**CMR ENGINEERING COLLEGE**

**KANDLAKOYA (V), MEDCHAL (M), R.R.DIST.**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

SUBJECT - OPERATING SYSTEMS

STEP MATERIAL

**Q1. What are the services provided by an operating system? Discuss.**

**Ans.** An operating system provides an environment for the execution of programs. The operating system provides certain services to programs and to the users of those programs. The specific services provided will differ from one operating system to another, but there are some common classes that we can identify. These operating system services are provided for the convenience of the programmer, to make the programming task easier. Some of these services are listed below:

**1.Program execution services:** The system must be able to load a program into memory and run it.

**2. I/O operation services:** Every device is controlled by the operating system and not by any other device.

**3. File system services:** creation and deletion of files is provided by name.

**4. Communication services:** Communication is done in two ways. In parallel systems, communication is done through the shared memory. Whereas in distributed systems communication takes place through passing messages.

**5. Error detection services:** Various types of errors are detected by the operating system and it informs the user to take appropriate action.

**6. Accounting services:** Operating system takes care of the processor time taken by each process that is helpful in improving the existing system.

**7. Protection services:** Each user works securely by using passwords. This avoids unauthorized access.

**Q 2. Differentiate between multiprogramming and time-sharing systems.**

**Ans.** Multiprogramming is the name given to the execution of two or more different & independent programs by the same processor. Two or more user programs are kept in the main memory & executed concurrently. With multiple programs simultaneously residing in the main memory, when a program that was executing starts performing I/O operations, the CPU is allocated to another program in the main memory that is ready to use the CPU, instead of allowing the CPU to be idle. The CPU switches from one program to another. Hence, in multiprogramming, several programs share the time of the CPU to keep it busy.

Time-sharing is a mechanism to provide simultaneous use of a computer system by many users in such a way that each user is given the impression that he has his own computer. It uses multiprogramming with a special CPU scheduling algorithm to achieve this. This algorithm allocates a very short period of CPU time one by one to each user process, beginning from the first process and proceeding through the last one and then again beginning from the first one. This short period of time during which a user process gets the attention of the CPU is known as time slice or quantum. In other words, time-sharing system is an implementation of the multiprogramming system by using a special algorithm known as round robin.

**Q3. When are caches useful? What problems do they solve? What problems do they cause?**

**Ans.** Information is normally kept in some storage system such as main memory. As it is used, it is copied into a faster storage system known as cache, on a temporary basis. When we need a particular piece of information, we first check whether it is in the cache. If it is, we use the information directly from the cache. If it is not, we use the secondary memory & put a copy in the cache assuming that it will be needed again. Since caches have limited size, cache management is an important design problem.

In a hierarchical storage structure, the same data may appear in different levels of the storage. For example, consider an integer A located in file B that is to be incremented by 1. Suppose that File B resides on magnetic disk. The increment operation proceeds by first issuing on I/O operation to copy the disk block on which A resides to main memory. A copying of A to the cache & to an internal register follows this operation. Thus, the copy of A appears in several places, which can lead to inconsistencies. This problem is called the cache coherency problem.

**Q4. Explain Virtual memory.**

**Ans: -** We have seen that a program has to be loaded in the main memory before executing on a CPU. Therefore, a computer should have free physical memory at least as big as the program size. Virtual memory is a technique for executing a program whose memory requirement is larger than the available physical memory. The executable image of the program is created and stored in a secondary storage, called swap device. A part of this secondary storage image is loaded into main memory and the execution on CPU begins. As long as the CPU accessed memory locations are in the main memory the execution proceeds as usual. When the CPU tries to access a memory location outside the part loaded in memory, an illegal memory access interrupt occurs and the system execution branches to the operating system. The OS then determines the cause of the interrupt and executes interrupt routine which loads the missing block of the program memory into main memory possibly replacing the existing block. The interrupt handler then returns and the user program execution then proceeds from the point of interruption and completes the execution of the attempted memory access and continue the execution until the next illegal memory access interrupt, and so on.

**Q5. Explain the layered approach of O.S.**

**Ans.**

Layer N(User Interface)

…..

Layer 1

Layer 0(Hardware)

In the layered approach of Operating System, the operating system is broken up into a number of layers (levels). The bottom layer (Layer 0) is the hardware; the highest layer (Layer N) is the user interface. A typical operating system layer, say layer x consists of data structures and a set of routines that can be invoked by higher-level layers. Layer x can, in turn, can invoke operations on lower level layers.

Each layer is implemented with only those operations provided by lower level layers. A layer does need to know how these operations are implemented; it needs to know only what these operations do. Hence each layer hides the existence of certain data structures, operations and hardware from higher-level layers.

**Advantages:**

1. The main advantage is the simplicity of construction and debugging. It is due to modularity.

2.Layered structure provides abstraction, hiding complex data structures and operations from other layers.

**Disadvantages:**

1. It is difficult to appropriately define various layers.

2. On a large system, the CPU scheduler may have more information than can fit in memory. This will require swapping in and out of memory.

3. It is less efficient than other types.

**Q6. Explain the various steps of booting.**

**Ans.** For a computer to start running, when it is powered up or rebooted, it needs an initial program to run. This initial bootstrap program tends to be simple. It initializes all aspects of the system, from CPU registers to device controller & contents of the main memory & then starts the operating system. To do its job, it finds the operating system kernel on disk, loads the kernel into memory. For most computers, the bootstrap is stored in ROM. This location is convenient because ROM needs no initialization and is at a fixed location that the processor can start executing when powered up or reset. The full bootstrap program is stored in a partition called the boot blocks, at a fixed location on the disk. The code in the boot ROM instructs the disk controller to read the boot blocks into memory & then starts executing the code. The full bootstrap program is thus able to load the entire operating system from a non-fixed location on a disk & to start the operating system

.

**Q7. What is the purpose of command interpreter? Why is it usually separate from the kernel?**

**Ans.** One of the most important system programs for an operating system is the command interpreter, which is the interface between the user & the operating system. When a user gives instructions to the computers by using system calls, the command interpreter interprets these commands & directs the system resources in handling the requests. Hence, the command interpreter provides a user interface to hide the hardware details of the system from the user.

A kernel is different from an interpreter. The size of the operating system is very large & it is not wise to keep the full operating system in the memory all the time because very little space would be left for other application programs, due to the limited size of the memory. Therefore, the operating system is divided into two parts. One consists of the very essential routines, which are required all the time, & other consists of less frequently required routines. The important portion is called the kernel of the operating system. This is the innermost layer of the operating system close to the hardware & controlling the actual hardware.

Some operating systems include the command interpreter in the kernel. Other operating systems, such as MS-DOS & UNIX, treat the command interpreter as a special program that is running when a job is initiated, or when the user first logs on.

**Q8. What is a process? Discuss various states of a process with the help of a process state transition diagram.**

**Ans.** As a process executes it changes its state. This is known as process state. The current activity of the process defines the state of the process. Each process may be in one of the following state.

New: Process is being created.

Running: Instructions are executing

Waiting: P. waiting for some event to occur.

Ready: The process is waiting to be assigned to the processor.

Terminated: The process has finished execution

**Process state transition diagram**

NEW

**ready**

**running**

**waiting**

**terminated**

*admitted*

**interrupt**

**exit**

**Scheduler dispatch**

**I/O or event** *wait*

I/O event completion

### Q9. What is context switching? How does it help the system in processing?

**Ans.** Switching the CPU to another process required saving the state of the old process & loading the saved state for the new process. This task is known as context switch. When a process issues an I/O system call, the operating system takes over this I/O function on behalf of that process, keeps this process away & starts executing the I/O, after storing the context of the original process in its register save area. When the I/O is completed for the original process, that process can be executed again. The entire operation is very fast & therefore, the user thinks that he is the only one using the whole machine. Context switch times are highly dependent on hardware support.

Q10. What is the average turnaround time for the following processes using

a) FCFS b) SJF non-preemptive c) Preemptive SJF

Process Arrival Time Burst Time

P1 0.0 8

P2 0.4 4

P3 1.0 1

**Ans.** A) FCFS:

Averages turnaround time = 8+12+13 = 33

----------- --- = 11

3 3

B) SJF non-preemptive

Average turnaround time = 1+5+13 = 19

------------ --- = 6.33

3 3

C) SJF Preemptive

Average turnaround time = 5+1+0 = 6

------------ ------ = 2

3 3

**Q11. What are the two main categories of CPU scheduling? Explain with examples.**

**Ans.** **CLASSIFICAITON OF CPU SCHEDULING OR SCHEDULING**

**Preemptive:** In this scheduling resources are preempted from one process and assigned to another process. The preemptive scheduling scheme takes place in these cases:

1.When a process switches from running state to ready state. (For e.g. when an interrupt occurs).

2.When a process switches from waiting state to ready state (for e.g. when complete I/O operation.

Preemptive means CPU can leave a process in between even it has not completed.

An example of preemptive scheduling is the Round Robin algorithm.

**Non-preemptive: -** When resources are not forcefully taken from one process and assigned to another process. The non-preemptive scheduling scheme takes place in these cases:

1.When a process switches from running state to waiting state (for e.g. it may require I/O resource or may be some of its child process is executing

2. When a process terminates.

An example of non-preemptive scheduling algorithm is FCFS algorithm

**Q12. In which of the algorithms of process scheduling, starvation may occur. What can be the solution to this starvation?**

**Ans.** A major problem with priority scheduling algorithms is indefinite blocking or starvation. A process that is ready to run but having low priority can be considered blocked, waiting for the CPU. A priority-scheduling algorithm can leave some low-priority processes waiting indefinitely for the CPU. In a heavily loaded computer system, a stream of high-priority processes can prevent a low priority process from ever getting the CPU.

A solution to this problem 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 1(low) to 100 (high), we would increment the priority of a waiting process by 5 every 1 minute. Eventually, even a process with an initial priority of 0 would have the highest priority in the system & would be executed.

**Q13. What are virtual machines?**

**Ans**. It is a concept, which creates an illusion of a real machine. It is created by a virtual machine Operating System that makes a single real machine appear to be several real machines. By using CPU scheduling and virtual memory techniques an Operating System can create the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.

Whenever more than one program runs on the computer concurrently, it is known as multiprogramming. In case of multiprogramming only one CPU is switched back and forth between the processes in such a way that every process thinks that it has its own CPU for processing. It this way one real CPU gives an illusion of several virtual CPU’s. This is possible through process scheduling and virtual memory organization technique.

**Q14. What is Round Robin Scheduling? Explain taking any example. Can it be useful for a single user system? If yes, then explain. If no, then why not?**

**Ans.** **Round Robin Scheduling / Time Slice Scheduling**

1.Here the ready queue is FIFO circular queue.

2. This algorithm is similar to FCFS but the difference is that it is preemptive scheduling.

3. Here every process is given a time slice or time quantum. The time quantum is a small unit of time defined from 10 to 100 milliseconds.

4. When the time is expired for a process, its CPU is preempted and given to next process in ready queue and this process is placed at the end or tail of the queue.

5.In this scheduling a number of processes are allocated to the CPU for more than 1 time quantum in a row. If a process’s time quantum exceeds CPU burst, that process is preempted and put back in the ready queue.

6.The performance of RR algorithm depends heavily on the size of the time quantum.

#### Advantages:

1. Component utilization is good.

2. Turn around time depends on the size of the quantum. The average turnaround increases for a smaller time quantum.

#### Disadvantage: 🡪

Average waiting time is often quit long.

#### Working

The average wait time in the RR scheduling may be often quite long. Consider the following set of processes that arrive at time 0, with the required CPU-time in milliseconds given as under:

Queue: 1st 2nd 3rd

Process: P1 P2 P3

Burst time: 24 3 3

Assuming that the CPU time-slice is 4 milliseconds, the scheduling takes place as explained next. The first process P1 is allocated the CPU. It runs for 4 milliseconds when a context switching takes place and P1 is placed in the end of the queue with required CPU burst-time (24 – 4 =20) milliseconds. The schedule is shown as below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P1  **0**  **4**  **7**  **10**  **14**  **18**  **22**  **26** | P2 | P3 | P1 | P1 | P1 | P1 | P1 |

**30**

Process P2, however, completes its execution before the time-slice expires and then P3 is allocated the CPU.

Turn around time for process P1 = 3 0

Turn around time for process P2 = 7

Turn around time for process P3 = 10

Average turnaround time = (30 + 7 + 10)/3 = 15.6 milliseconds.

Wait time for process P1 = 10 - 4 = 6 milliseconds

Wait time for process P2 = 4 milliseconds

Wait time for process P3 = 7 milliseconds

Average wait time = (6 + 4 + 7)/3 = 5.66 milliseconds.

A single user can have multiple processes & in order to run these process, multiprogramming & time-sharing is implemented. Time-sharing is implemented through Round Robin. Thus this algorithm is useful in an environment in which there are multiple processes

.

**Q15. Explain the difference in the degree to which the following scheduling algorithms discriminate in the favour of short processes.**

a) FCFS b) RR c) Multilevel feedback queues

**Ans. FCFS : -** In the FCFS techniques, the CPU-bound process will get the CPU & hold it. During this time, all other processes finish their I/O & move into the ready queue, waiting for the CPU. While the processes wait in the ready queue, the I/O devices all idle. Finally, the CPU-bound process finishes its CPU burst & moves to an I/O device. All the I/O bound processes, which have very short CPU bursts execute quickly & move back to the I/O queues. However, for a big process, all the other short processes have to wait. There, this algorithm supports the execution of short processes but with low CPU utilization.

**Round robin: -** RR: - In the round-robin algorithm, no process is allocated the CPU for more than 1 time quantum. Thus, the short processes will finish early as compared to the other larger processes that will continue in the same manner after the shorter processes are finished.

**Multilevel feedback queues: -** This algorithm allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a priority queue may be moved to a higher-priority queue. This form of aging prevents starvation. Long processes are automatically queued at the end and are served in FCFS order.

**Q16. Write a short note on inter process communication.**

**Ans.** As we know cooperating processes can communicate in a shared memory environment. The scheme requires that these processes share a common buffer pool & the code for implementing the buffer be explicitly written by the application programmer. Another way to achieve the same effect is for the operating system to provide the means for cooperating processes to communicate with each other through an inter process communication (IPC) facility. IPC provides a mechanism to allow processes to communicate & to synchronize their actions. IPC is best provided by a message system. The function of a message system is to allow processes to communicate with each other without the need to resort to shared variables. An IPC facility provides at least the two operations send (message) & receive (message). Message can be of either fixed or variable size.

Processes that want to communicate must have a way to refer to each other. They can use either direct or indirect communication. In the direct communication, each process that wants to communicate must explicitly name the recipient. The messages are sent to & received from mailboxes. A mailbox can be viewed as an object into which messages can be placed by processes & from which messages can be removed.

**Q17. Explain the I/O Data Transfer Technology.**

**Ans**. Most of the devices are connected or interfaced to the computer system as one or more memory locations. All I/O devices will have a controller with some registers or memory locations, and interfacing address, data and control lines called buses. There are three ways by which data can be sent from I/O device to the memory by the CPU.

* 1. **Programmed I/O:** - In programmed I/O mode, when a input output instruction is encountered in a program, the CPU checks the input output flag. It keeps on checking the flag till the flag is set to 1 when the flag is set to 1 the CPU transfers data from I/O device to the memory.
  2. **Interrupt driven I/O**: - In interrupt driven I/O system, the device that is ready to transmit sends a single to the CPU indicating that it requires data transmission. The CPU then takes action accordingly.
  3. **Polling**: - Polling is a method of I/O handling by CPU wherein the CPU continuously interrogates devices attached to it to determine if a device wishes to transmit. If a device is ready to transmit, it sends back anknowledgement code and the transmission sequence begins.

Direct Memory Access (DMA): - DMA is a method of transferring data from the computer’s RAM to another part of the computer without processing it using the CPU. While most data that is input or output from your computer is processed by the CPU, some data does not require processing, or can be processed by another device. In these situations, DMA can save processing time and is a more efficient way to more data from the computer’s memory to other devices.

**Q18. Describe the general objectives of the memory management system within an O.S.**

**Ans.** The main purpose of a computer system is to execute programs. These programs, together with the data they access, must be in main memory during execution. To improve both the utilization of CPU & the speed of its response to its users, the computer must keep several processes in memory. There are many different memory management schemes. These schemes reflect various approaches to memory management & the effectiveness of the different algorithms depends on the particular situation. Since main memory is usually too small to accommodate all the data and programs permanently, the computer system must provide secondary storage to backup main memory.

In general, the memory management module has the following objectives.

1. To keep track of all memory locations – free or allocated and if allocated, to which process & how much.

2. To decide the memory allocation policy: which is which process should get how much memory, when & where.

3. To use various techniques & algorithms to allocate & locate memory locations. This is achieved with the help of some special hardware.

**Q19. Differentiate between internal and external fragmentation.**

**Ans.** As processes are loaded & removed from memory the free memory space is broken into little pieces. External fragmentation exists when enough total memory space exists to satisfy a request, but it is not contiguous. That is, storage is fragmented into a large number of small holes.

Consider the hole of 18,464 bytes. Suppose that the next process requests 18,462 bytes. If we allocate exactly the requested block, we are left with a hole of 2 bytes. The overhead to keep track of this hole will be larger than the hole itself. Thus, the allocated memory may be slightly larger them the requested memory. The difference between these two numbers is internal fragmentation, which means that memory that is internal to a partition, but is not being used.

Q20. Explain the following allocation algorithms

a) First Fit b) Best Fit c) Worst – Fit

Also, given memory partitions of 100k, 500k, 200k, 300k, & 600k, how would each of the above algorithms place processes of 212k 417k, 112k & 426 K? Which algorithm makes the most efficient use of memory?

**Ans.** **First fit:** The first fit algorithm allocates the first hole that is big enough. Searching can start either at the beginning of the set of holes or where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough

**Best fit:** Best fit will search the smallest partition large enough to hold the process.

**Worst fit:** It allocates the largest hole, so that some other process can use the resulting hole.

1. First process: 212 K

a) First-Fit 500 k will be allocated

b) Best-Fit: 300 k will be allocated

c) Worst Fit 600 k will be allocated

2. Second process: 417 k

a) First-Fit: 500 k will be allocated

b) Best-Fit: 500 k will be allocated

c) Worst Fit 600 k will be allocated

3) Third process: 112 k

a) First Fit: 500 k will be allocated

b) Best Fit: 200 k will be allocated

c) Worst Fit 600 k will be allocated

4) Fourth process: 426 k

a) First Fit: 500 k will be allocated

b) Best Fit: 500 k will be allocated

c) Worst Fit: 600 k will be allocated

Thus the best-fit algorithm makes the most efficient use of memory.

**Q21. Draw the diagram of segmentation memory management scheme and explain its principle.**

**Ans.** Segmentation is a memory management scheme, which supports programmer view of memory. Programmers don’t think of their programs as a linear array of words. Rather they think of their programs as a collection of logically related entities, such as subroutines or procedures, functions, global or local data area stack etc.

Segments are formed at program translation time by grouping together logically related entities. Formation of these segments varies from one compiler to another. A Pascal compiler might create separate segments for code of each procedure, global data, local data or variable, stack for calling procedure and storing in parameter. On the other hand, a FORTRAN compiler might create a separate segment for each common block. Arrays might be formed as separate segment; each segment in a program is numbered and refers to by a segment number rather than a segment name.

Sub routine

Stack

Sqrt

**Symbol**

**table**

**Main program**

**Q22. In a variable partition scheme, the operating system has to keep track of allocated and free space. Suggest a means of achieving this. Describe the effects of new allocations and process terminations in your suggested scheme.**

**Ans.** In a variable partition scheme, partitions are created dynamically as and when the processes arrive. Here the size of the partitions is equal to the size of the process. In this type of partitioning more process can be allocated and more memory can be utilized in very good way. When process p1 arrives and it requires 50 KB so the partition size will be 50kb. Similarly p2 arises which requires 100 kb so 2nd partition is of 100 kb and so on.

When p1 terminates, its 50 KB become free. Lets say there is another hole of 20 KB lying somewhere else. Now if a process p3 comes, which requires 60 kb of memory, then it will create a problem, as there is enough memory in the system to run p3, but the memory is not in contiguous block, so p3 wont be able to run.

**OS**

**P1**

**P2**

**P3**

**P4**

**P5**

**0**

**40**

**90**

**190**

***310***

***510***

***630***

***640***

This problem is called external fragmentation. This problem can be removed by compaction by which all the holes are either shifted upwards or downwards so that continuous memory is available for the new processes

#### BENEFITS:

1. Number of processes is not dependent on the number of partitions.

2. Degree of multiprogramming is not fixed

3. Size of process does not depend on partitions.

4. CPU utilization is more.

#### DRAWBACKS:

1.Internal fragmentation

2. Non-contiguous blocks of memory.

3.Compaction is very time consuming.

4.Complex memory management system as the partition is created at the arrival of the process.

**Q23. Define the following.**

a) Paging b) Threads c) Virtual memory d) Time Slice

**Ans. a) Paging:** Paging is a technique that allows the logical address space of a process to be non-contiguous, thus allowing a process to be allocated physical memory wherever it is available.

**b) Threads:** A process contains a number of resources such as address space, open files, accounting information, etc. In addition to these resources, a process has a thread of control, e.g., program counter, register contents, and stack. The idea of threads is to permit multiple threads of control to execute within one process. This is often called **multithreading** and threads are often called **lightweight processes**. Because the threads in a process share the same namespace, switching between them is much less expensive than switching between separate processes.

**c) Virtual memory:** Virtual memory is a technique that allows the execution of process that may not be completely in memory. The main advantage of this scheme is that programs can be larger than physical memory. Also, it abstracts main memory into an extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory. This technique frees programmers from concern over memory storage limitations.

**d) Time Slice:** - A short period of time during which a user process gets the attention of the CPU is known as time slice.

**Q24. Which of the following programming techniques & structures are good for the demand-page environment & which are not good. Explain your answer:**

a) Stack b) Hashed symbol table

c) Sequential Search d) Binary Search.

**Ans.** Careful selection of data structures & programming structures can increase locality & hence lower the page-fault rate & the number of pages in the working set.

1. A stack has a good locality since access is always made to the top.
2. A hash table, on the other hand is designed to scatter references producing bad locality.

c) Sequential search technique is also not a very good technical as the locality is not so good. The reason is that access is made but very slowly.

d) In Binary search technique, the locality is good as the access is made very fast by dividing the total input into two every time.

**Q25. What is a page fault? When do page faults occur? Describe the actions taken by the O.S. when the page fault occurs.**

**Ans.** When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used any way.

In many systems, when a process is executing with only a few pages in memory, & when an instruction is encountered which refer to any instruction or data in some other page, which is outside the main memory, a page fault occurs. At this point, the operating system must bring the required page into memory before the execution of that instruction can restart. In this way, the number of pages in the physically memory for that process gradually increases with the page faults. After a while, when a sufficient number of required pages build up, the pages are normally found in the memory & then the frequency of page fault reduces.

**Q26. What is page replacement? Explain any one algorithm of page replacement.**

**Ans.** As the number of processes & the number of pages in the main memory for each process increase, at some point in time, all the page frames become occupied. At this time, if a new page is to be brought in, the operating system has to overwrite some existing page in the memory. The page to be chosen is selected by the page replacement policy.

**1) FIFO Algorithm: -** This algorithm is associated with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. We replace the page at the head of the queue. When a page is brought into memory, we insert it at the end of the queue.

Reference String

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

For our example reference string, our three frames are initially empty. The first 3 references (7,0,1) cause page faults, & are brought into these empty frames. The next reference (2) replaces page 7, because page 7 was brought in first. Since 0 is the next reference & 0 is already in memory, we have no fault for this reference. The first reference to 3 results in page 0 being replaced, since it was the first of the three pages in memory (0,1 & 2) to be brought in. This replacement means that the next reference to 0. will fault. Page 1 is replaced by page 0. this process continues as shown in fig.

**2. LRU (Least Recently used) Algorithm:** In this algorithm, when a page must be replaced, it chooses that page that has not been used for the longest period of time. This strategy looks backward in time.

Reference String:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

First 7,0, cause page faults, then 2 causes another page fault and 7 is swapped out. 0 causes no page fault. 3 causes 1 to be swapped out. No page fault for 0. 4 causes 2 to be swapped out and so on.

**Q27. What is thrashing? Why does it happen, How will you avoid it?**

**Ans.** If the number of frames allocated to a process falls below the minimum number required by the computer architecture, we must suspend that process’s execution. We should then page out its remaining pages. Freeing all its allocated frames. This provision introduces a swap-in, swap-out level of CPU scheduling. If the process does not have the minimum required number of frames, it will very quickly page fault. At this point it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away consequently, it very quickly page faults again & again. The process continues to fault, replacing pages for which it will then fault & bring back in right away. This high paging activity is called thrashing. A process is thrashing if it is spending more time swapping pages than executing. To overcome this problem, we choose a global replacement policy in which any page from any process can be replaced.

### Q28. Explain the process of demand paging

**Ans.** When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not used anyway, decreasing the swap time and the amount of physical memory needed. With this scheme, we need some form of hardware support to distinguish between those pages that are in memory and those pages that are on the disk. The valid-invalid bit scheme can be used for this purpose. This time, however, when this bit is set to “valid,” this value indicates that the associated page is both legal and in memory. If the bit is set to “invalid,” this value indicates that the page either is not valid (that is, not in the logical address space of the process), or is valid but is currently on the disk. The page-table entry for a page that is brought into memory is set as usual, but the page-table entry for a page that is brought into memory is set as usual, but the page-table entry for a page that is not currently in memory is simple marked invalid, or contains the address of the page on disk.

**Q29. Why are segmentation & paging sometimes combined into one scheme**?

**Ans.** Segmentation can be combined with paging to provide the efficiency of paging with the protection & sharing capabilities of segmentation. As with simple segmentation, the logical address specifies the segment number & the offset within the segment. However, when paging is added, the segment offset is further divided into a page number & a page offset. The segment table entry contains the address of the segment’s table. The hardware adds the logical address’s page number bits to the page table address to locate the page table entry. The physical address is formed by appending the page offset to the page frame number specified in the page table entry.

**Q30. When can address binding be done? Explain briefly.**

**Ans.** The binding of instructions and data to memory addresses can be done at any of the following stages:

1. **Compile time:** if it is known at compile time where the process will reside in memory, then absolute code can be generated.
2. **Load time:** If it is not known at compile time where the process will reside in memory, then the compiler must generate relocatable code.
3. **Execution time:** If the process can be moved during its execution from one memory Segment to another, then binding must be delayed until run time.

**Q31. Explain LFU page replacemet algorithm.**

**Ans:** The least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count. This algorithm suffers from the situation in which a page is used heavily during the initial phase of the process, but then is never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed. One solution is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average usage count.

**Q32. What are different file access methods? Discuss.**

**Ans.** Files store information, which when required, may be read into the main memory. There are several different ways in which the data stored in a file may be accessed for reading and writing. The operating system is responsible for supporting these file access methods. Access methods are:

#### Sequential Access: - A sequential file is the most primitive of all file structures. It has no directory and no linking pointers. The records are generally organized in lexicographic order on the value of some key. In other words, a particular attribute is chosen whose value will determine the order of the records. Sometimes when the attribute value is constant for a large number of records a second key is chosen to give an order when the first key fails to discriminate.

The implementation of this file structure requires the use of a sorting routine.

#### Direct Access : - The direct access method is based on the disk model of a file , since disks allow random access to any file block. For direct access, the file is viewed as a numbered sequence of blocks or records. Thus, we may read block 14, then block 54, and then write block 17. There are no restrictions on the order of reading or writing for a direct access file.

Databases are often of his type.

**Indexed Access**

This method involves the construction of an index for the file. The index like an index in the back of a book contains pointers to the various blocks. To find a record in the file, we first search the index, and then use the pointer to access the file directly and to find the desired record.

**Q33. What is Swapping? Explain.**

Ans. A process needs to be in memory to be executed. A process, however, can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution. For example, assume a multiprogramming environment with a round robin CPU scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished, and to swap in another process to the memory space that has been freed. In the meantime, the CPU scheduler will allocate a time slice to some other process in memory. When each process finishes its quantum, it will be swapped with another process. The memory manager can swap processes fast enough that some processes will be in memory, ready to execute, when the CPU scheduler wants to reschedule the CPU. The quantum must also be sufficiently large that reasonable amounts of computing are done between swaps.

**Q34. Explain the necessary conditions for a deadlock to occur in a system.**

**Ans.** A deadlock situation arises if the following is conditions holds simultaneously in a system.

**1. Mutual exclusion:** At least one resource, which is held, must be a non-shareable (like printer)

**2. Hold and wait:** A process is holding one resource and waiting for some other resource held by some other process.

**3. No preemption:** A resource cannot be preempted. i.e. resource cannot be forcefully released from a process. When a process completes execution only than the resource is released.

**4. Circular wait:** There must be a cycle of waiting processes for the resources. Let processes {P0, P1, P2\_ \_ P4} are in waiting state in a way such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2…. And P4 is waiting for resources held by P0.

**Q35. List three examples of deadlocks that are not related to a computer system environment.**

**Ans. 1.Traffic system:** -In a traffic system, if vehicles are waiting in opposite directions, then there is a deadlock in the system as no vehicle is able to move in either direction.

**2. Communication system:** -In a communication system for example telephone system, if a line is always held up, the other numbers or telephones will not be able to access that line.

**3.Banking system:** -In a banking system, if the cash resources are held up by some customers, then the other customers will not be able to withdraw money on time & a deadlock will occur.

**Q36. What is a resource allocation graph? How does it help in detecting deadlocks?**

**Ans.** Deadlocks can be described more precisely in terms of a directed graph called a resource allocation graph. This graph consists of a set of vertices V and a set of edges E.

If there is a cycle present in the graph then there is a deadlock existing in the system.

**R1**

**R3**

**R2**

*R0*

**Q37. What is a safe state & what is its use in deadlock avoidance.**

**Ans.** A state is safe if the system can allocate resources to each process in some order & still avoid a deadlock. In other words, a system is in a safe state only if there exists a safe sequence. A sequence of processes <P1,P2------, PN> is a safe sequence for the current allocation state if, for each P1,the resources that P1 can still request can be satisfied by the currently available resources plus the resources held by all the Pj. In this situation, if the resources that process Pi needs is not immediately available, then Pi can wait until all Pj have finished. When they have finished, Pi can obtain all its resources needed, complete its required task, return its allocated resources & terminate. To show this, we consider a system with 12 magnetic tape drives & 3 processes: P0,P1 & P2. Process P0 requires 10 tape drives, process P1 needs 4 & process P2 need 9. Suppose that at time to, process P0 is holding 5 tape drives, process P1 is holding 2, & process P2 is holding 2 tape drives.

Maximum needs Current needs

P0 10 5

P1 4 2

P2 9 2

At this time, the system is in a safe state. The sequence <P1,P0,P2> satisfies the safety condition, since process P1 can immediately be allocated all its tape drives & then return them, then process Po can get all its tape drives & return them & finally process P2 could get all its tape drives & return them.

**Q38. What are the different methods used for deadlock recovery?**

**Ans.** There are two main methods used for deadlock recovery:

**1) Process Termination:** To eliminate deadlocks by aborting process, we use one of the two methods. In both methods, the system reclaims all resources allocated to the terminated processes.

**a) Abort all deadlocked process:** This method will clearly break the deadlock cycle, but at a great expense, since these processes may have computed for a long time, & the results of these partial computations must be discarded & must be recomputed later.

**b) Abort one process** at a time until the deadlock cycle is eliminated. This methods requires a lot of overhead, since after each process is aborted, a deadlock detection algorithm must be invoked to determine whether any processes are sill dead locked.

**2) Resource preemption:** -To eliminate deadlocks using resource preemption we successively preempt some resourced from processes & give these resourced to other processes until the deadlock cycle is broken. If preemption is required to deal with deadlocks, then these issues need to be addressed:

**a) Selecting a victim:** Which resources & which processes are to be preempted. We must determine the order of preemption to minimize cost. Cost factors may include such parameters as the no of resources a deadlock process is holding & the amount of time a deadlocked process has consumed during its execution.

**b) Rollback:** If we preempt a resource from a process, what should be done with that process? We must rollback the process to some safe state and restart if from that state.

**c) Starvation:** We have to ensure that starvation will not occur. For this we must ensure that a process can be picked as a victim only a small finite number of times.

**Q39. Consider a system consisting of four resources of the same type that are shared by 3 processes, each of which needs at most two resources. Show that the system is deadlock free.**

**Ans.** In the problem, the maximum numbers of resources available are 4, the total numbers of processes are 3, and at any time a process requires maximum 2 resources. We can show that the system is deadlock free by proving that it is in a safe state. The first & second processes can be given two resources each and the third process is allowed to wait. When any of the first or second process terminates, the resources of that terminated process can be removed from that process and given to the third process. Thus this situation will create a safe sequence and hence it proves that the system is in a safe state.

Maximum need Available – 4

P1 2

P2 2

P3 2

Process P1 & P2 can be given each 2 resources & process P3 will wait when one of the processes P1 or P2 terminate, its resources are given to process P3.

**Q40. What is the producer – consumer problem?**

**Ans.** Concurrent execution of cooperating processes requires mechanisms that allow processes to communicate with one another and to synchronization their actions. This is shown by a model called the producer consumer problem. A producer process produces information that is consumed by a consumer process. For example, a print program produces characters that are consumed by the printer driver.

To allow producer consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consumer an item that has not yet been produced.

Bounded Buffer: - The bounded buffer producer consumer problem assumes a fixed buffer size. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

Unbounded Buffer: - The unbounded buffer producer consumer problem places no practical limit on the size of buffer. The consumer may have to wait for new items, but the producer can always produce new items.

**Q41. Explain in detail any one classical problem of process synchronization.**

**Ans.** One of the popular classical problem of synchronization is the Readers and writers problem. A data object such as a file or record is to be shared among several concurrent processes. Some of these processes may only want to read the content of the shared object whereas others may want to update (i.e. to read & write) the shared object. If two readers access the shared data object simultaneously, no adverse effects will result. However, if a writer & some other processes (either a reader or a writer) access the shared object simultaneously, there may be problems. To ensure that these difficulties do not arise, we require that the writers have exclusive access to the shared object. This synchronization problem is known as the readers-writes problem. This problem has many variations. One of them is the first readers-writers problem, which requires that no readers will be kept waiting unless a writer has already obtained permission to use the shared object. In other words, no reader should wait for other readers to finish simply because a writer is waiting.

**Q42. What is Critical Section Problem? Explain.**

**Ans.** A section of code or collection of operations in which only one process may be executing at a given time, is called critical section. Consider a system containing n processes {P0, P1, 2, ..., Pn }.Each process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file, and so on. The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. Thus, the execution of critical sections by the processes is mutually exclusive in time. The critical section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the entry section. The critical section my be followed by an exit section. The remaining code is the remainder section.

do

{

Entry Section

Critical Section

Exit Section

Remainder Section

} While (1);

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

1. **Mutual Exclusion**: - If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

2. **Progress**: - If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision on which will enter its critical section next.

3.**Bounded Waiting**: - There exists a bound 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.

**Q43. What are Semaphores? Explain**.

**Ans.** The solutions to the critical section problem are not easy to generalize to more complex problems. To overcome this difficulty, we can use a synchronization tool called a semaphore. A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: Wait and signal.

The classical definition of wait in pseudo code is:

Wait (S)

{

While (S<0)

; //no loop

S--;

}

The classical definition of Signal in pseudo code is:

Signal (S)

{

S++;

}

Modification to the integer value of the semaphore in the wait and signal operations must be executed indivisibly. That is, when one process modifies the semaphores value, no other process can simultaneously modify that same semaphore value. In addition, in case of the wait (S), the testing of the integer value of S(S<0), and its possible modification (S--), must also be executed without interruption.

**Q44. Define the levels of Security.**

**Ans**. Security levels of a system are defined on the basis of the resources that are being secured. The levels of Security are as fellow: -

a) **Physical level**: - This level is achieved by securing the physical hardware components. The computers that store sensitive information are secured to prevent their unauthorized physical access. You can prevent your computer from unauthorized physical access by using special devices such as, locking your computer cases. You can achieve physical security in your organization by employing a guard who restricts the entry of unauthorized users within the organization.

b) **Human level**: -This level is achieved by using passwords to gain access to resources. Passwords should not be stored in a file as any user can have access to that file. Human security is violated when that user reveals the information to intruders can be authorized and unauthorized end users who can access the resources of other authorized users without their permission.

c) **Network level**: - It is achieved by protecting data and resources to be transferred over network against intruders In a network, data is transferred through leased private lines, dial –up lines and the internet. Intruders can easily attack the data while it is being transmitted over a network. User authentication and anti virus software are being used for protecting the data within the network. Am anti virus software is used to guard the data against viruses. Virus is a computer program that affects your computer and hence, affects other programs executing in the computer and thereby damaging the whole system.

d) **Operating system Level**: - This level provides dial-up and login gates for maintaining security while communicating in a network. Gates are referred to as network passwords that enable different users to gain entry into the network by verifying passwords.

**10 Marks Questions**

**Q1. Discuss the evolution of operating system. How many types of operating systems are there?**

**Ans.** **1) Simple batch system:**

It has a single processor, runs a single programming at a time and interacts with single user at a time. To speed up processing, jobs with similar needs were batched together and were run through the computer as a group. Thus the programmers would leave their programs into batches with similar requirements and, as the computer became available, would run each batch. The output from each job would be sent back to the appropriate programmers.

**2) Batch processing system:**.

In batch processing programs, data and system commands are submitted to computer in the form of a job. It takes several jobs in a queue and automatically keeps executing one job to next job in the queue. There may be little or no interaction between the user and executing program. The batch monitor does all the functions of the batch processing system. Batch monitor are permanently stored on the lower part of the memory and the rest part of memory is left for the execution of current job.

**3) Multiprogramming operating system:**

When more than one program is executing in the system then it is called multiprogramming. The operating system that supports more than one program to execute simultaneously is called multiprogramming operating system.

Multitasking or multiprogramming operating system supports concurrent execution of two or more programs (at the same time). Multiprogramming increases CPU utilization by organizing jobs in several ways such that the CPU always has one job to execute. The idea behind this is -- As the operating system keeps several jobs in the memory at a time, some jobs are kept in the job queue. The operating system picks and executes one of the jobs. This job may have to wait for some task, then operating system switches to and executes another job. When that job needs to wait, the CPU switches to another job and so on. When first job finishes waiting then it gets CPU back. Hence there is always some job to execute, so CPU will never sit idle.

**4) Time sharing system:**

A time-shared operating system allows many users to share the computer simultaneously. As each process in the time-shared system is very short so only little CPU time is needed for each user. As the system switches rapidly from one user to another the user thinks that he is the only user working in the system whereas actually many users share the computer.

Time-sharing system sets the time for every process. Therefore it requires round robin scheduling. In this system, every job is allocated a time slice. If time of one job is completed but job is not finished then this job is placed at the end of queue of waiting program.

**5) Parallel system:**

Most systems today are single processor systems, that means they have only one main CPU.A system containing more than one processor is multiprocessor system. Such systems have more than one processor in close communication, sharing the computer bus, the clock and sometimes memory and peripheral devices. These systems are referred to as tightly coupled system.

**6) Distributed system:**

A recent trend in computer system is to distribute computation among several processors. In this scheme the processors do not share memory or clock, instead every processor has its own memory and clock chip.

The processors can communicate with one another through various communication lines. These systems are usually referred to as loosely coupled systems or distributed systems. The processors in this scheme are referred to by a number of different names like sites, nodes, computers and so on.

**7) Real time system:**

Real time operating system is used in an environment where a large number of events, must be accepted and processed in a short time. It has well defined fixed time constraints. Processing must be done within he defined constraints, or the system will fail. Such application includes telephone switching equipment, flight control and real time simulation.

**Q2. What are the different components of an O.S. Explain each of them briefly.**

**Ans.** The different components of operating system are :

1. **Process management:** The various activities of this component are:

creation and deletion of both system and user processes.

Suspension and resumption of processes

The facility of methods for process synchronization

The facility of methods for process communication

The facility of methods for deadlock handling

2) **Memory management:** The various activities of this component are:

Keeps track of which parts of memory are currently being used and by whom

Decides which processes are to be loaded into memory when memory space

becomes available.

Allocates and deallocates memory space as required

3) **File management:** The various activities of this component are :

Helps in creation and deletion of files

Helps in creation and deletion of directories

Backup of files on storage media

4) **I/O system management:** This component includes:

Memory management component including buffering, caching and spooling

Device driver interface

Drivers for specific hardware devices.

5) **Secondary storage management:** The various activities of this component are :

free space management

storage management

disk scheduling

6) **Networking:** A distributed system is a collection of processors that do not share memory or peripheral devices. Each processor has its own local memory and the processors communicate with each other in various communication lines such as telephone lines.

Access to a shared resource allows computation speed up, increased data availability and increased reliability.

7) **Protection systems:** If a computer system has multiple users and allows concurrent execution of multiple processes then the various processes must be protected from one another. For this purpose methods are provided to ensure that files , memory segments, CPU and other resources can be operated only by those process that have taken proper authorization from the operating system.

8) **Command interpreter system:** It is one of the most important system programs for an operating system. It is an interface between the user and the operating system. When a new job is started or when a user logs on to a time-shared system a program that reads and interprets control statement is executed automatically. This program is known as command line interpreter or shell.

**Q3. What are system calls? Explain the different types of system calls.**

**Ans.** System calls provide the interface between a process and the operating system. These calls are generally available as assembly-language instructions, and are usually listed in the manuals used by assembly-language programmers.

Some systems may allow system calls to be made directly from a higher-level language program, in which case the calls normally resemble predefined function or subroutine calls.

Several languages — such as C, Bliss, BCPL, PL/360, and PERL — have been defined to replace assembly language for systems programming. These languages allow system calls to be made directly. Some Pascal systems also provide an ability to make system calls directly from a Pascal program to the operating system. C and PERL implementations include direct system call access. Most FORTRAN systems provide similar capabilities, often by a set of library routines.

As an example of how system calls are used, consider writing a simple program to read data from one file and to copy them to another file. The first input that the program will need is the names of the two files: the input file and the output file. One approach is for the program to ask the user for the names of the two files. In an interactive system, this approach will require a sequence of system calls, first to write a prompting message on the screen, and then to read from the keyboard the characters that defines the two files.

Once the two file names are obtained, the program must open the input and create the output file. Each of these operations requires another system call. There are also possible error conditions for each operation. When the program tries to open the input file, it may find that there is no file of that name or the file is protected against access. In these cases, the program should print error message on the console (another sequence of system calls) and then terminate abnormally (another system call). If the input file exists, then we must create a new output file. We may find that there is already an output file with same name. This situation may cause the program to abort (a system call) or we may delete the existing file (another system call) and create a new file (another system call). System calls can be roughly grouped into five major categories:

Process control, file manipulation, device manipulation, information maintenance, and communications that may be provided by an operating system. Some of the system services are:

**Process control**

End, abort

Load, execute

Create process and terminate process

Get process attributes, set process attributes

Wait for time

Wait event, signal event

Allocate and free memory

**File manipulation**

Create file, delete file

Open, close

Read, write, reposition

Get file attributes, set file attributes

**Device manipulation**

Request device, release device,

Read, writes, reposition,

Get device attributes, set device attributes

Logically attach or detach devices

**Information maintenance**

Get time or date, set time or date

Get system data, set system data

Get process, file, or device attributes

Set process, file, or device attributes

**Communications**

Create, delete communication connection

Send, and receive messages

Transfer status information

**Q4. Whatis the function of PCB? Explain with the help of a diagram.**

**Ans.** To implement the process model, the Operating System maintains a table called the process table with one entry per table.

The Operating System groups all the information about a particular process into a data structure called the PROCESS CONTROL BLOCK. It simply serves as the storage of information about the process. When a process is created the Operating System creates its corresponding PCB and when it terminates Operating System releases its PCB.

Whenever a process is switched back and other process may come for execution then the PCB of previous process will be same and new PCB for new process will be created. When this new process is finished then it takes PCB of previous process in memory and executes it. The PCB of finished process will be released and terminated. A PCB is known as:

**PROCESS C0NTROL BLOCK: -** It contains many piece of information associated with the specific process including these:

**Process State:** It tells the current state of process, which may be new , ready , running , waiting, terminated.

**Program Counter: -** A counter indicates the address of next instruction to be executed for this process.

**CPU Registers:**  Registers used by and required by the process. The registers may vary in number and type.

**Process Number: -** Each process is identified by its process number called process I/O.

**CPU Scheduling Information: -** This information includes process priority pointers to scheduling queues and other scheduling parameters.

**Memory Management Information: -** It contains value of base and limit registers the page table or the segment table.

**Accounting Information: -** It contains amount of CPU and real time.

**I/O Status Information: -** Includes the list of I/O devices allocated to this process list of open files and so on.

|  |  |
| --- | --- |
| Pointer | Process state |
| Process number | |
| Program counter | |
| Registers | |
| Memory limits | |
| List of open files | |
| ….. | |

**Q5. Compare the different scheduling algorithms & advantages of them over each other.**

**Ans.** **First come first serve scheduling:**

* It is simplest CPU scheduling algorithm.
* Ready queue is implemented as FIFO (first in first out) queue.
* The code of FCFS scheduling is simple to write and understand
* Average waiting time in FCFS scheduling algorithm is quite long
* This scheduling algorithm is non preemptive i.e. once the CPU is allocated to the process, then the process keeps the CPU until it terminates or requesting for I/O device.

**Advantage:** It is easy to design and implement.

**Disadvantages:**

* Low CPU and device utilization. Average waiting time is quite long.
* It cannot be implemented in time-sharing system.
* It results poor performance as component utilization and system through put rate may be quite low.

As an example, consider the following set of processes. The length of CPU burst time is given in milliseconds.

Working

Process: P1 P2 P3 P4

Burst time: 6 8 7 3

Turnaround time is the time taken by a process between its getting into the ready queue and finishing execution. Wait time is the time a process must wait before it starts executing. Following Gantt chart depicts the schedule under this scheme:

|  |  |  |  |
| --- | --- | --- | --- |
| P1 | P2 | P3 | P4 |

0

**6**

**14**

**21**

**24**

Turn around time for process P1 = 6

Turn around time for process P2 = 6 + 8 = 14

Turn around time for process P3 = 14 + 7 = 21

Turn around time for process P4 = 21 + 3 = 24

Average turnaround time = (6 + 14 + 21 + 24)/4 = 16.25 milliseconds.

Wait time for process P1 = 0

Wait time for process P2 = 0 + 6 = 6

Wait time for process P3 = 6 + 8 = 14

Wait time for process P4 = 14 + 7 = 21

Average wait time = (0 + 6 + 14 + 21)/4 = 10.25 milliseconds

##### SHORTEST JOB FIRST

Main points:

1) The jobs taking lesser processor time will be executed first. So the jobs are placed in ready queue are arranged in the increasing order of their CPU bust time with the first job having the minimum burst time.

2) It may be preemptive or non-preemptive depending upon the CPU burst time of the processor

3) The average waiting time of P.S.J.F.(Preemptive Shortest Job First) or N.P.S.J.F. (Non Preemptive Shortest Job First)is less compare to F.C.F.S.

4) The average waiting time of P.S.J.F is less than NP –SJF

Turn around time of preemptive SJF<Turn around time of non-preemptive SJF

**Advantages:**

1. It is optional as it gives the minimum average time for a process.

2.Component utilization is comparatively high.

3. SJF scheduling is used frequently in long-term scheduler.

**Disadvantages:**

1. The bust time of every process must be known as it requests.

2. It cannot be implemented at short-term scheduler level.

Working

As an example, consider the following set of processes, with the length of the CPU burst time given in milliseconds:

Process: P1 P2 P3 P4

Burst time: 6 8 7 3

Using SJF, the schedule is depicted by the following Gantt chart:

|  |  |  |  |
| --- | --- | --- | --- |
| P4 | P1 | P3 | P2 |

0

**3**

**9**

**166** *666666*

**24**

Turn around time for process P1 = 3 + 6 = 9

Turn around time for process P2 = 16 + 8 = 24

Turn around time for process P3 = 9 + 7 = 16

Turn around time for process P4 = 3

Average turnaround time = (3 + 9 + 16 + 24)/4 = 13 milliseconds.

Wait time for process P1 = 0 + 3 = 3

Wait time for process P2 = 9 + 7 = 16

Wait time for process P3 = 3 + 6 = 9

Wait time for process P4 = 0

Average wait time = (0 + 3 + 9 + 16)/4 = 7 milliseconds.

As is obvious, there is an improvement both in average turnaround time and average waiting time as compared to FCFS.

#### PRIORITY SCHEDULING:

Here ready queue is implemented as a priority queue.

The CPU is allocated to a process with highest priority. Priority may be defined internally or externally.

If two processes have same priority then they are scheduled as FCFS order.

It is not necessary that low number represent low priority. Some system takes low number to represents low priority and some takes low number to represents high priority. Here we assume the low number as high priority

SJF is a special case of priority scheduling algorithm.

It may be preemptive or non preemptive when a process arrives in the ready queue then its priority is compared to priority of currently running process. If priority of new process is high then the current process will preempt the CPU to new process. If current process has high priority then non-preemptive scheduling will put the newly process in the head of ready queue.

**Disadvantage:** Indefinite blocking or starvation. The algorithm can leave some low priority processes waiting indefinitely for CPU.

**Solution:** A solution of this problem is aging. Aging is a technique of gradually increasing the priority of the processes that wait in a system for a long time.

#### ROUND ROBIN SCHEDULING / TIME SLICE SCHEDULING

1.Here the ready queue is FIFO circular queue.

2. This algorithm is similar to FCFS but the difference is that it is preemptive scheduling.

3. Here every process is given a time slice or time quantum. The time quantum is a small unit of time defined from 10 to 100 milliseconds.

4. When a time slice for a process is over, its CPU is preempted and given to next process in ready queue and this process is placed at the end or tail of the queue.

5.In this scheduling, a number of processes are allocated to the CPU for more than 1 time quantum in a row. If a process requires more time than CPU burst time, it is preempted and put back in ready queue.

6.The performance of RR algorithm depends heavily on the size of the time quantum.

#### Advantages:

1. Resource utilization is good.

2. Turn around time depends on the size of the quantum. The average turnaround increases for a smaller time quantum.

#### Disadvantage:

Average waiting time is often quit long.

#### Working:

The average wait time in the RR scheduling may be often quite long. Consider the following set of processes that arrive at time 0, with the required CPU-time in milliseconds given as under:

Queue: 1st 2nd 3rd

Process: P1 P2 P3

Burst time: 24 3 3

Assuming that the CPU time-slice is 4 milliseconds, the scheduling takes place as explained next. The first process P1 is allocated the CPU. It runs for 4 milliseconds when a context switching takes place and P1 is placed in the end of the queue with required CPU burst-time (24 – 4 =20) milliseconds. The schedule is shown as below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P1 | P1 | P1 | P1 | P1 |

**0**

**4**

**7**

**10**

**14**

**18**

**22**

**26**

**30**

Process P2, however, completes its execution before the time-slice expires and then P3 is allocated the CPU.

Turn around time for process P1 = 3 0

Turn around time for process P2 = 7

Turn around time for process P3 = 10

Average turnaround time = (30 + 7 + 10)/3 = 15.6 milliseconds.

Wait time for process P1 = 10 - 4 = 6 milliseconds

Wait time for process P2 = 4 milliseconds

Wait time for process P3 = 7 milliseconds

Average wait time = (6 + 4 + 7)/3 = 5.66 milliseconds.

Q6. Discuss the need of process scheduling & types of schedulers

**Ans.** Scheduling is the process of bringing the processes to the ready queue and then sending them to the processor. The processes are residing in the ready queue and are given to the CPU one by one according to the scheduling algorithm used.

**Scheduler**

A scheduler is a module of operating system, which gives the time to each process. It refers to a set of policies and mechanisms supported by Operating System that controls the order in which the work has to be done. All computer resources and processes are scheduled before use and executes.

**SCHEDULING TECHNIQUES / TYPES OF SCHEDULERS / LEVELS OF SCHEDULING**

Basically there are 3 types of scheduler which may co-exists in the complex Operating System:

1.Long-term schedule 2.Medium-term scheduler 3.Short-term scheduler

**Long-Term Scheduler (LTS)**

The job of LTS is to take a process from job queue and place it in the ready queue. It is also known as job scheduler. It executes much less frequently as short time scheduler because it has to decide which process to place in the ready queue. As it controls the degree of multiprogramming. It makes a careful selection of process, which may be a mix of I/O bound process, and CPU bound process. If all process is I/O bound then all come in suspended state.

**Short-Term Scheduler /CPU Scheduler (STS)**

The job of STS is to take a process from ready queue and allocate CPU to them. Ready queue is not necessary implemented as FIFO queue. It depends on various scheduling algorithm. The ready queue may be implemented as a FIFO queue, a priority queue, a tree or simply an unordered linked list. The short-term scheduler is also known as CPU scheduler. The short-term scheduler selects a new process for the CPU quite frequently.

#### Medium Term Scheduler (MTS)

When a process has to wait for some event to occur then Operating System switches the CPU to other process. The previous or waiting process is swapped out and the new or ready process is swapped in to the ready queue.

Medium term scheduler does the process of swapping-in and swapping-out. Some processes are suspended and some processes are brought into memory. The job of swapping out and in is done by M.T.S. This is done so that the memory may be available for new processes.

**Q7. What is thread? Explain user thread and kernel thread**..

**Ans**: - A thread is created explicitly when modern process is created. The process now includes only the process control block data structure and the resources. A thread is different from the process. A thread requires a program counter(PC), a stack and a set of CPU registers for execution. A process can have more than one thread of execution. All threads of a process share the address space of the process. Therefore, no new address space needed to be created while creating a thread unlike process creation. Hence a thread is called a lightweight process (LWP). However, each thread must have its own CPU registers, program counter and stack. So, as in the cse of processes in an uni-processor system, if only one set of CPU registers and PC are available, these resources must be shared in a time multiplexed way. Each thread will have a data structure called thread control lock or thread descriptor. A thread descriptor includes: -

1. State: - A thread’s current state like ready, blocked and running
2. Time of execution, start time, etc.
3. Process: - A reference to the PCB of associated process
4. List of related threads: - A reference to the list of parent/child
5. Stack: - The location of the base thread’s stack in the main memory
6. Other resources: - A reference to the thread specific resources

**User Thread:** - These are threads created and managed by the user. These are created, managed and scheduled by the user using thread library functions in the user space. User thread creation and termination is faster than kernel threads as user thread creation/termination does not involve kernel mode change. However, the blocking of a single user thread may lead to the blocking of the entire process.

**Kernel Thread**: - These are threads created and managed by the kernel itself. A user process will be executed as one or more kernel threads. Threads are scheduled directly by kernel. No user level library is needed for the creation and management of threads. One major advantage is that when one thread get blocked due to a system call, the kernel may choose to schedule the other threads in the process making applications to run faster with improved response time. However, kernel thread creation/termination are costly as it involves the time-consuming kernel mode change.

## 

Q8. Explain process scheduling. What are scheduling queues, schedulers and context switch?

Ans The objective of multiprogramming is to have some process running at all times, so to maximize CPU utilization. Time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running. A uniprocessor system can have only one running process. If more processes exist, the rest must wait until the CPU is free anf can be rescheduled.

**Scheduling Queues**: - As processes enter the system, they are put into a job queue. This queue consists of all processes in the system. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue. The list of processes waiting for a particular I/O device called a device queue.

A new process is put in the ready queue. It waits in the ready queue until it is selected for execution. Once the process is assigned to the CPU and is executing, one of several events could occur: -

* The process could issue an I/O request, and then be placed in an I/O queue.
* The process could create a new sub process and wait for its termination.

**Scheduler**

A scheduler is a module of operating system, which gives the time to each process. It refers to a set of policies and mechanisms supported by Operating System that controls the order in which the work has to be done. All computer resources and processes are scheduled before use and executes.

**SCHEDULING TECHNIQUES / TYPES OF SCHEDULERS / LEVELS OF SCHEDULING**

Basically there are 3 types of scheduler which may co-exists in the complex Operating System:

1.Long-term schedule 2.Medium-term scheduler 3.Short-term scheduler

**Long-Term Scheduler (LTS)**

The job of LTS is to take a process from job queue and place it in the ready queue. It is also known as job scheduler. It executes much less frequently as short time scheduler because it has to decide which process to place in the ready queue. As it controls the degree of multiprogramming. It makes a careful selection of process, which may be a mix of I/O bound process, and CPU bound process. If all process is I/O bound then all come in suspended state.

**Short-Term Scheduler /CPU Scheduler (STS)**

The job of STS is to take a process from ready queue and allocate CPU to them. Ready queue is not necessary implemented as FIFO queue. It depends on various scheduling algorithm. The ready queue may be implemented as a FIFO queue, a priority queue, a tree or simply an unordered linked list. The short-term scheduler is also known as CPU scheduler. The short-term scheduler selects a new process for the CPU quite frequently.

#### Medium Term Scheduler (MTS)

When a process has to wait for some event to occur then Operating System switches the CPU to other process. The previous or waiting process is swapped out and the new or ready process is swapped in to the ready queue.

Medium term scheduler does the process of swapping-in and swapping-out. Some processes are suspended and some processes are brought into memory. The job of swapping out and in is done by M.T.S. This is done so that the memory may be available for new processes.

**Context Switch**: - Switching the CPU to another process required saving the state of the old process & loading the saved state for the new process. This task is known as context switch. When a process issues an I/O system call, the operating system takes over this I/O function on behalf of that process, keeps this process away & starts executing the I/O, after storing the context of the original process in its register save area. When the I/O is completed for the original process, that process can be executed again. The entire operation is very fast & therefore, the user thinks that he is the only one using the whole machine. Context switch times are highly dependent on hardware support.

**Q9. Explain the following briefly.**

a) File attributes b) File operations c) File types

**Ans.** **File Attributes**

File attributes are properties of a file. The operating system treats a file according to its attributes. Following are a few common attributes of a file:

H: for hidden A: for archive D: for directory

X: for executable R: for read only. These attributes can be used in combinations also.

**File Operations**

Major file operations are as follows:

1.Read operation: 2. Write operation: 3. Execute: 4. Copying file:

5. Renaming file 6. Moving file 7. Deleting file 8. creating file 9. Merging files 10. Sorting file 11. Appending file 12. comparing file

The files under UNIX can be categorized as follows

1. Ordinary files. 2. Directory files. 3. Special files. 4. FIFO files.

**a. Ordinary Files**

Ordinary files are the one, with which we all are familiar. They may contain executable programs, text or databases. You can add, modify or delete them or remove the file entirely.

**b. Directory Files**

Directory files are files that contain other files. They contain list of file names and other information related to these files. Some of the commands, which manipulate these directory files, differ from those for ordinary files.

**c. Special Files**

Special files are also referred to as device files. These files represent physical devices such as terminals, disks, printers and tape-drives etc. These files are read from or written into just like ordinary files, except that operation on these files activates some physical devices. These files can be of two types i)Character device files and ii)block device files.

In character device files data is handled character by character, as in case of terminals and printers. In block device files, data is handled in large chunks of blocks, as in the case of disks and tapes.

**d. FIFO Files**

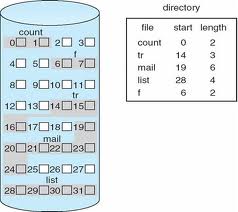
FIFO (first-in-first-out) is a file that allows unrelated processes to communicate with each other. They are generally used in applications where the communication path is in only one direction, and several processes need to communicate with a single process. For an example of FIFO file, consider the example of pipe in UNIX. This allows transfer of data between processes in a first-in-first-out manner. A pipe takes the output of the first process as the input to the next process, and so on.

**Q10.** **What are different Allocation Methods?**

**Ans.** There are many files will be stored on the same disk. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly. Three major methods of allocating disk space are in wide use: - Contiguous, Linked, and Indexed.

**1.CONTIGUOUS ALLOCATION**: -The contiguous allocation method requires each file occupy a set of contiguous blocks on the disk. Disk addresses define a liner ordering on the disk. Contiguous allocation of a file is defined by the disk

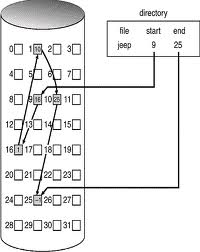
address and length of the first block. Due to contiguous allocation the accessing of file is made easy. Contiguous allocation has some problems; one difficulty is finding space for a new file.



**LINKED ALLOCATION:** -

Linked allocation solves all problems of contiguous allocation. With linked allocation, each file is a linked list of disk blocks, the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. For E.g. a file of five blocks might start at block 9, continue at block 16,then block 1, block 10 and finally block 25.Each block contains a pointer to the next block. These pointers are not made pointer to the user. There is no external

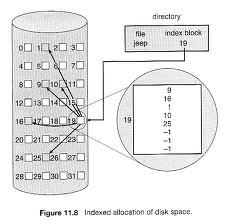
Fragmentation with linked allocation and any free block on the free space list can be used of a file does not need to be declared when that file is created. The major problem is that it can be used effectively only for sequential access files. To find the ith block of a file, we must start at the beginning of that file, and follow the pointers until we get to the ith block. Another disadvantage to linked allocation is the space required for the pointers. If a pointer requires 4 bytes out of 512-byte block, then 0.78 percent of the disk is being used for pointers, rather than for information.



INDEXED ALLOCATION: -

Index allocation is the advancement over linked allocation. Linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and need to be retrieved in order. Indexed allocation solves this problem by bringing all the pointers together into one location. Each file has its own index block, which is an array of disk block addresses. The directory contains the address of the index block. To read the ith block, we use the pointer in the ith index block entry to find and read the desired block.

Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk may satisfy a request for more space. Indexed allocation does suffer from wasted space. The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.



**Q11. Explain the method of paging with the help of a diagram.**

**Ans.** **BASIC METHOD OF PAGING**

The physical memory is broken into fixed size blocks known as frames and the logical memory is broken in to the blocks of same size called pages. When a program runs it pages are loaded in to any of the frame. The size of the page and the frame are same. CPU generates the virtual or logical address of the program. The virtual or logical address has two parts.

#### Page number

#### Off set

**Page number:** The page number is used to index into a page table

**Off set:** Itis combined with the page number to generate the address

E.g.If page 5 contains total 15 lines and stored in 54th position in of this page then value of off set will be 5+54=59.

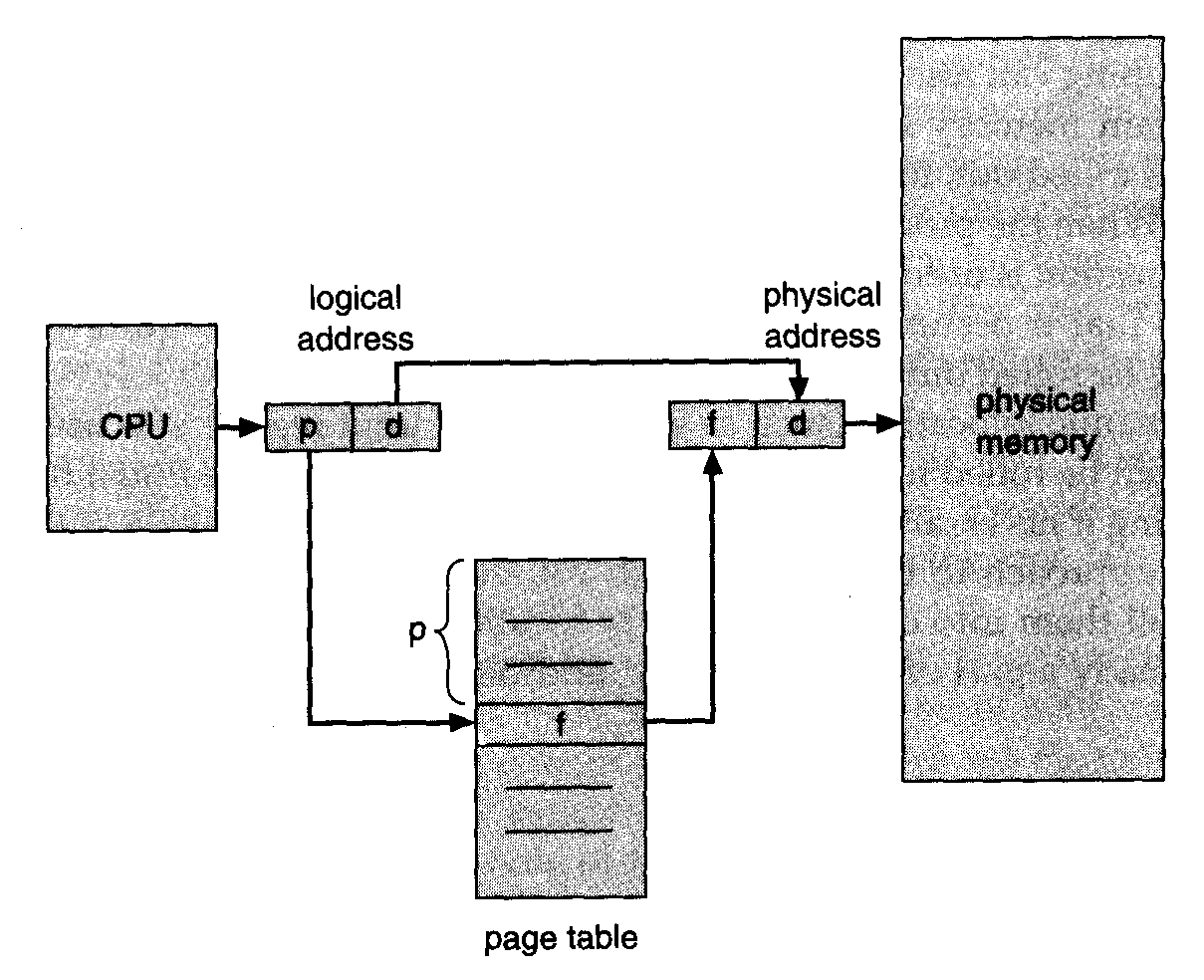
*5 59*

# And virtual address space is

Page size is defined by the hardware.

**PMT (Page map Table):** The information that which page is stored in which frame is known from PMT.

**Address Mapping in a Paging System:** Logical or virtual memory is divided in to pages and virtual address has two parts page number and offset. Physical memory is divided into frames. PMT is a page table that stores the information that which page is placed in which frame. The CPU generates virtual addresses. CPU refers the PMT that tells the position of page. Then CPU accesses the physical address and invokes the information needed.



**Q12. Explain the various directory structures used in operating system for storing files. Give merits & demerits of all directory structures.**

**Ans.**

**1) Single-level Directory: -** It is the simplest directory structure. All files are contained in the same directory, which is easy to support & understand. However, problems arise when the number of files increases or when these is more than one user. Since all files are in the same directory, they must have unique names. Even with a single user, as the number of files increases, it becomes difficult to remember the name of all the files.

**2) Two-level directory:** -The standard solution to the single-level directory is to create a separate directory for each user.

In the two-level directory structure, each user has his own user file directory. Each UFD has a similar structure, but lists only the files of a single user. When a user logs in, the system’s master file directory (MFD) is searched. The master file directory is indexed by user name & each entry points to the UFD for that user.

This structure isolates one user from another. This is an advantage when the users are independent but is a disadvantage when the users want to cooperate on to some task & access each other’s file. Some systems do not allow local user files to be accessed by other users.

**3) Tree-Structured directories: -** This structure allows users to create their own subdirectories and to organize their files accordingly for example; MS-DOS is structured as a tree. The tree has a root directory. Every file in the system has a unique path name. A path name is the path from the root, through all the sub-directories.

A problem with this structure is how to handle the deletion of a directory. If a directory is empty, its entry in its containing directory can simply be deleted. However, if the directory is not empty, then MS-DOS will not delete it.

**4) Acyclic-Graph Directories: -** This structure allows directories to have shared sub-directories & the files. The same file or subdirectory may be in two different directories. One problem with this structure is that a file may now have multiple path names. Another problem involves deletion. Another problem with this structure is to ensure that there are no cycles. Similar problem is inconstancy.

**Q13. What is page replacement? Explain page replacement algorithms.**

Ans: - As the number of processes & the number of pages in the main memory for each process increase, at some point in time, all the page frames become occupied. At this time, if a new page is to be brought in, the operating system has to overwrite some existing page in the memory. The page to be chosen is selected by the page replacement policy.

**1) FIFO Algorithm: -** This algorithm is associated with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. We replace the page at the head of the queue. When a page is brought into memory, we insert it at the end of the queue.

Reference String

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

For our example reference string, our three frames are initially empty. The first 3 references (7,0,1) cause page faults, & are brought into these empty frames. The next reference (2) replaces page 7, because page 7 was brought in first. Since 0 is the next reference & 0 is already in memory, we have no fault for this reference. The first reference to 3 results in page 0 being replaced, since it was the first of the three pages in memory (0,1 & 2) to be brought in. This replacement means that the next reference to 0. will fault. Page 1 is replaced by page 0. this process continues as shown in fig.

**2. LRU (Least Recently used) Algorithm:**

In this algorithm, when a page must be replaced, it chooses that page that has not been used for the longest period of time. This strategy looks backward in time.

Reference String:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

First 7,0, cause page faults, then 2 causes another page fault and 7 is swapped out. 0 causes no page fault. 3 causes 1 to be swapped out. No page fault for 0. 4 causes 2 to be swapped out and so on.

**3. Optimal Page Replacement**

An Optimal page replacement algorithm has the lowest page fault rate of all algorithms, and will never suffer from Belady’s anomaly. Such an algorithm does exist, and has been called OPT or MIN. It is simply replace the page that will not be used for the longest period of time. Use of this page replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.

**4. NRU (Not recently used)**

This algorithm chooses the page that has not been used recently as the victim page for replacement. A page that has been used recently is kept there in the hope that it will be used again in the immediate future. Implementation of NRU requires the use of a referenced bit and a modified bit for each page. A referenced bit is set whenever a page is accessed for execution. This is normally done by the hardware even though software can also do it. The modified bit of a page is set when a page is modified. As in case of the referenced bit, the hardware does the setting of a modified bit. A timer may be used to trigger a hardware circuit to clear the referenced bit at regular prefixed intervals or it can be done through software interrupt handlers. The referenced bits of those pages that were accessed during current period of the timer are set. This period or interval may be chosen appropriate for capturing the locality of the reference of the process.

**5. Clock Algorithm**

This is also a modified version of first in first out, but its overhead is lower than that of second chance. A clock algorithm does not have to move a referenced page from the front to the tail of the queue; instead, the same effect is brought about by using a circular queue and a pointer pointing to the oldest page or the next page to be examined for replacement. When a page is to be replaced, the page at the pointer location is examined for referenced bit, if is not referenced it is chosen as the victim page for replacement. If the pafe is already referenced, its referenced bit is cleared and the pointer is incremented to point to the next page, and that page is examined for replacement, and the process repeats until a page is found that is not referenced for replacement.

**6. LRU (Least Recently Used)**

The least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count. This algorithm suffers from the situation in which a page is used heavily during the initial phase of the process, but then is never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed. One solution is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average usage count.

**7.NFU (Not Frequently Used)**

This algorithm keeps track of the usage of pages it has since loaded for execution. NFU uses a counter with every page and a timer loaded with a time interval. The counter is initialized to zero when the page is loaded. The counter values of all the pages that are referenced during each time out interval of the timer are incremented by one. The counter thus keeps track of the frequency of page usage. The algorithm chooses the page with the lowest counter value for replacement.

**Q14. What is memory allocation? Explain First Fit, Best Fit and Worst Fit.**

**Ans: -** One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. Thus, the number of partitions binds the degree of multiprogramming. In this multiple partitions method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. As processes enter the system, they are put into an input queue. When a process is allocated space, it is loaded into memory and it can then compete for the CPU. When a process terminates, it releases its memory, which the operating system may then fill with another process from the input queue.

At any given time, we have a list of available block sizes and the input queue. The operating system can order the input queue according to a scheduling algorithm. Memory is allocated to process until, finally, the memory requirements of the next process cannot be satisfied; no available block of memory is large enough to hole that process. The operating system can then wait until a large enough block is available or it can skip down the input queue to see whether the smaller memory requirements of some other process can be met.

The first fit, best fit and worst fit strategies are the most common ones used to select a free hole from the set of available holes.

* Fist Fit: - Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or where the previous first fit search ended. We can stop searching as soon as we find a free hole that is large enough.
* Best Fit: - Allocate the simplest hole that is big enough. We must search the entire list, unless the list is kept ordered by size. This strategy produces the smallest leftover hole.
* Worst Fit: - Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best fit approach.

Q15. What is thrashing?

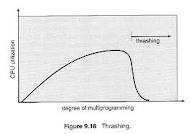
Ans: - If the number of frames allocated to a low priority falls below the minimum number required by the computer architecture, we must suspend that process execution. We should then page out its remaining pages, freeing all its allocated frames. This provision introduces a swap-in, swap-out level of intermediate CPU scheduling. In fact, look at any process that does not have enough frames. Although it is technically possible to reduce the number of allocated frames to the minimum, there is some number of pages in active use. If the process does not have this number of frames, it will quickly page fault. At this point, it must replace some page. However, since all its page are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, and again. The process continues to fault, replacing pages for which it then faults and brings back in right away.

Cause of Thrashing

The operating system monitors CPU utilization. If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system. A global page replacement algorithm is used; it replaces pages with no regard to the process to which they belong. Now suppose that a process enters a new phase in its execution and needs more frames. It starts faulting and taking frames away from other processes. These processes need those pages, however, and so they also fault, taking frames other processes. These faulting processes must use the paging device to swap pages in and out. As they queue up for the paging device, the ready queue empties. As process wait for the paging device, CPU utilization decreases.

The CPU scheduler sees the decreasing CPU utilization, and increases the degree of multiprogramming as result. The new process tries to get started by taking frames from running processes, causing more page faults, and a longer queue for the paging device. As a result, CPU utilization drops even further, and the CPU scheduler tries to increases tremendously. As a result, the effective memory access time increases. No work is getting done, because the processes are spending all their time paging.

To increase CPU utilization and stop thrashing, we must decrease the degree of multiprogramming. We can limit the effects of thrashing by using a local replacement algorithm. With local replacement, of one-process starts thrashing, it cannot steal frames from another process and cause the latter to thresh also. Pages are replaced with regard to the process of which they are a part. However, if



**Q16. Explain the concept of disk scheduling.**

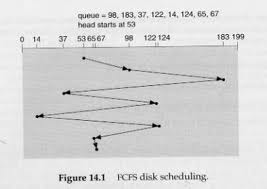
**Ans:** Disks provide the bulk of secondary storage for computer systems.there are several requests present in the disk queue. how the requests are processed i.e in which order the requests are processed is given by disk scheduling algorithms.

There are following disk scheduling algorithms used:

**1. FCFS(First come first serve):** It is the simplest type of disk scheduling. It processes in first come first serve manner. That is, the request is served first which has come first in disk queue.

It is easy to understand and implement but generally it does not provide the fastest service and hence the performance decreases.

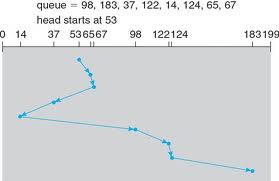
for ex, a disk queue has the requests for the blocks in this order: 98,183,37,122,14,124,65,67. If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183,122, 14,124,65, and finally to 67, for total head movement of 640.



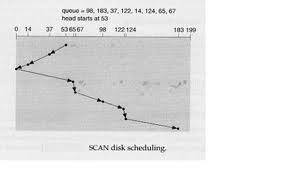
**2.SSTF(Shortest seek time first):** It selects the request with the minimum seek time from the current head position. It serves all the requests close to the current head position before moving the head far away to service other requests.

It gives the improvement in the performance but it may cause starvation of some requests.

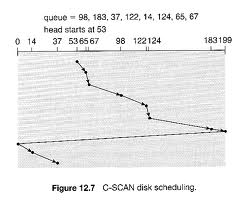
For example, a disk queue has the requests for the blocks in this order: 98,183,37,122,14,124,65,67. For our example request queue, the closest request to the initial head position 53 is at cylinder 65. Once we are at cylinder 65, the next closest request is at cylinder 67. From there the request at cylinder 37 is closer than 98, so 37 is served next. Continuing, we service the request at cylinder 14, then 98,122,124 and finally 183. This scheduling method results in a total head movement of only 236 cylinders.



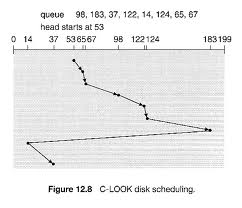
**3.SCAN scheduling:** in this algorithm, disk arm starts at one end of the disk, and moves towards the other end by serving the requests as it reaches each cylinder until it reaches the other end. At the other end, the direction of the head movement is reversed, and servicing continues.



**4.C-SCAN:** it is a variant of SCAN scheduling, which is designed to provide a much uniform wait time. Like SCAN, C-SCAN also moves the head from one end of the disk to the other end, but it immediately returns to the beginning of the disk without servicing any requests on the return trip.



**5.LOOK** in this scheduling, the arm goes only as far as the last request in a direction and then reverses direction immediately without first going all the way to the end of the disk.



**Q17. Explain Banker’s Algorithm with example.**

Ans: -The resource allocation graph algorithm is not applicable to a resource allocation system with multiple instances of each resource type. The deadlock avoidance algorithm that we describe next is applicable to such a system, but is less efficient than the resource allocation graph scheme. This algorithm is commonly known as the Banker’s algorithm. The name was chosen because this algorithm could be used in a banking system to ensure that the bank never allocates its available cash such that it can no longer satisfy the needs of all its customers.

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

Several data structures must be maintained to implement the banker’s algorithm. These data structures encode the state of the resource allocation system. Let n be the number of processes in the system and m be the number of resource types. We need the following data structures: -

* Available: - A vector of length m indicates the number of available resources of each type. If Available [j]=k, there are k instances of resource type Rj available.
* Max: - An n \* m matrix defines the maximum demand of each process. If max [i,j]=k, then process Pi may request at most k instances of resource type Rj
* Allocation: - An n \* m matrix defines the number of resources of each type currently allocated to each process. If allocation[i,j]=k, then process Pi is currently allocated k instances of resource type Rj.
* Need: - An n\* m matrix indicates the remaining resource need of each process. If need[i,j]=k, then process Pi may need k more instances of resource type Rj to complete its task. Note that Need[i,j]=Max[i]j-Allocation[i,j].

Example: -

Allocation Max Available Need

**A B C A B C A B C A B C**

**P0 0 1 0 7 5 3 3 3 2 7 4 3**

**P1 2 0 0 3 2 2 1 2 2**

**P2 3 0 2 9 0 2 6 0 0**

**P3 2 1 1 2 2 2 0 1 1**

**P4 0 0 2 4 3 3 4 3 1**

**P1: - Available= 332**

**Need = -122**

**210**

**Allocation = 200**

**Need =+122**

**322**

**Available = 210 New Available=532**

**MAX = +322**

**532**

**P3: - Available= 532**

**Need = -011**

**521**

**Allocation = 211**

**Need =+011**

**222**

**Available = 521 New Available=743**

**MAX = +222**

**743**

**P4: - Available= 743**

**Need = -431**

**312**

**Allocation = 002**

**Need =+431**

**433**

**Available = 312 New Available=745**

**MAX = +433**

**745**

**P0: - Available= 745**

**Need = -743**

**002**

**Allocation = 010**

**Need =+743**

**753**

**Available = 002 New Available=745**

**MAX = +743**

**745**

**P2: - Available= 745**

**Need = -600**

**145**

**Allocation = 302**

**Need =+600**

**902**

**Available = 145 New Available=1045**

**MAX = +902**

**1045**

**Q18. What is Disk storage? Explain Seek latency, Rotational latency and Transfer latency.**

Ans A disk drive consists of a disk pack assembly, which includes a drive mechanism and a read/write head mechanism. The disk pack has several platters in the for of phonograph records. Data is recorded using magnetic polarity on the surfaces of each platter. All the platters are fixed on to a rotating mechanism and they can move together as one unit. Many concentric tracks are made on the surfaces of the platters for recording information. These tracks are divided into many sectors. One sector may have the capacity to record many bytes of data, usually 512 bytes. The capacity of a sector is called disk block. One sector is the smallest unit of data that is transferred between the computer system and the disk drive. The actual number of sectors per track and bytes per sector are decided by the formatting done during manufacture. In some disk drives, this low level formatting is done under software control so that the capacity of the disk drive is changed to some extent whenever the low level formatting is done

The read/write assembly consist of one read/write head per surface. A disk drive system with 10 recording surface will have 10 read/write heads, all fitted to the same mechanical frame so that the movements of heads are all together. The heads over different surfaces move in either radial directions to position them over the tracks before starting sector or track read/write operation. This movement of heads to positions over a track is called head seek. The rotating motor in the drive mechanism positions the appropriate sectors over different surfaces under the read/write heads. The set of tracks over different surfaces that comes under a position of read/write heads is called a cylinder. So, a disk drive will have as many cylinders as there are tracks on a surface and the number of tracks on a cylinder will be exactly equal to the number of surfaces in disk drive.

**Seek Latency**: - This is the time taken to move the read/write heads from the present cylinder position to the new desired cylinder position from where the next read/write operation has to begin. This time depends on the number of cylinders between the present position and the new desired position of the head. An average value of seek latency is 10 milliseconds. To improve the performance of data transfer, track/cylinder skewing, is employed to account for the time in seeking adjacent track while transferring data on multiple sectors that span on the adjacent cylinder.

**Rotational Latency**: - This is the time taken for rotating the platter to bring the desired sector under the read/write head position. The actual value depends on the speed of the disk; 10 milliseconds is a typical value for rotational latency. Sectors may be interleaved to improve the performance of multiple sector data transfers in one operation. That is, while transferring the data of the previous sector between the disk buffer and the system, one or more sectors might have passed under the read/write head. So, to allow this delay In transferring, and to make available the next sector under the head when completing the previous transfer, the sectors are interleaved by one or more sector positions depending on the rotational speed of platters.

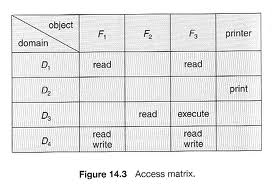
**Transfer latency**: - This is the time taken to scan the sector for transferring the data between memory and disk sector. Transfer latency is normally very low when compared to seek and rotational latencies.

### Q19. What is Access Matrix?

Ans Our model of protection can be viewed abstractly as a matrix, called an access matrix. The rows of the access matrix represent domains, and columns represent objects. Each entry in the matrix consists of a set of access rights. Because the column defines objects explicitly, we can omit the object name from the access right. The entry access defines the set of operations that a process, executing in domain Di, can invoke on object Oj.

There are four domains and four objects, three files and one laser printer. When a process executes in domain D1, it can read files F1 and F3. A process executing in domain D4 has the same privileges as it does in domain D1, but in addition, it can also write onto files F1 and F3. Note that the laser printer can be accessed only by a process executing in domain D2

The access matrix scheme provides us with the mechanism for specifying a variety of policies. The mechanism consists of implementing the access matrix and ensuring that the semantic properties we have outlined indeed hold. More specifically, we must ensure that a process executing in domain Di can access only those objects specified in row I, and then only as allowed by the access matrix entries.



The access matrix provides an appropriate mechanism for defining and implementing strict control for both the static and dynamic association between processes and domains. When we switch a process from one domain to another, we are executing an operation on an object. We can control domain switching by including domains among the objects of the access matrix. Similarly, when we change the content of the access matrix, we are performing an operation on an object: the access matrix. Again, we can control these changes by including the access matrix itself as an object. Actually, since each ectry in the access matrix may be modified individually, we must consider each entry in the access matrix as an object to be protected.

.

.

**Q20. What is critical section problem? Explain producer consumer problem**.

Ans A section of code or collection of operations in which only one process may be executing at a given time, is called critical section. Consider a system containing n processes {P0, P1, 2, ..., Pn }. Each process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file, and so on. The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. Thus, the execution of critical sections by the processes is mutually exclusive in time. The critical section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the entry section. The critical section may be followed by an exit section. The remaining code is the remainder section.

do

{

Entry Section

Critical Section

Exit Section

Remainder Section

} While (1);

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

1. Mutual Exclusion: - If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.
2. Progress: - If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision on which will enter its critical section next.
3. Bounded Waiting: - There exists a bound 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.

**PRODUCER CONSUMER PROBLEM**

Concurrent execution of cooperating processes requires mechanisms that allow processes to communicate with one another and to synchronization their actions. This is shown by a model called the producer consumer problem. A producer process produces information that is consumed by a consumer process. For example, a print program produces characters that are consumed by the printer driver.

To allow producer consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consumer an item that has not yet been produced.

Bounded Buffer: - The bounded buffer producer consumer problem assumes a fixed buffer size. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

Unbounded Buffer: - The unbounded buffer producer consumer problem places no practical limit on the size of buffer. The consumer may have to wait for new items, but the producer can always produce new items.