Example:

This four-line code is executed by each process in order to store its file in the spooler directory to print.

```
Load Ri, m[in]
store SD[Ri], "F-N"
INCR Ri
store m[in], Ri
```

in: Shared variable
m: Memory location
Ri: Register
F-N: Filename
SD: Spooler directory

Line 1: In line one we are loading free memory location m[in], in register Ri
Line 2: In line two we are storing file name (F-N) in spooler directory (SD) at position Ri, which is for instance 0
Line 3: In line three we are incrementing the count of Ri from 0 to 1, so next file can be stored in at index 1
Line 4: In line four the new file will be stored at incremented memory location m[in]

So this was all about printer spooler problems. Hope you liked it and learned something new from it.

Real-Life Example

Managing and supporting a large enterprise’s print infrastructure is a complex task. It is a huge challenge to manage printers in different buildings, different plants, even different countries. You don’t know immediately when a printer is offline, jammed, out of toner, or just swamped with too many print jobs. This lack of a single view of your enterprise print infrastructure leads to lost productivity at your company.

Previous

Next

Interview Questions

Interview Questions

Q 1.Explain Printer Spooler Problem.

Please refer to the notes for definitions of the aforementioned operating systems under the heading “Printer Spooler Problem”.

Q 2.How to fix my printer spooler problem?

How to fix a printer spooler error

1. Delete all the print jobs in this folder.
2. Head back to 'Services,' find 'Print Spooler' and right click. Now select 'Start. '
3. Now open the program you closed down and try and print the document you wanted as normal.

Q 3.What causes printer spoolers to stop working?

Sometimes Print Spooler service can keep stopping because of Print Spooler files - too many, pending, or corrupt files. Deleting your print spooler files can clear pending print jobs, or too many files or solve the corrupt files to resolve the problem.

Previous

Next

Producer Consumer Problem Notes

Producer Consumer Problem

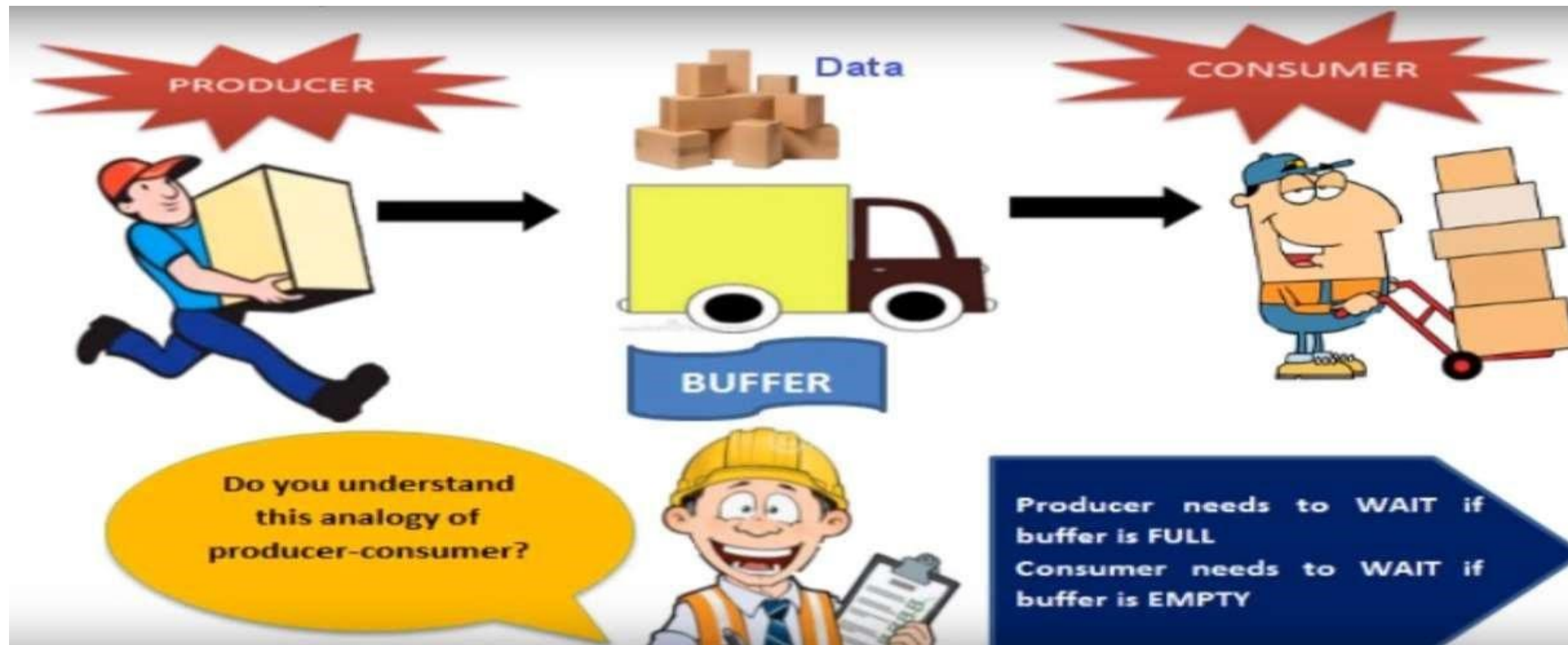
The producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. In the problem, two processes share a fixed-size buffer. One process produces information and puts it in the buffer, while the other process consumes information from the buffer. These processes do not take turns accessing the buffer, they both work concurrently.

The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

Producer:- The producer's job is to generate a piece of data, put it into the buffer and start again.

Consumer:- The consumer is consuming the data (i.e., removing it from the buffer) one piece at a time.

Real Life Example:



Solution for the Producer-Consumer

- The producer wants to put but buffer- full Go to sleep and wake up when a consumer takes one or more
- The consumer wants to take but buffer-empty go to sleep and wake up when the producer puts one or more

The producer-consumer problem using semaphores

```
#define N 100                /* number of slots in the buffer */
typedef int semaphore;       /* semaphores are a special kind of int */
semaphore mutex = 1;        /* controls access to critical region */
semaphore empty = N;        /* counts empty buffer slots */
semaphore full = 0;         /* counts full buffer slots */

void producer(void)
{
    int item;

    while (TRUE) {          /* TRUE is the constant 1 */
        item = produce_item(); /* generate something to put in buffer */
        down(&empty);        /* decrement empty count */
        down(&mutex);        /* enter critical region */
        insert_item(item);    /* put new item in buffer */
        up(&mutex);          /* leave critical region */
        up(&full);           /* increment count of full slots */
    }
}

void consumer(void)
{
    int item;

    while (TRUE) {          /* infinite loop */
        down(&full);         /* decrement full count */
        down(&mutex);        /* enter critical region */
        item = remove_item(); /* take item from buffer */
        up(&mutex);          /* leave critical region */
        up(&empty);          /* increment count of empty slots */
        consume_item(item);   /* do something with the item */
    }
}
```

The most common real-life example of the Producer-Consumer algorithm is a process, called print spooling. Print spooling refers to putting jobs(in this case documents that are about to be printed)into a special location(buffer) in either computer memory or hard disk so that a printer could access this document whenever the printer is ready. There are a couple of advantages of spooling.

First of all the printer can access data from the buffer at any rate that is suitable for the printer.

Secondly, the work of the computer is not interrupted while printing, thus a user can perform other tasks.

[Previous](#)
[Next](#)
[Interview Questions](#)

Interview Questions

Q 1. Explain Producer-Consumer problem

Please refer to the notes for definitions of the aforementioned operating systems under the heading “Producer-Consumer problem”.

Q 2. What is the producer and consumer in the problem?

Producer: The producer's job is to generate a piece of data, put it into the buffer and start again.

Consumer: The consumer is consuming the data (i.e., removing it from the buffer) one piece at a time.

[Previous](#)
[Next](#)
[Critical Section Problem Notes](#)

Critical Section Problem

When a process is accessing shared modifiable data or a resource that can only operate on behalf of one process at a time, the process is said to be in a critical section. When one process is in a critical section, all other processes (at least those that access the shared modifiable data and/or resource) are excluded from their critical section.

The Critical Section Problem

- n processes all competing to use some shared data
- Each process has a code segment, called the critical section, in which the shared data is accessed.

- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.

Example of critical section

❖ **Transfer Rs. 100 from saving account to checking account**

P1
 Saving = saving - 100
 Checking = checking + 100

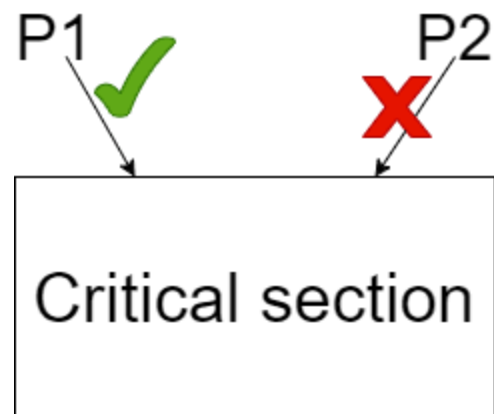
P2
 saving = saving * 1.01
 checking = checking * 101

Initially : saving = 100
 checking = 0

P1 ran first & P2 ran second	P2 ran first & p1 ran second	P1's first line then P2 & P1's second line
Saving = 0 Checking = 101	saving = 1 checking = 100	saving = 0 checking = 100
✓	✓	✗

Solution To Critical Section Problem

1.Mutual Exclusion: If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.
 Suppose if P1 wants to execute a critical section and start executing it, and then at the same time (while P1 was executing the critical section), P2 wants to enter the critical section in that case P2 will be blocked.
 And by that, we will achieve mutual exclusion.



2.Progress: If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.

Example:

Suppose there is some common code written in the critical section and at present, the critical section is empty (means that there is no process in it).

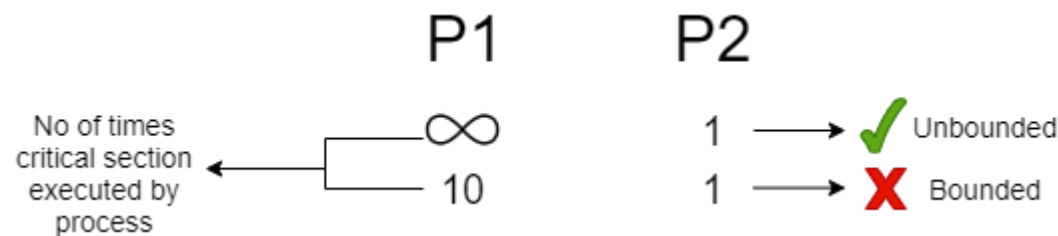
So, now P1 wants to enter the critical section and P2 is blocking it. Maybe because of some code is written in the entry section of P2.

Therefore, in this case, P2 is not going in the critical section and also blocking P1 to enter the critical section.

This means that progress is not there and this should not happen.

3.Bounded Waiting: Abound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

- Assume that each process executes at a nonzero speed
- No assumption concerning the relative speed of the n processes.



Interview Questions

Q 1. What is the critical section problem? (Amazon)

Each process has a segment of code called the critical section in which a process may be changing common variables, updating a shared table, writing a shared file, and so on. The important feature of the system is that, when one process is executing in its critical sections, no other processes are allowed to execute in its critical sections at the same time.

Each process must request permission to enter its critical section. The section of code implementing this request is the Entry section.

The critical section is followed by the exit section.

The remaining code is the remainder section.

Q 2.How does semaphore solve critical section problems? (Hp)

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that is used for process synchronization. The wait operation decrements the value of its argument S if it is positive. If S is negative or zero, then no operation is performed.

Q 3.What are the three requirements of any solution to the problem of the critical section? (Lenovo)

Three must rules which must be enforced by the critical section are: 1) Mutual Exclusion 2) Process solution 3)Bound waiting.

Q 4.What is progress in the critical section problem? (Vmware)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**solution to critical section problem 2. progress**”.

Q 5. Where do critical sections arise? (Amazon)

Critical Section in OS is a part of the program where shared resources are accessed by the processes. Critical Section Problem arises when multiple processes access the critical section at the same time.

[Previous](#)

[Next](#)

[Semaphores Notes](#)

Semaphore

As it is difficult for the application programmer to use these hardware instructions, to overcome this difficulty we use the synchronization tool called Semaphore (that does not require busy waiting). Semaphore is a type of flexible or non-flexible data that is used to control access to common shared resources through multiple processes in the same system as a multitasking operating system. It is a mechanism that can be used to provide synchronization of tasks. It is a low-level synchronization mechanism.

A semaphore is an object that consists of a counter, a waiting list of processes, and two methods: signal and wait. All the modifications to the integer value of the semaphore in the wait() and signal() atomic operations must be executed indivisibly. i.e. when one process changes the semaphore value, no other process will change the same semaphore value simultaneously.

When the count for the semaphore is 0 means that all the resources are being used by some processes. Otherwise, resources are available for the processes to allocate.

When a process is currently using a resource means that it blocks the resource until the count becomes > 0.

For example, let us consider two processes P1 and P2, and a semaphore initialized with a value of 1. The value of the semaphore becomes 0 if the process P1 enters the critical section. If the process P2 intends to enter the critical section then the value of the semaphore has to be greater than 0, until that the process has to wait. This is only possible when P1 completes the critical section and calls the Signal operation on the semaphore. Mutual exclusion is obtained this way.

Operation in Semaphore

1. **Wait**
2. **Signal**

Wait

This operation is used to control the entry of a task into the critical section.

The process that wishes to use a resource must perform the wait() operation (count is decremented).

The definition for wait() is as follows:

```
wait (S)

{

    while S <= 0 ; // no-op

    S--;

}
```

Signal

This operation is used to control the exit of a task from a critical section. This increases the value of the argument by 1.

The process that releases a resource must perform the signal() operation (count is incremented).

The definition for signal() is as follows:

```
signal (S)
```

```
{
```

```
    S++;
```

```
}
```

Types of semaphore

1. **Counting Semaphore**
2. **Binary Semaphore**

Counting semaphore

The value of the Counting Semaphore can range over an unrestricted domain. Counting Semaphores are used to control the access of given resources each of which consists of some finite no. of instances. This counting semaphore is initialized to the number of resources available.

Binary semaphore

The value of the Binary Semaphore can range between 0 and 1 only.

In some systems, the Binary Semaphore is called Mutex locks, because they are locks to provide mutual exclusion. We can use the Binary Semaphore to deal with critical section problems for multiple processes.

Implementation of semaphore

The main disadvantage of the semaphore definition is, it requires busy waiting. Because when one process is in the critical section and if another process needs to enter into the critical section must have to loop in the entry code continuously.

Implementation of semaphore with no busy waiting:

*To overcome the need for the busy waiting we have to modify the definition of wait() and signal() operations. i.e. when a process executes wait() operation and finds that it is not positive then it must wait.

* Instead of engaging the busy-wait, the process blocks itself so that there will be a chance for the CPU to select another process for execution. It is done by block() operation. - Blocked processes are placed in the waiting queue.

*Later the process that has already been blocked by itself is restarted by using wakeup() operation so that the process will move from waiting for state to ready state. - Blocked processes that are placed in the waiting queue are now placed into the ready queue.

*To implement the semaphore with no busy waiting we need to define the semaphore of the wait() and signal() operation by using the 'C' Struct.
Which is as follows:

```
typedef struct  
  
{  
  
    int value;  
  
    struct process *list;  
  
}semaphore;
```

- i.e. each semaphore has an integer value stored in the variable “value” and the list of processes list.
- When a process performs the wait() operation on the semaphore then it will add a list of processes to the list.
- When a process performs the signal() operation on the semaphore then it removes the processes from the list.

Semaphore Implementation with no Busy waiting

Implementation of wait: (definition of wait with no busy waiting)

```
wait (S)  
  
{  
  
    Value--;  
  
    if (value < 0)  
  
    {  
  
        add this process to waiting queue  
  
        block();  
  
    }  
  
}
```

Implementation of signal: (definition of signal with no busy waiting)

```
Signal (S)

{

    Value++;

    if (value <= 0)

    {

        remove a process P from the waiting queue

        wakeup(P);

    }

}
```

Advantages

The different advantages of semaphores are given below:

- They do not allow more than one process to enter the critical section. In this way, mutual exclusion is achieved and thus they are extremely efficient than other techniques for synchronization.
- Due to busy waiting in semaphore, there is no wastage of process time and resources. This is because the processes are only allowed to enter the critical section after satisfying a certain condition.
- They are machine-independent as they run in the machine-independent code of the microkernel.
- They allow flexible management of resources.

Disadvantages

The different limitations of semaphores are given below:

- There may be a situation of priority inversion where the processes having low priority get access to the critical section than the processes having higher priority.
- To avoid deadlocks, the wait() and signal() operations have to be executed in the correct order.
- Semaphore programming is complicated and there are chances of not achieving mutual exclusion.

The problem in the implementation of a semaphore

Dining Philosophers Problem

The problem with dining philosophers is that there are five philosophers who share a round table and eat and think differently. There is a small bowl of rice for each philosopher and 5 chopsticks. The philosopher needs both their right and left chopsticks in order to eat. A hungry philosopher can only eat if there are both sticks available. Otherwise, the philosopher puts down his stick and begins to think again. The dining philosopher is an old synchronization problem as it points to a large class of concurrency control problems.

Solution using semaphore

- Chopsticks are shared items (by two philosophers) and must be protected.
- Each chopstick has a semaphore with an initial value of 1.
- A philosopher calls wait() before picking up a chopstick and calls signal() to release it.

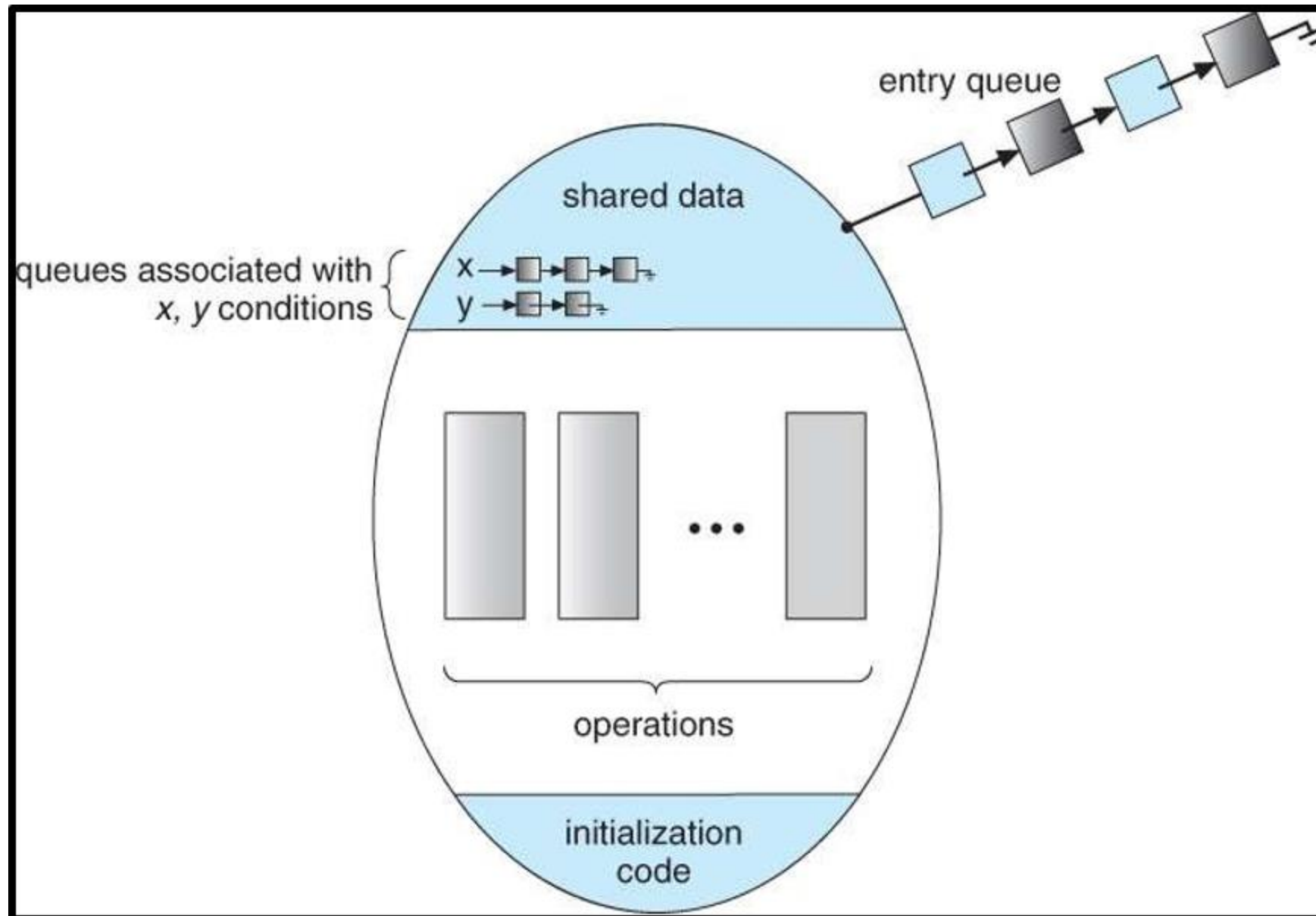
```
semaphore C[5] = 1;
```

philosopher *i*

```
while (1) {  
    // thinking  
    C[i].wait();  
    C[(i+1)%5].wait();  
    // eating  
    C[(i+1)%5].signal();  
    C[i].signal();  
    // finishes eating  
}
```

Monitor

In concurrent programming, a monitor is a synchronization construct that allows threads to have both mutual exclusion and the ability to wait (block) for a certain condition to become true. Monitors also have a mechanism for signaling other threads that their condition has been met.



A monitor can have variables of the condition type that can be accessed by wait() and signal() operations only. The operation x.wait() means that the process making this operation is suspended until another process invokes x.signal(). The operation x.signal() resumes exactly one suspended process and has no effect if there is none.

Syntax:

```
monitor monitor_name

{

    // shared variable declarations initialization_code (...)

    { ... }

    procedure P1 (...)

    { ... }

    procedure Pn (...)

    { ... }

}
```

[Previous](#)

[Next](#)

[Interview Questions](#)

Interview Questions

Q 1. What is Semaphore? (Amazon)

Semaphore is a variable, whose status reports common resource, Semaphore is of two types one is Binary semaphore and other is Counting semaphore.

Q 2. How is Mutex accessed?

must be accessed from only one process

Q 3. What will happen if a non-recursive mutex is locked more than once? (Barclays)

If a thread that had already locked a mutex, tries to lock the mutex again, it will enter into the waiting list of that mutex, which results in a deadlock. It is because no other thread can unlock the mutex.

Q 4. At a particular time of computation, the value of a counting semaphore is 7. Then 20 P operations and 15 V operations were completed on this semaphore. The resulting value of the semaphore is? (GATE 1987)

Answer: 2

Explanation: P represents Wait and V represents Signal. P operation will decrease the value by 1 every time and V operation will increase the value by 1 every time.

Q 5. Semaphores are mostly used to implement _____.

Answer: IPC mechanisms

Q 6. What is the use of a semaphore? (Infosys)

In computer science, a semaphore is a variable or abstract data type used to control access to a common resource by multiple processes and avoid critical section problems in a concurrent system such as a multitasking operating system.

Q 7. What are the main disadvantages of spinlock? (Capgemini)

The primary disadvantage of a spinlock is that, while waiting to acquire a lock, it wastes time that might be productively spent elsewhere.

Q 8. What is the difference between semaphore and mutex? (Infosys)

KEY DIFFERENCE

Semaphore supports wait and signal operations modification, whereas Mutex is only modified by the process that may request or release a resource. Semaphore value is modified using wait () and signal () operations, on the other hand, Mutex operations are locked or unlocked.

Q 9. Can Semaphore be negative? (Amazon)

A semaphore is an integer with a difference. If the resulting semaphore value is negative, the calling thread or process is blocked, and cannot continue until some other thread or process increments it.

Q 10. Which is faster semaphore or mutex? (Adobe)

Whereas semaphore can be used across process space and hence it can be used for interprocess synchronization. ii) Mutex is lightweight and faster than semaphore. Futex is even faster. iii) Mutex can be acquired by the same thread successfully multiple times with condition that it should release it the same number of times.

[Previous](#)

[Next](#)

[Peterson's Solution Notes](#)

Peterson’s Solution

Peterson’s Solution

It is restricted to two processes that alternate the execution between their critical and remainder sections.

Consider that the LOAD and STORE instructions are atomic; that is, cannot be interrupted.

The two processes share two variables:

- * int turn;
- * Boolean flag[2]

The variable turn indicates whose turn it is to enter the critical section.

The flag array is used to indicate if a process is ready to enter the critical section.

flag[i] = true implies that process Pi is ready!

Note:- Peterson’s Solution is a software-based solution.

Algorithm for Process Pi

```
while (true)

{

    flag[i] = TRUE;

    turn = j;

    while ( flag[j] && turn == j);

    CRITICAL SECTION flag[i] = FALSE;

    REMAINDER SECTION

}
```

do

```
{

    acquire lock

    Critical Section

    release lock

    Remainder Section

}

while (True);
```

Note:-Race Conditions are prevented by protecting the critical region by the locks.

Two Process Working Concurrently

Process 1

```
do
{
    flag1 = TRUE;
    turn = 2;
    while (flag2 && turn == 2);
    critical section.....
    flag1 = FALSE;
    remainder section.....
} while (1)
```

Process 2

```
do
{
    flag2 = TRUE;
    turn = 1;
    while (flag1 && turn == 1);
    critical section.....
    flag2 = FALSE;
    remainder section.....
} while (1)
```

Shared Variables }--- flag1, flag2, turn

Example

Process 0:

flag[0] := TRUE
turn := 1
check (flag[1] = TRUE and turn = 1)
*Condition is false because flag[1] = FALSE
* Since condition is false, no waiting in while loop
* Enters the critical section

Phase-1

Process 0:

*Leaves critical section
Sets flag[0] := FALSE
* Start executing the remainder (anything else a process does besides using the critical section)
* Process 0 happens to lose the processor

Phase-3

Process 1:

flag[1] := TRUE
turn := 0
check (flag[0] = TRUE and turn = 0)
*Since condition is true, it keeps busy waiting until it loses the processor *Process 0 resumes and continues until it finishes in the critical section

Phase-2

Process 1:

check (flag[0] = TRUE and turn = 0)
* This condition fails because flag[0] = FALSE
* No more busy waiting
*Enter the critical section

Phase-4

Implementation

```
public class cSection {  
  
    int turn;  
  
    boolean flag[] = new boolean[2];    int i = 0, j = 1;  
  
    // CSC variables  
  
    int counter = 0;// counter for giving processes an upper bound  
  
    int cscVar = 13;
```

```
private class ProcessI extends Thread { // process thread for i

@Override

public void run() {

try {

do {

flag[i] = true;  turn = j;

while (flag[j] && turn == j)

; // wait for j to finish

// critical section

System.out.println("I is in critical section");  cscVar++;

System.out.println(cscVar);  counter++;

System.out.println("counter is " + counter + "n ");

//

flag[i] = false;

// remainder section

} while (counter < 100); // 100 is upper bound, to remove

// infinite looping

}

catch (Exception ex) {  ex.printStackTrace();
```



```
}
```

```
}
```

```
}
```

```
private class ProcessJ extends Thread { // process thread for j
```

```
@Override
```

```
public void run() {
```

```
try {
```

```
do {
```

```
flag[j] = true; turn = i;
```

```
while (flag[i] && turn == i)
```

```
; // wait for i to finish
```

```
// critical section
```

```
System.out.println("J is in critical section"); cscVar--;
```

```
System.out.println(cscVar); counter++;
```

```
System.out.println("counter is " + counter + "n ");
```

```
//
```

```
flag[j] = false;
```

```
// remainder section
```

```
} while (counter < 100); // 100 is upper bound, to remove
```

```
// infinite looping

}

catch (Exception ex) {    ex.printStackTrace();

}

}

}

}

public cSection() {

System.out.println("Starting Threads/Processes");    Thread I = new ProcessI();

Thread J = new ProcessJ();    I.start(); // start process i    J.start(); // start process j

}

public static void main(String[] args) {    cSection cSec = new cSection();

}

}
```

[Previous](#)

[Next](#)

[Interview Questions](#)

Interview Questions

1. What is Peterson's solution?

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Peterson's solution**”.

2.How many processes can be synchronized with Peterson's solution? (Adobe)

Two processes

[Previous](#)

[Next](#)

[Test and Set Solution Notes](#)

Test and Set Solution for Critical Section Problem

In this, shared key variables are initialized to false. The TestAndSet (lock) algorithm works this way - it always returns any value sent to it and sets the lock to reality. The first process will go into a critical phase simultaneously as TestAndSet (lock) will return false and exit the time loop. Some operations cannot enter now as the lock is set to true so while the loop remains true. When the first process comes out of a critical phase, the lock is converted into a false one. So, now some processes can go in one step at a time. There is progress. However, after the initial procedure, any procedure can be entered. No queue has been saved, so any new system that detects whether a key is false again may be incurred. The restricted wait is not guaranteed.

TestAndSet Instruction

Definition:

```
boolean TestAndSet (boolean *target) @{ @ boolean rv = *target;@ *target = TRUE;@ return rv:@ }@ Solution using TestAndSet Shared boolean variable lock., initialized to false.
@Solution:@ do  i{@ while ( TestAndSet (&lock )) ; // do nothing // critical section @lock = FALSE; // remainder section @} while (TRUE);
```

Implementation

This algorithm solves the critical section problem for n processes using a Test and Set instruction (called TaS here). This instruction does the following function atomically:

```
function TaS(var Lock: boolean): boolean;
begin
    TaS := Lock;
```

```
    Lock := true;
end;
```

The solution is:

```
1 var waiting: shared array [0..n-1] of boolean;
2   Lock: shared boolean;
3   j: 0..n-1;
4   key: boolean;
5   ...
5 repeat    (* process Pi *)
6   waiting[i] := true;
7   key := true;
8   while waiting[i] and key do
9     key := TaS(Lock);
10  waiting[i] := false;
11  (* critical section goes here *)
12  j := i + 1 mod n;
13  while (j <> i) and not waiting[j] do
14    j := j + 1 mod n;
15  if j = i then
16    Lock := false
17  else
18    waiting[j] := false;
19 until false;
```

lines 1-2: These are global to all processes, and are all initialized to false.

lines 3-4: These are local to each process i and are uninitialized.

lines 5-10: This is the entry section. Basically, waiting[i] is true as long as process i is trying to get into its critical section; if any other process is in that section, then Lock will also be true, and process i will loop in lines 8-9. Once the process i can go on, it is no longer waiting for permission to enter, and sets waiting[i] to false (line 10); it then proceeds into the critical section. Note that Lock is set to true by the TaS instruction in line 9 that returns false.

lines 12-18: This is the exit section. When process i leaves the critical section, it must choose which other waiting process may enter next. It starts with the process with the next higher index (line 12). It checks each process to see if that process is waiting for access (lines 13-14); if no-one is, it simply releases the lock (by setting Lock to false;

lines 15-16). However, if some other process process j is waiting for entry, process i simply changes waiting[j] to false to allow process j to enter the critical section (lines 17-18).

Previous

Next

Interview Questions

Interview Questions

1. What is a test and set instruction? (Amazon)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Test and set solution**”.

2. What is the drawback in test and set lock solutions? (Vmware)

It does not guarantee bounded waiting and may cause starvation. It suffers from spinlock. It is not architecturally neutral since it requires the operating system to support test-and-set instruction. It is a busy waiting solution that keeps the CPU busy when the process is actually waiting.

Previous

Next

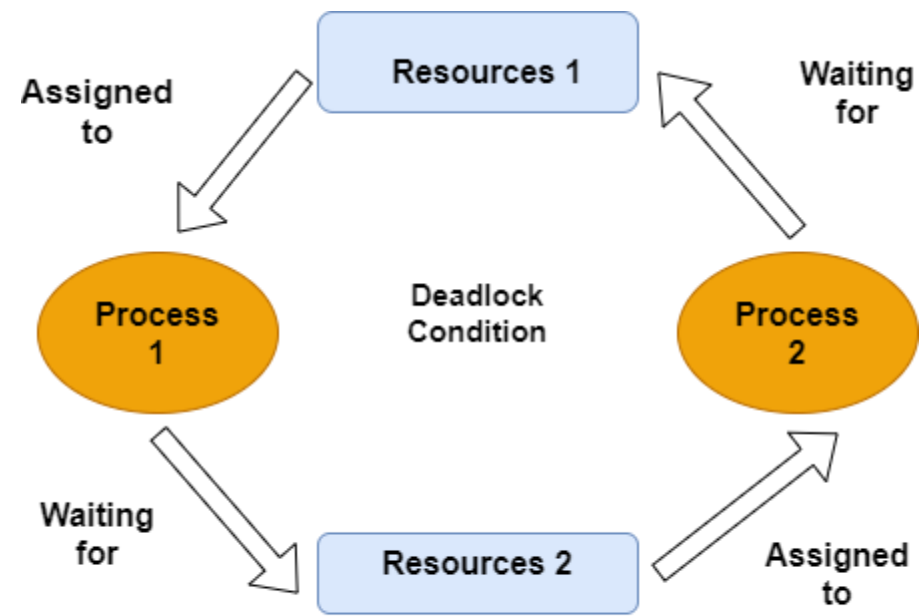
Deadlock Notes

Deadlock

In a multiprogramming system, several processes may compete for a finite number of resources. A process requests resources, and if the resources are not available at the time then the process enters the waiting state. Sometimes, a process will wait indefinitely because the resources it has requested are being held by other similar waiting processes.

Deadlock is a state in which two or more processes are waiting indefinitely because the resources they have requested are being held by one another.

A process is deadlocked if it is waiting for an event that is never going to happen. Deadlocks can occur via system calls, locking, etc.



Deadlock In Os

Example of deadlock

Let S and Q be two semaphores initialized to 1

P0	P1
wait (S);	wait (Q);
wait (Q);	wait (S);
.	
.	
.	
signal (S);	signal (Q);
signal (Q);	signal (S);

Next

Interview Questions

Interview Questions

Q 1. WHAT IS DEADLOCK? (Vmware)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “Deadlock”.

Q 2. What are Real life examples of deadlock? (Amazon)

Real-life Example

- Bridge traffic can only be in one direction.
- Each entrance of the bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (resource preemption).
- Starvation is possible.

Previous

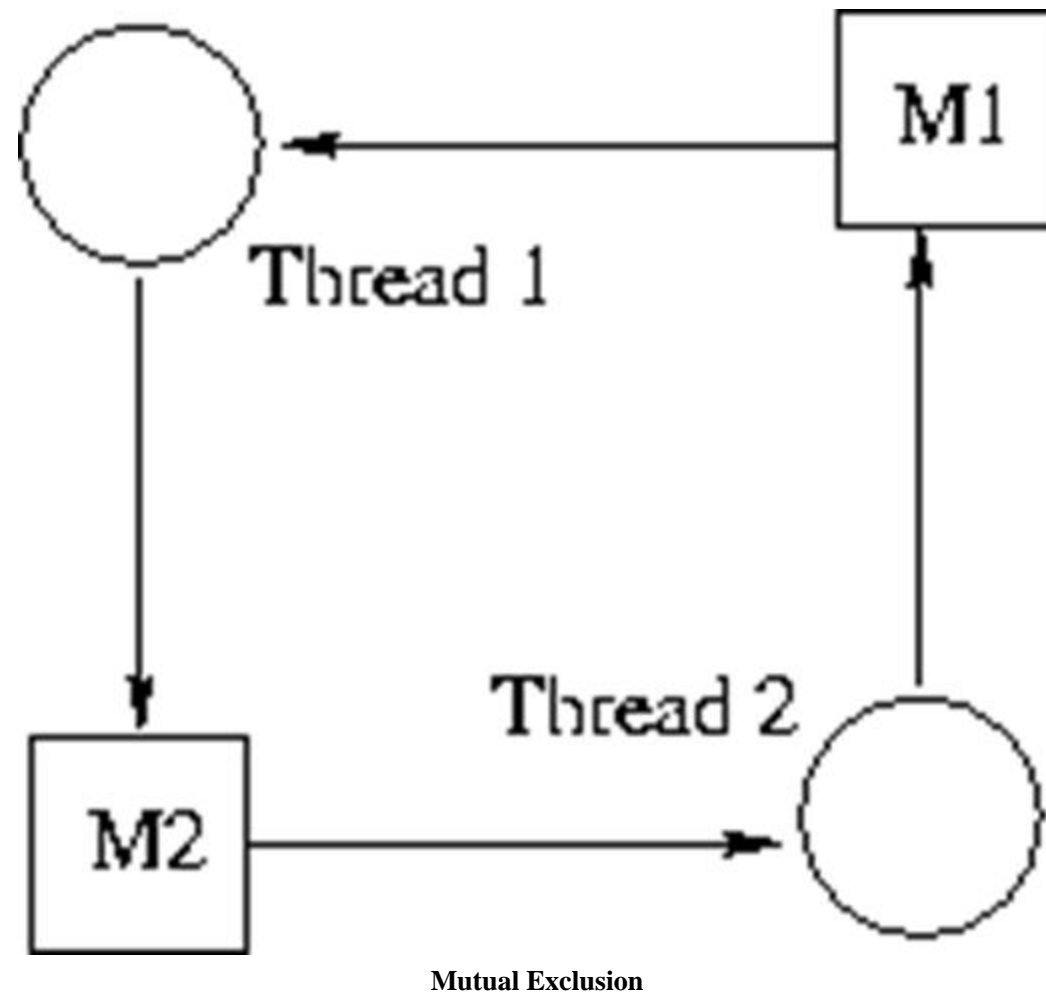
Next

Condition For Deadlock Notes

Conditions for Deadlock

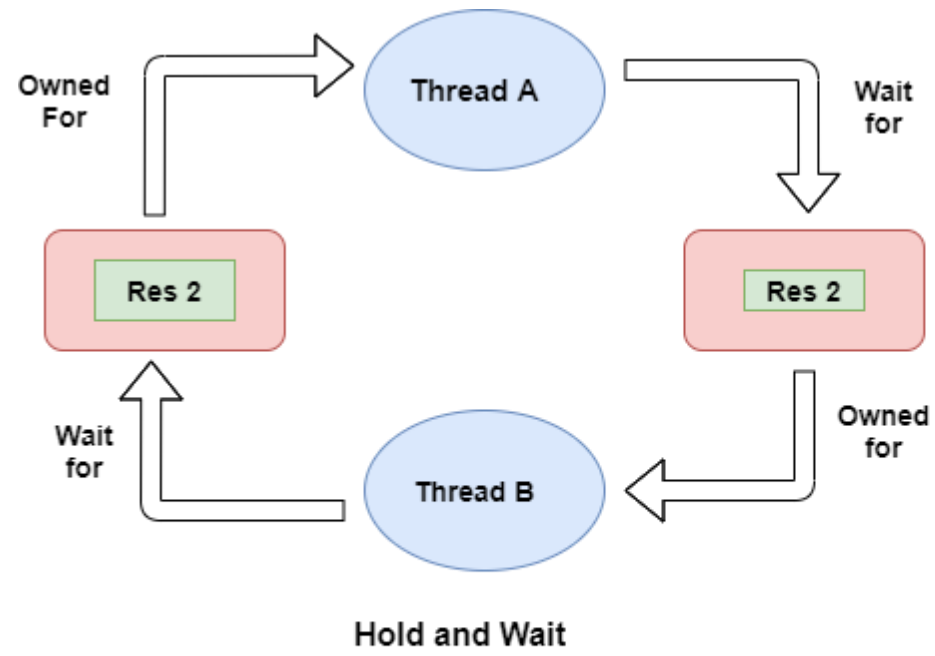
Mutual Exclusion

- The resources involved must be unshareable.
- Every resource is either currently allocated to exactly one processor it is available. (Two processes cannot simultaneously control the same resource).



Hold and Wait

There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.



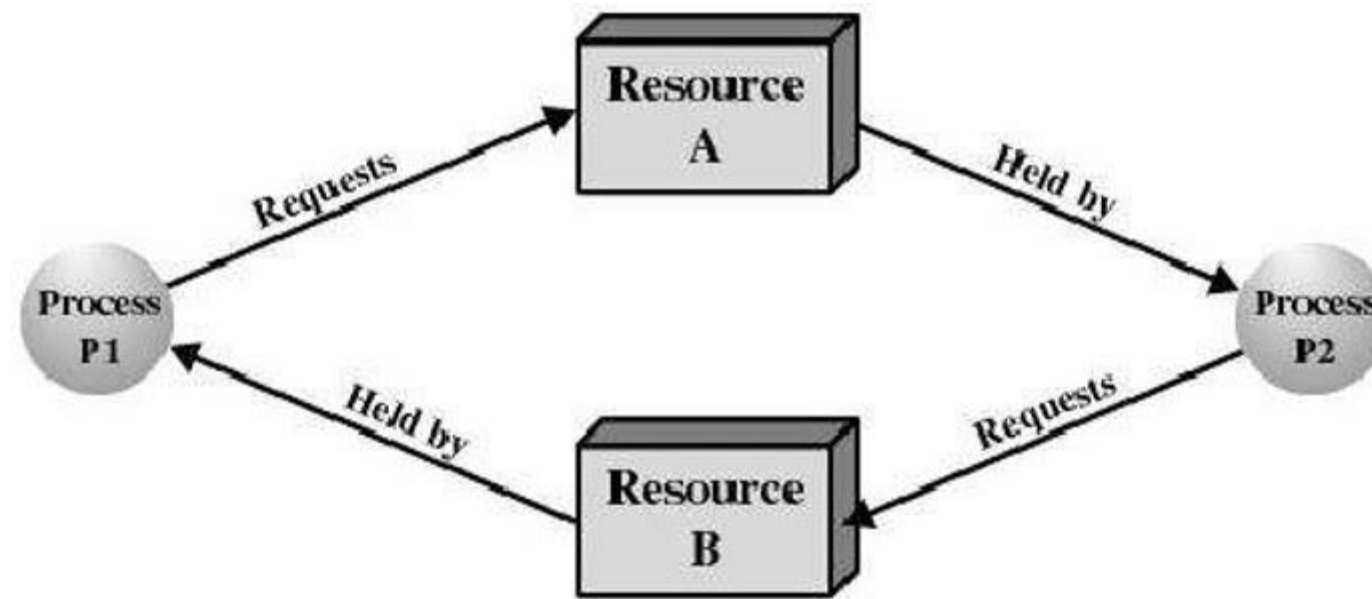
Text

No Preemption Condition

If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released implicitly. Then the preempted resources are added to the list of resources for which the process is waiting.

Circular Wait

In circular wait, a chain of processes exists in which each process waits for one or more resources held by the next process in the chain.



Circular Wait

[Previous](#)
[Next](#)
[Interview Questions](#)

Interview Questions

Q 1. What is the mutual exclusion condition?

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Mutual Exclusion**”.

Q 2. Explain circular wait conditions. (Vmware)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**circular wait**”.

Q 3. What happens in the hold and wait for conditions? (Wipro)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Hold and Wait**”.

Q 4.What are the 4 necessary conditions for a deadlock to occur?

(Amazon)

- Mutual Exclusion.
- Hold and Wait.
- No Preemption.
- Circular Wait.

[Previous](#)

[Next](#)

[Methods Of Deadlock Handling Notes](#)

Methods Of Deadlock Handling

Banker's Algorithm

This algorithm can be used in a bank to ensure that the bank never allocates the available money in a way that it could no longer satisfy the needs of all its clients. A new task must declare the maximum number of instances of each resource type that it may need. This number should not exceed the total number of instances of that resource type in the system.

When a process requests a set of resources, the system must determine whether allocating these resources will leave the system in a safe state. If yes, then the resources may be allocated to the process. If not, then the process must wait till other processes release enough resources.

Data Structures for the Banker's Algorithm

Available: Vector of length m. If $available[j] = k$, there are k instances of resource type R_j available

Max: $n \times m$ matrix. If $Max[i,j] = k$, then process P_i may request at most k instances of resource type R_j

Allocation: $n \times m$ matrix. If $Allocation[i,j] = k$ then P_i is currently allocated k instances of R_j

Need: $n \times m$ matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task $Need[i,j] = Max[i,j] - Allocation[i,j]$

Finish: Boolean value, either true or false. If $finish[i]=true$ for all i return safe else unsafe

Safety Algorithm

1.Let Available and Finish be vectors of length m and n, respectively.

Initialize: $Finish[i] = false$ for $i = 0, 1, \dots, n-1$

2.Find an i such that both: (a) $Finish[i] = false$ (b) $Need_i \leq Available$ If no such i exists, go to step 4

3. $Available = Available + Allocation$ $Finish[i] = true$ go to step 2
4. If $Finish[i] == true$ for all i , then the system is in a safe state

Resource-request algorithm checks if a request can be safely granted. P_i is requesting for more resources and $Request[m]$ be the request.

1. If $Request > Need[i]$, then error
2. If $Request > Available$, then wait
3. Pretend to allocate the request

$Available = Available - Request$

$Allocation[i] = Allocation[i] + Request$

$Need[i] = Need[i] - Request$

If the resultant state is safe then the resources are actually allocated, else values of $Available$, $Allocation[i]$ and $Need[i]$ are restored to their previous values.

Time complexity = $O(m)$.

Ostrich Algorithm

- The simplest approach is the ostrich algorithm: stick your head in the sand and pretend there is no problem at all.
- Different people react to this strategy in different ways. Mathematicians find it totally unacceptable and say that deadlocks must be prevented at all costs.
- Engineers ask how often the problem is expected, how often the system crashes for other reasons, and how serious a deadlock is.
- If deadlocks occur on average once every five years, but system crashes due to hardware failures, compiler errors, and operating system bugs occur once a week, most engineers would not be willing to pay a large penalty in performance or convenience to eliminate deadlocks.
- Most operating systems, including UNIX and Windows, just ignore the problem on the assumption that most users would prefer an occasional deadlock to a rule restricting all users to one process, one open file, and one of everything.
- If deadlocks could be eliminated for free, there would not be much discussion. The problem is that the price is high, mostly in terms of putting inconvenient restrictions on processes, as we will see shortly. Thus we are faced with an unpleasant trade-off between convenience and correctness, and a great deal of discussion about which is more important, and to whom. Under these conditions, general solutions are hard to find.

Resource preEmption

To eliminate deadlocks using resource preemption, preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

There are 3 methods to eliminate the deadlocks using resource preemption. These are

- a) SELECTING A VICTIM: Select a victim resource from the deadlock state and preempt that one.
- b) ROLLBACK: If a resource from a process is preempted, what should be done with that process. The process must be rolled back to some safe state and restart from that state.
- c) STARVATION: It must be guaranteed that resources will not always be preempted from the same process to avoid starvation problems.

[Previous](#)
[Next](#)
[Interview Questions](#)

Interview Questions

- Q 1. Explain Banker’s Algorithm. (Amazon)**
Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Banker’s Algorithm**”.
- Q 2. Explain resource preemption**
Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Resource Preemption**”.
- Q 3. What is an ostrich algorithm (Vmware)**
Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Ostrich Algorithm**”.

[Previous](#)
[Next](#)
[Memory Management Notes](#)

Memory Management

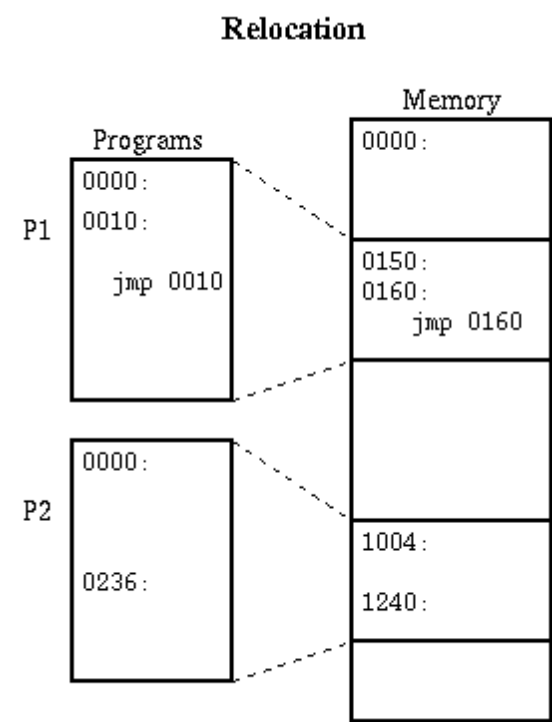
Logical and Physical Address Space

The logical address is the one that is generated by the CPU, also referred to as the virtual address. CPU also referred to as virtual address. The program perceives this address space. program perceives this address space. Logical address space is the set of all logical addresses generated by a program.

A physical address is an actual address understood by computer hardware i.e., memory understood by computer hardware i.e., memory unit. Logical to physical address translation unit. Physical address space is the set of all physical addresses generated by a program.

Program Relocation

The term program relocatability refers to the ability to load and execute a given program into memory. Relocation is a way to map virtual addresses into physical addresses. When a task is loaded in the main memory since there are several instructions inside the process so here addresses of these different instructions inside the process are relocatable addresses which are converted into actual addresses by the loader in the software approach. But there is some problem with this approach: suppose if a process is first loaded then removed and then after loaded again so in this situation the loader will get confused. So to avoid this problem the operating system uses another method for relocation. Instead of using this load time binding operating system used runtime binding.



Static Relocation

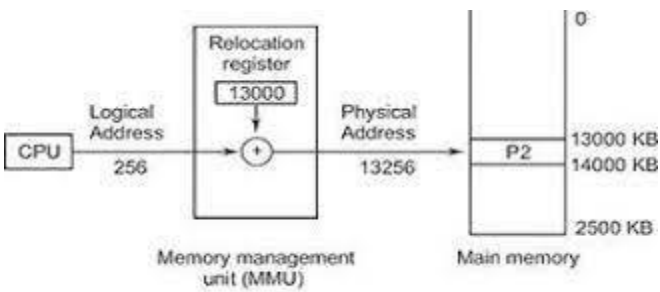
At load time, the OS adjusts the addresses in a process to reflect its position in memory. This method is a slow process because it involves software translation. It is used only once before the initial loading of the program. Once a process is assigned a place in memory and starts executing it, the OS cannot move it.

Dynamic Relocation:

hardware adds relocation register (base) to virtual address to get a physical address. hardware compares address with limit register (address must be less than base). If the test fails, the processor takes an address trap and ignores the physical address

Relocation Register

Relocation register is a special register in the CPU used for program relocation means mapping of logical addresses used by the program to physical addresses of the system's main memory.



For example, if the base(relocation register content) is at 14000, then an attempt by the user to address location 0 is dynamically relocated to location 14000, and access to location 346 is mapped to location 14346.

[Previous](#)

[Next](#)

[Interview Questions](#)

Interview Questions

Q 1.What is a Physical Address? (TCS)

A physical address is an actual address understood by computer hardware i.e., memory understood by computer hardware i.e., memory unit.

Q 2.What is a Logical Address? (Infosys)

The logical address is the one that is generated by the CPU, also referred to as the virtual address. CPU also referred to as virtual address.

Q 3.Explain the term Relocation. (Amazon)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Program Relocation**”.

Q 4.Types of relocation

[Previous](#)

[Next](#)

[Storage Allocation Notes](#)

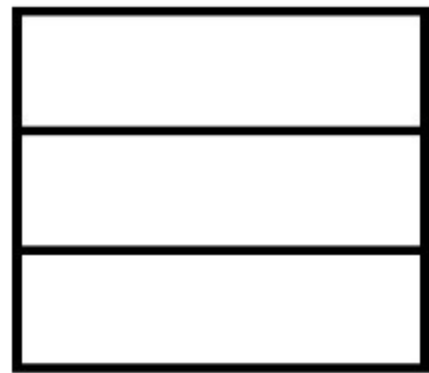
Storage Allocation Methods in Memory Management

Storage Allocation and Management TechniquesThe Storage allocation can be of two types:

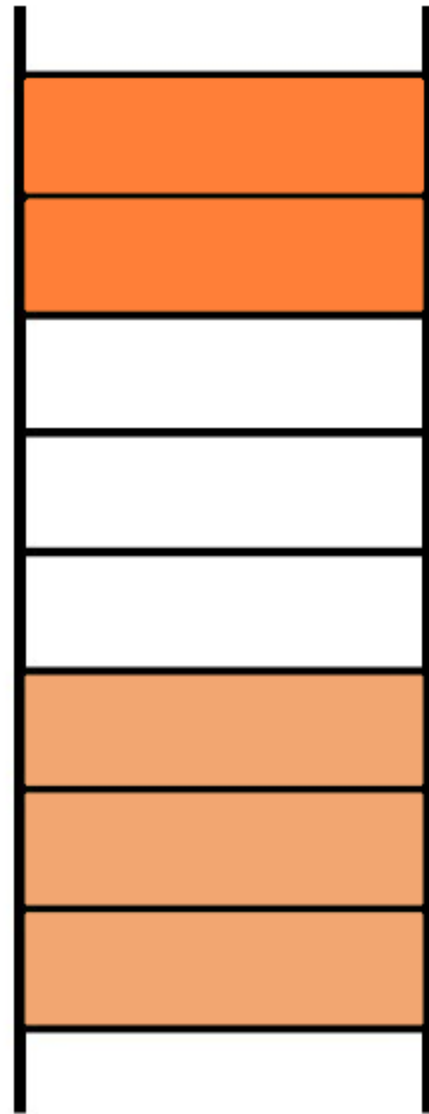
- (i) Contiguous storage allocation.
- (ii) Non-contiguous storage allocation.

Contiguous Storage Allocation

- ▶ Contiguous storage allocation implies that a program's data and instructions are assumed to occupy a single contiguous memory area.
- ▶ It is further subdivided into a Fixed-partition storage allocation strategy and variable-partition storage allocation strategy.



Process



Memory blocks

Contiguous Memory Allocation

Contiguous: Divide the memory into fixed-sized blocks

1. Fixed-partition contiguous storage allocation

The processes with small address space use small partitions and processes with large address space use large partitions. This is known as fixed partition contiguous storage allocation.

2. Variable - partition contiguous storage allocation

This notion is derived from the parking vehicles on the sides of streets where the one who manages to enter will get the space. Two vehicles can leave a space between them that cannot be used by any vehicle. This means that whenever a process needs memory, a search for the space needed by it, is done. If contiguous space is available to accommodate that process, then the process is loaded into memory.

External Fragmentation

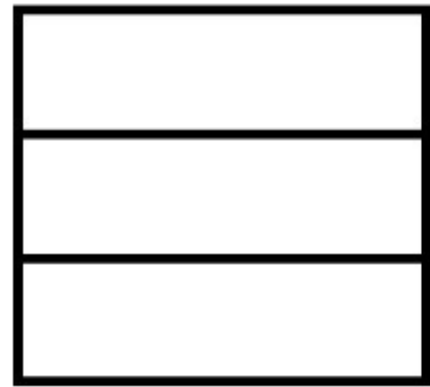
This phenomenon of entering and leaving the memory can cause the formation of unusable memory holes (like the unused space between two vehicles). This is known as External Fragmentation.

Non-contiguous Storage Allocation

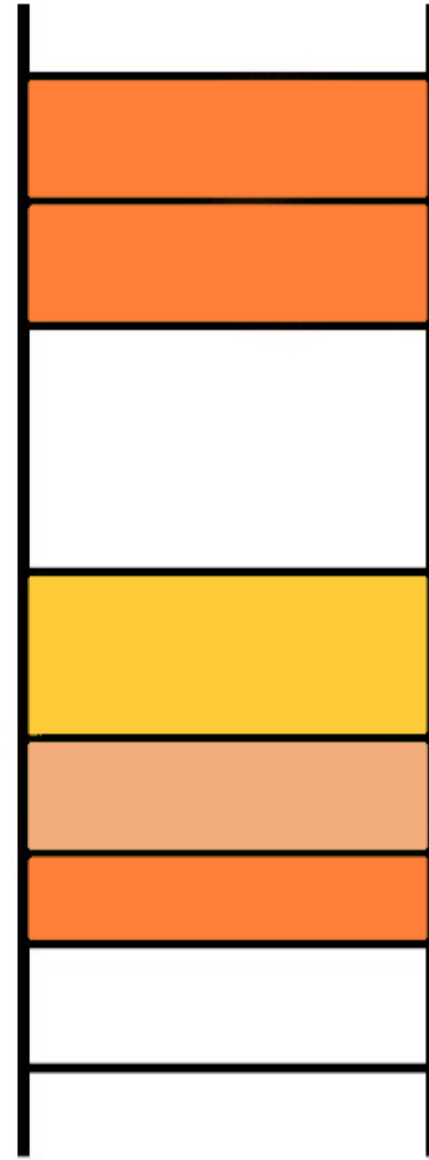
To resolve the problem of external fragmentation and to enhance the degree of multiprogramming to a greater extent, it was decided to sacrifice the simplicity of allocating contiguous memory to every process. It was decided to have a non-contiguous physical address space of a process so that a process could be allocated memory wherever it was available.

There are 2 techniques for non-contiguous allocation:

1. Paging
2. Segmentation



Process



Memory blocks

Non-Contiguous Memory
Allocation

Contiguous Memory Allocation

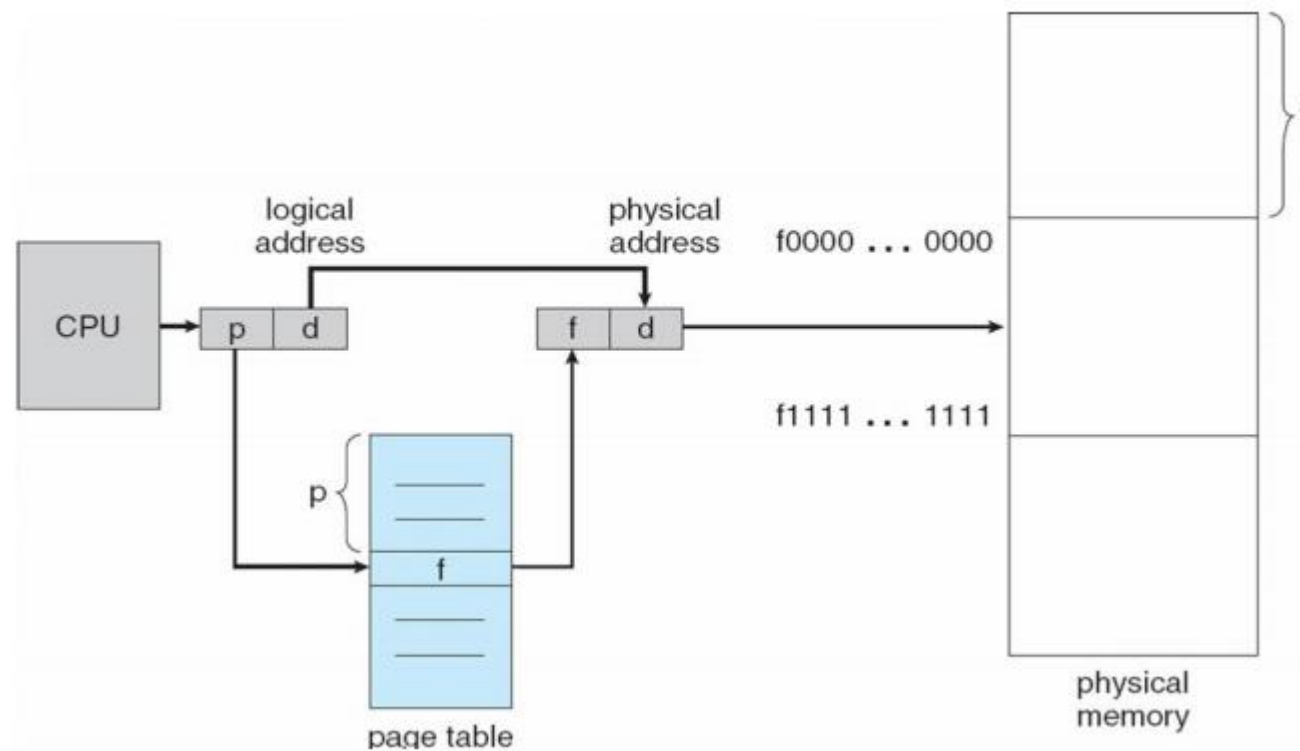
Paging

In paging, physical memory is divided into fixed-size blocks called frames and the logical memory is divided into fixed-sized blocks called pages.

The size of a page is the same as that of a frame. The key idea of this method is to place the pages of a process into the available frames of memory, whenever, this process is to be executed. The address mapping is done by Page table.

Physical memory is divided into fixed-size- blocks called FRAMES. (size is the power of 2, for example, 512 bytes)

Logical memory is divided into blocks of the same size called PAGES.



Example

Suppose, if the main memory size is 16 KB and the Frame size is 1 KB. Here, the main memory will be divided into a collection of 16 frames of 1 KB each.

Advantages and Disadvantages of paging

Advantages :

no external fragmentation.

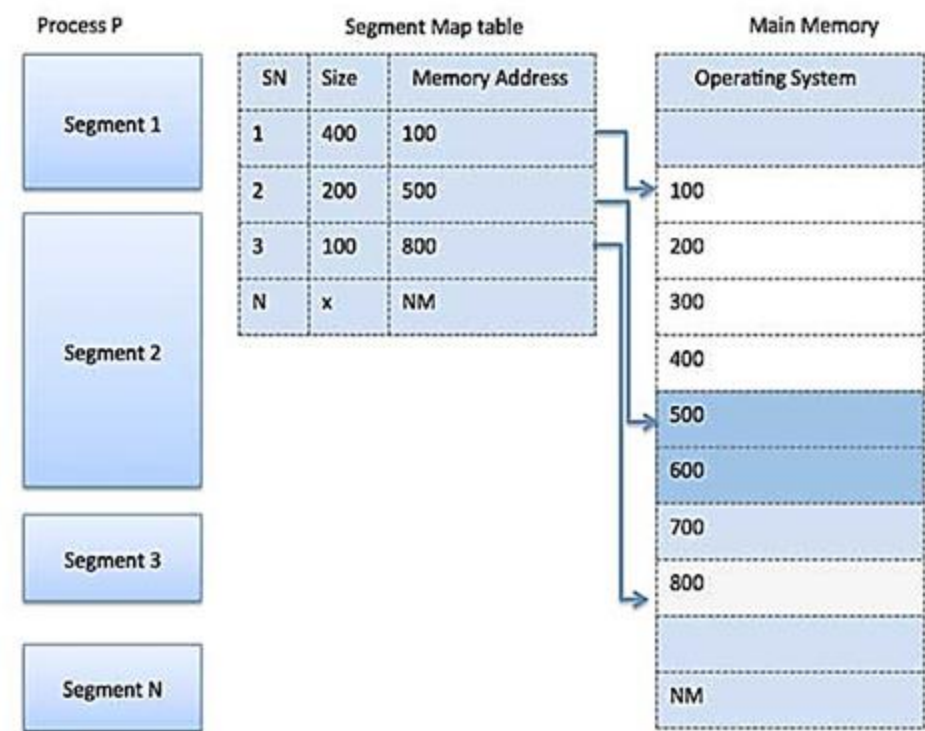
Simply memory management algorithm.
Swapping is easy(equal-sized pages and page frames).

Disadvantages:

Internal fragmentation.
Page tables may consume more memory.

Segmentation

segmentation is another technique for noncontiguous storage allocation. It is different from paging as it supports users' view of his program.
For a programmer, it might be more relevant to divide the logical address space of his program into variable-sized segments (with respect to his view of the main program, subroutines, data, etc.) than to divide it into fixed-size pages.
Such variable-sized segments, which are a collection of logically related information, are the basis of the segmentation technique.



segmentation

For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.

Advantages and disadvantages of segmentation

Advantages :

no internal fragmentation.

Segment tables consume less memory than pages.

Lends itself to sharing data among processes.

Lends itself to protection.

Disadvantages:

Costly memory management algorithm.

Previous

Next

Interview Questions

Interview Questions

Q 1.What is paging. (AMAZON)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**paging**”.

Q 2.Advantages of Paging (VMWARE)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**Paging**”.

Q 3.Explain contiguous memory Allocation. (ADOBE)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “**contiguous memory Allocation.**”.

Previous

Next

Paging Notes

Paging

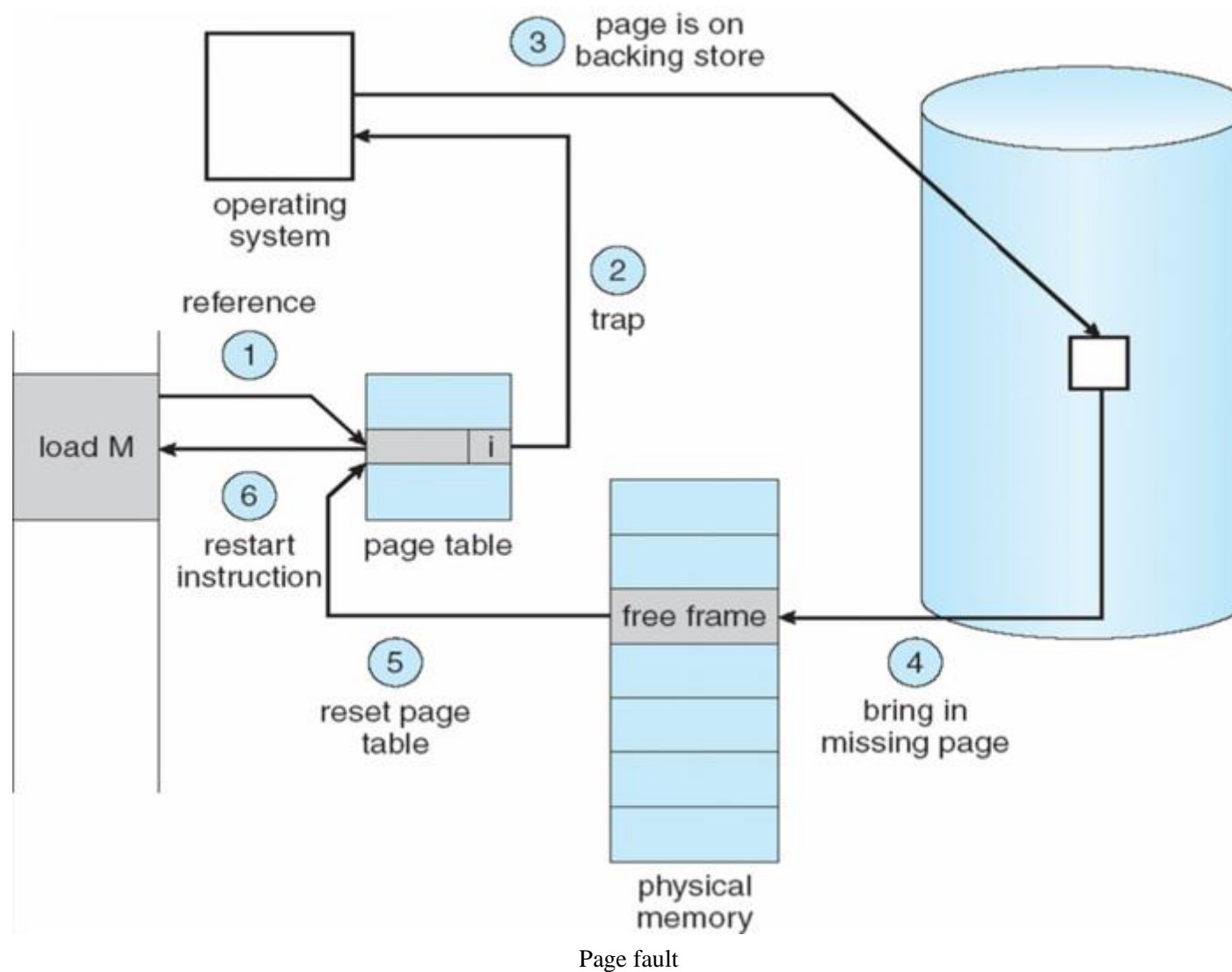
The memory Management technique that permits the physical address space of a process to be non-contiguous is known as paging. Paging is used for faster access to data, and it is a logical concept.

Virtual memory

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. The paging technique plays an important role in implementing virtual memory.

Page faults

Page fault dominates like an error. If any program tries to access a piece of memory but which does not exist in physical memory, meaning main memory, then page fault will occur. The fault specifies the O/S that it must trace all data into virtual memory management, and after that moves it from secondary memory like a hard disk to the primary memory of the system.



Handling of a Page Fault

1. Check the location of the referenced page in the PMT
2. If a page fault occurred, call on the operating system to fix it
3. Using the frame replacement algorithm, find the frame location
4. Read the data from disk to memory
5. Update the page map table for the process
6. The instruction that caused the page fault is restarted when the process resumes execution.

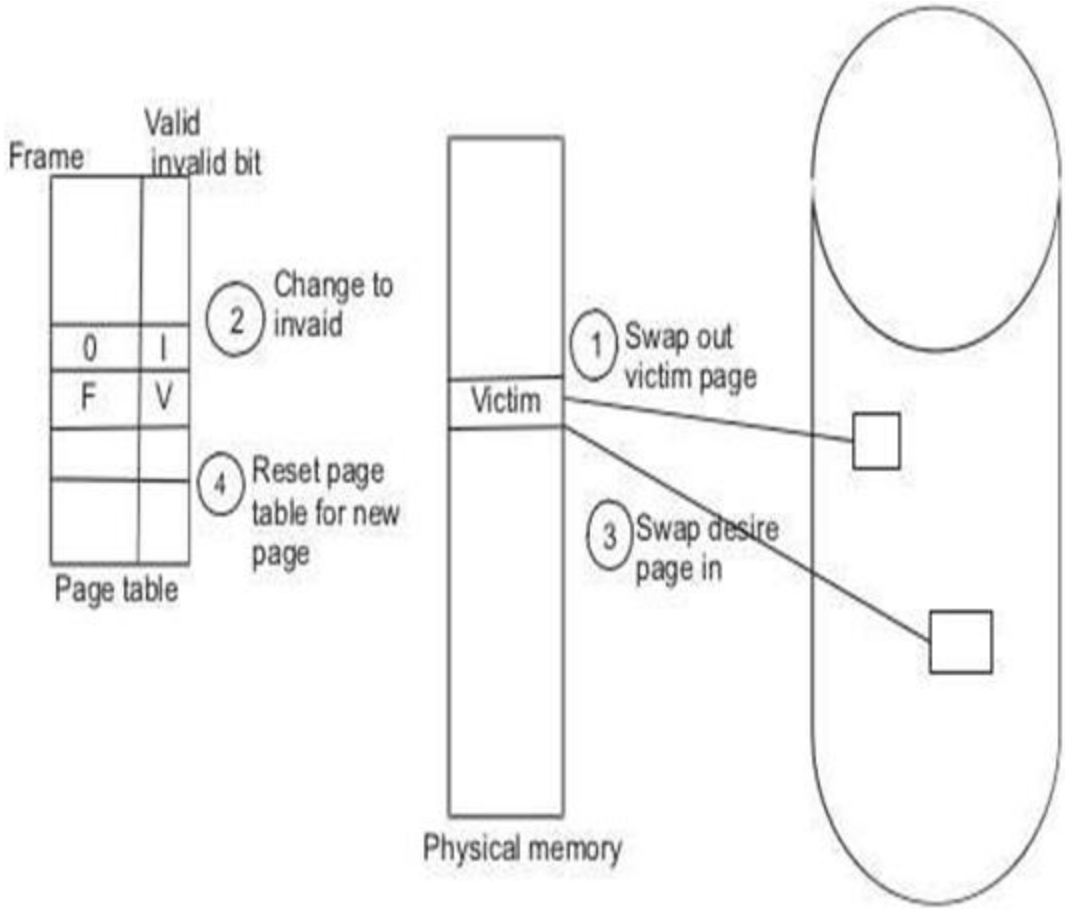
Page Replacement Algorithm

What is page Replacement?

When memory located in secondary memory is needed, it can be retrieved back to the main memory.

Process of storing data from main memory to secondary memory ->swapping out

Retrieving data back to main memory ->swapping in



Page Replacement

Why do we need a page replacement algorithm?

The main goal of page replacement algorithms is to provide the lowest page fault rate

Algorithms

- First In First Out
- Optimal Replacement
- Not Recently Used
- Second Chance
- CLOCK
- Not Frequently Used
- Least Recently Used
- Random Replacement

First-In-First-Out (FIFO)

- Pages in main memory are kept in a list.
- The newest page is in head and the oldest in the tail.
- It does not take advantage of page access patterns or frequency.

Optimal Replacement (OPT)

- When the memory is full, evict a page that will be unreferenced for the longest time.
- The OS keeps track of all pages referenced by the program.
- Only if the program's memory reference pattern is relatively consistent.

Not Recently Used (NRU)

- It favors keeping pages in memory that have been recently used.
- The OS divides the pages into four classes based on usage during the last clock tick:
 3. Referenced, modified
 2. Referenced, not modified
 1. Not referenced, modified
 0. Not referenced, not modified
- Pick a random page from the lowest category for removal,i.e. the not referenced, not modified page

Second Chance

- A modified version of FIFO

- Instead of swapping out the last page, the referenced bit is checked
- Gives every page a "second chance"

Clock

- A modified version of FIFO
- The set of frame candidates for replacement is considered as a circular buffer.

Least Recently Used (LRU)

- It swaps the pages that have been used the least over a period of time.
- It is free from Belady's anomaly

Not frequently used (NFU)

- This page replacement algorithm requires a counter
- The counters keep track of how frequently a page has been used
- The page with the lowest counter can be swapped out

Random

- This algorithm replaces a random page in memory.
- It fares better than FIFO.

Previous

Next

Interview Questions

Interview Questions

Q 1.What is virtual memory? (TCS, AMAZON)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “virtual memory”.

Q 2. Explain page fault. (ADOBE)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “ page fault”.

Q 3. What happens in optimal replacement Algorithm

Please refer to the notes for definitions of the aforementioned operating systems under the heading “optimal replacement Algorithm”.

Previous

Next

Disk Scheduling Algorithm Notes

Disk Scheduling Algorithm

First Come First Serve (FCFS)

It is the simplest form of disk scheduling algorithms. The I/O requests are served or processed according to their arrival. The request arrives first and will be accessed and served first.

Example: Given the following track requests in the disk queue, compute for the Total Head Movement2 (THM) of the read/write head: 95, 180, 34, 119, 11, 123, 62, 64 .

Consider that the read/write head is positioned at location 50. Prior to this track location 199 was serviced. Show the total head movement for a 200 track disk (0-199).

Solution:

Total Head Movement Computation:

(THM) = (180 - 50) + (180-34) + (119-34) + (119-11) + (123-11) + (123-62) + (64-62)

= 130 + 146 + 85 + 108 + 112 + 61 + 2

(THM) = 644

tracks Assuming a seek rate of 5 milliseconds is given, we compute for the seek time using the formula:

Seek Time = THM * Seek rate

= 644 * 5 ms

Seek Time = 3,220 ms

There are some requests that are far from the current location of the R/W head which causes the access arm to travel from innermost to the outermost tracks of the disk or vice versa. In this example, it had a total of 644 tracks and a seek time of 3,220 milliseconds. Based on the result, this algorithm produced a higher seek rate since it follows the arrival of the track requests.

Shortest Seek Time First (SSTF)

This algorithm is based on the idea that the R/W head should proceed to the track that is closest to its current position. The process would continue until all the track requests are taken care of.

Example: Given the following track requests in the disk queue, compute for the Total Head Movement² (THM) of the read/write head: 95, 180, 34, 119, 11, 123, 62, 64

Consider that the read/write head is positioned at location 50. Prior to this track location 199 was serviced. Show the total head movement for a 200 track disk (0-199).

Solution:

(THM) = 12+2+30+23+84+24+4+57

(THM) = 236 tracks

Seek Time = THM * Seek rate

= 236 * 5ms

Seek Time = 1,180 ms

In this algorithm, the request is serviced according to the next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue up to the last track request. There are a total of 236 tracks and a seek time of 1,180 ms, which seems to be a better service compared with FCFS.

SCAN

This algorithm is performed by moving the R/W head back and forth to the innermost and outermost track. As it scans the tracks from end to end.

Example: Given the following track requests in the disk queue, compute for the Total Head Movement² (THM) of the read/write head: 95, 180, 34, 119, 11, 123, 62, 64

Solution:

$$(THM) = (50-0) + (180-0) = 50 + 180$$

$$(THM) = 230$$

$$\text{Seek Time} = THM * \text{Seek rate}$$

$$= 230 * 5\text{ms}$$

$$\text{Seek Time} = 1,150 \text{ ms}$$

This algorithm works like an elevator does.

In the algorithm example, it scans down towards the nearest end and when it reaches the bottom it scans up servicing the requests that it did not get going down. If a request comes in after it has been scanned, it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks and a seek time of 1,150. This is optimal for the previous algorithm.

Look

The disk arm starts at the first I/O request on the disk, and moves toward the last I/O request on the other end, servicing requests until it gets to the other extreme I/O request on the disk, where the head movement is reversed and servicing continues.

It moves in both directions until both last I/O requests; more inclined to serve the middle cylinder requests.

Example: Given the following track requests in the disk queue, compute for the Total Head Movement² (THM) of the read/write head: 95, 180, 34, 119, 11, 123, 62, 64 ALI Solution:

$$(THM) = (50-11) + (180-11) = 39 + 169$$

$$(THM) = 208 \text{ tracks}$$

$$\text{Seek Time} = THM * \text{Seek rate}$$

$$= 208 * 5\text{ms}$$

Seek Time = 1,040 ms

This algorithm has a result of 208 tracks and a seek rate of 1,040 milliseconds. This algorithm is better than the previous algorithm.

C-Scan

The head moves from one end of the disk to the other, servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.

Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

Provides a more uniform wait time than SCAN; it treats all cylinders in the same manner.

C-Look

Look at version of C-Scan.

Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.

In general, Circular versions are fairer but pay with a larger total seek time.

Scan versions have a larger total seek time than the corresponding Look versions.

RSS

It stands for random scheduling and just like its name it is nature. It is used in situations where scheduling involves random attributes such as random processing time, random due dates, random weights, and stochastic machine breakdowns this algorithm sits perfectly. This is why it is usually used for analysis and simulation.

LIFO

In LIFO (Last In, First Out) algorithm, the newest jobs are serviced before the existing ones i.e. in order of requests that get serviced the job that is latest or last entered is serviced first, and then the rest in the same order.

Advantages

- Maximizes locality and resource utilization

Disadvantages

- Can seem a little unfair to other requests and if new requests keep coming in, it causes starvation to the old and existing ones.

N-STEP SCAN

It is also known as the N-STEP LOOK algorithm. In this, a buffer is created for N requests. All requests belonging to a buffer will be serviced in one go. Also once the buffer is full no new requests are kept in this buffer and are sent to another one. Now, when these N requests are serviced, the time comes for another top N requests, and this way all get requests to get a guaranteed service

Advantages

It eliminates starvation of requests completely

FSCAN

This algorithm uses two sub-queues. During the scan, all requests in the first queue are serviced and the new incoming requests are added to the second queue. All new requests are kept on halt until the existing requests in the first queue are serviced.

Advantages

FSCAN along with N-Step-SCAN prevents “arm stickiness” (phenomena in I/O scheduling where the scheduling algorithm continues to service requests at or near the current sector and thus prevents any seeking)

Previous

Next

Interview Questions

Interview Questions

Q 1. Different types of disk scheduling Algorithms. (Goldman Sachs)

Please refer to the notes for definitions of the aforementioned operating systems under the heading “Disk scheduling algorithm notes”.

Q 2. How FCFS works?

Please refer to the notes for definitions of the aforementioned operating systems under the heading “FCFS”.

Previous

Next