OPERATING SYSTEMS

Review_of_ISA_1

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Course Syllabus - Unit 1

Class No.	Chapter Title / Reference Literature	Topics to be covered	% of Portions covered	
			Reference chapter	Cumulative
1		Introduction: What Operating Systems Do, Computer-System Organization	1.1, 1.2	21.4
2		Computer-System Architecture, Operating-System Structure & Operations	1.3,1.4,1.5	
3		Kernel Data Structures, Computing Environments	1.10, 1.11	
4		Operating-System Services, Operating-System Design and Implementation	2.1, 2.6	
5		Process concept: Process in memory, Process State, Process Control Block, Context switch, Process Creation & Termination,	3.1 – 3.3	
6		CPU Scheduling - Preemptive and Non-Preemptive, Scheduling Criteria, FIFO Algrorithm	5.1-5.2	
7		Scheduling Algorithms:SJF, Round-Robin and Priority Scheduling	5.3	
8		Multi-Level Queue, Multi-Level Feedback Queue	5.3	
9		Multiprocessor and Real Time Scheduling	5.5, 5.6	
10		Case Study: Linux/ Windows Scheduling Policies.	5.7	
11		Inter Process Communication – Shared Memory, Messages	3.4	
12.		Named and unnamed pipes (+Review)	3.6.3	

OPERATING SYSTEMS

Course Syllabus - Unit 2

13	Introduction to Threads, types of threads, Multicore Programming, Multithreading Models	4.1 - 4.3	42.8
14	Thread creation, Thread Scheduling	5.4	
15	Pthreads and Windows Threads	4.4	
16	Mutual Exclusion and Synchronization: software approaches,	6.1-6.2	
17	principles of concurrency, hardware support	6.3-6.4	
18	Mutex Locks, Semaphores	6.5, 6.6	
19	Classic problems of Synchronization: Bounded-Buffer Problem, Readers-Writers problem	6.7-6.8	
20	Dining-Philosophers Problem	6.8	
21	Synchronization Examples: Synchronisation mechanisms provided by Linux/Windows/Pthreads.	6.9	
22	Deadlocks: principles of deadlock, Deadlock Characterization	7.1-7.3	
23	Deadlock Prevention, Deadlock example	7.4	
24	Deadlock Detection, Algorithm	7.6	

OPERATING SYSTEMS

Course Syllabus - Unit 3

25	Main Memory: Hardware and control structures, OS support, Address translation	8.1	64.2
26	Dynamic linking, Swapping	8.2	
27	Memory Allocation (Partitioning, relocation), Fragmentation	8.3	
28	Segmentation	8.4	
29	Paging: OS Support, TLBs, Address Translation	8.5	
30	Structure of the Page Table	8.6	
31	Design Alternatives – Inverted Page Tables, Bigger Pages	8.7-8.8	
32	Virtual Memory: Demand Paging, Copy-OnWrite	9.1-9.3	
33	Page replacement policy – LRU etc. (in comparison with FIFO and Optimal)	9.4	
34	Page Replacement (contd.), Frame allocation	9.4,9.5	
35	Thrashing	9.6	
36	Case Study: Linux/ Windows Memory Management	9.10	

- # Review of ISA-1

Review_of_ISA_1: Q1a

Clearly explain different states of the process drawing state diagram	4 Marks

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:

- **New**: The process is being created.
- **Running**: Instructions are being executed.
- **Waiting**:The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- **Ready**: The process is waiting to be assigned to a processor.
- **Terminated**: The process has finished execution.

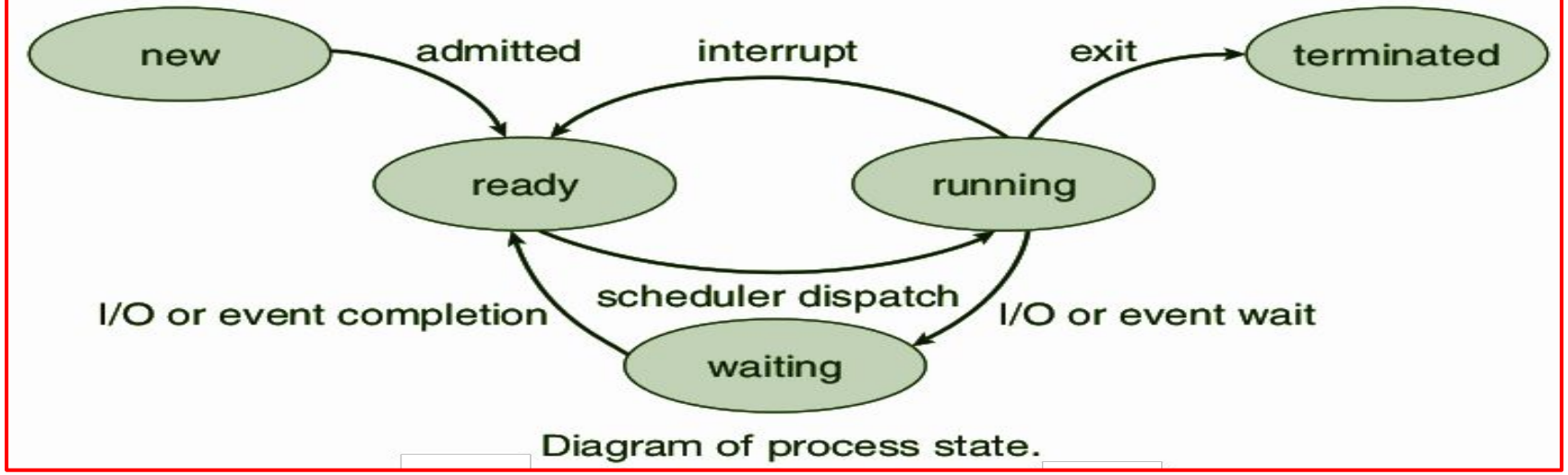


Diagram of process state.

Review_of_ISA_1: Q1b

Precisely Explain fork(), exec(), and wait() system calls including parameters and return value	3 Marks

The "fork()" system call

- A process calling fork()spawns a child process
- The child is almost an identical clone of the parent:Program Text (segment .text), Stack (ss), PCB (eg. registers), Data (segment .data)
- The fork()is called once, but returns twice!
- After fork()both the parent and the child are executing the same program.
    - pid<0:the creation of a child process was unsuccessful.
    - pid==0: the newly created child.
    - pid>0:the process ID of the child process passes to the parent or the process ID of the parent process in the parent section of the code

The "exec()" system call

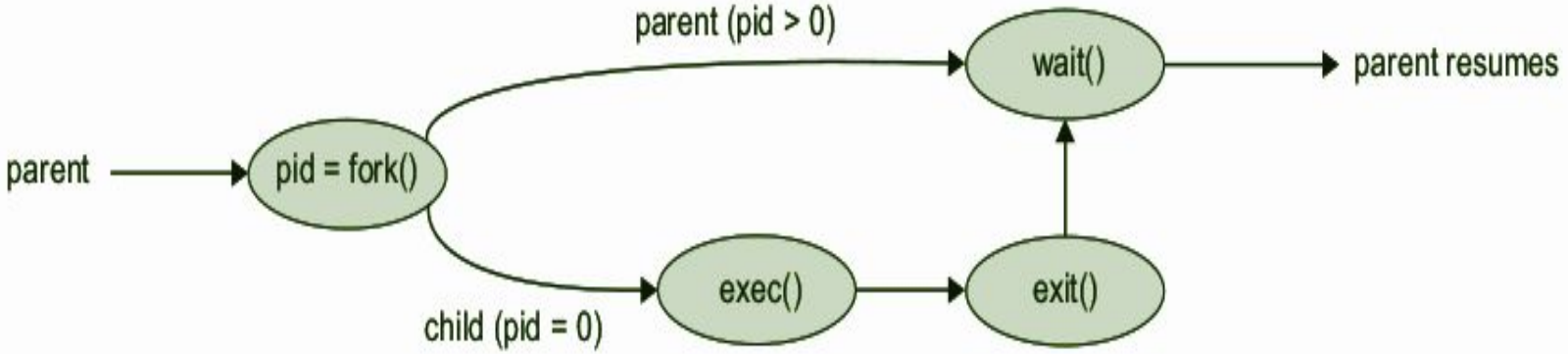
- The exec()call replaces a current process' image with a new one i.e. loads a new program within current process
- The new image is either regular executable binary file or a shell script.
- There's not a syscall under the name exec(). By exec()we usually refer to a family of calls
- Upon success, exec()never returns to the caller. It replaces the current process image, so it cannot return anything to the program that made the call. If it does return, it means the call failed. Typical reasons are: non-existent file (bad path) or bad permissions.

Review_of_ISA_1: Q1b

Precisely Explain fork(), exec(), and wait() system calls including parameters and return value	3 Marks

The "wait()" system call

- Forces the parent to suspend execution, i.e. wait for its children or a specific child to terminate).
- When the child process terminates, it returns an exit status to the operating system, which is then returned to the waiting parent process. The parent process then resumes execution.
- A child process that terminates but is never waited on by its parent becomes a **zombie** process. Such a process continues to exist as an entry in the system process table even though it is no longer an actively executing program.
- The return value is:PID of the exited process, if no error(-1) if an error has happened



Review_of_ISA_1: Q1c

Precisely and Clearly Explain Primary benefits of Multiprocessor Based Systems	3 Marks

Multiprocessor systems have three main advantages

- **Increased throughput:** By increasing the number of processors, we expect to get more work done in less time. The speed-up ratio with N processors is not N , however; rather, it is less than N . When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead, plus contention for shared resources, lowers the expected gain from additional processors. Similarly, N programmers working closely together do not produce N times the amount of work a single programmer would produce

- **Economy of scale:** Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies. If several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them than to have many computers with local disks and many copies of the data

- **Increased reliability:** If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slows it down. If we have ten processors and one fails, then each of the remaining nine processors can pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether.

Review_of_ISA_1: Q2a

Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed as Burst Time. In answering the questions, use non-preemptive scheduling, and base all decisions on the information you have at the time the decision must be made. Process Arrival Time Burst Time P1 0.0 8 P2 0.4 4 P3 1.0 1 1. What is the average turnaround time for these processes with the First Come First Serve (FCFS) scheduling algorithm? 2. What is the average turnaround time for these processes with the Shortest Job First (SJF) scheduling algorithm? 3. The SJF algorithm is supposed to improve performance, but notice that we chose to run process P1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P1 and P2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as future-knowledge scheduling.	5 Marks

Review_of_ISA_1: Q2a

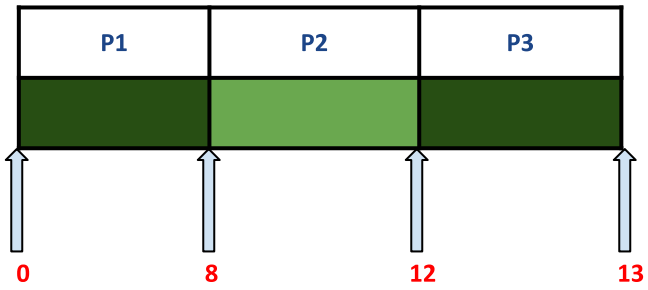
Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed as Burst Time. In answering the questions, use non-preemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
P1	0.0	8
P2	0.4	4
P3	1.0	1

1. What is the average turnaround time for these processes with the First Come First Serve (FCFS) scheduling algorithm?

5 Marks

GANTT Chart



Process ID	Arrival Time	Response Time	Waiting Time	Turn Around Time
P1	0.0	0.0	0.0	8
P2	0.4	8-0.4 =>7.6	8-0.4 =>7.6	12-0.4=>11.6
P3	1.0	12-1.0=>11	12-1.0=>11	13-1.0=>12
Average	0.46	18.6/3 =>6.2	18.6/3=>6.2	31.6/3 = > 10.53
Remarks	AAT	ART	AWT	ATAT

Final Answer
10.53

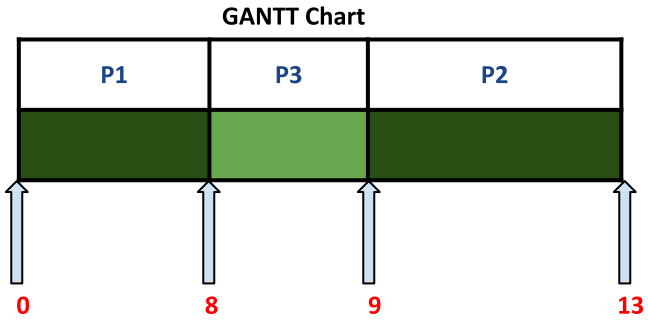
Review_of_ISA_1: Q2a

Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed as Burst Time. In answering the questions, use non-preemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
P1	0.0	8
P2	0.4	4
P3	1.0	1

2. What is the average turnaround time for these processes with the Shortest Job First (SJF) scheduling algorithm?

5 Marks



Process ID	Arrival Time	Response Time	Waiting Time	Turn Around Time
P1	0.0	0.0	0.0	8
P2	0.4	9-0.4 =>8.6	9-0.4 =>8.6	13-0.4=>12 .6
P3	1.0	8-1.0=>7	8-1.0=>7	9-1.0=>8
Average	0.46	15.6/3 =>5.2	15.6/3 =>5.2	28.6/3 = > 9.53
Remarks	AAT	ART	AWT	ATAT

Final Answer
9.53

Review_of_ISA_1: Q2a

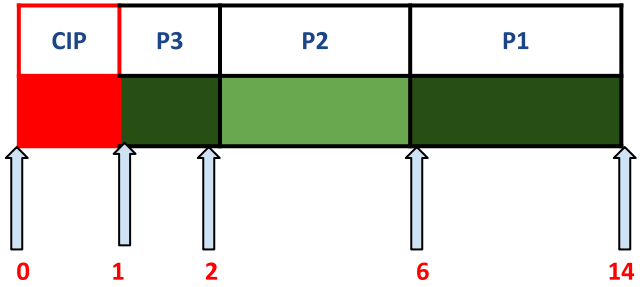
Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed as Burst Time. In answering the questions, use non-preemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
P1	0.0	8
P2	0.4	4
P3	1.0	1

3. The SJF algorithm is supposed to improve performance, but notice that we chose to run process P1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P1 and P2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as future-knowledge scheduling.

5 Marks

GANTT Chart



Final Answer

6.86

Process ID	Arrival Time	Response Time	Waiting Time	Turn Around Time
CIP	0	0	0	1
P1	0.0	6.0	6.0-0 =>6.0	14
P2	0.4	2-0.4 =>1.6	2-0.4 =>1.6	6-0.4=>5.6
P3	1.0	1-1.0=>0	1-1.0=>0	2-1.0=>1
Average	0.46	7.6/3 =>2.53	7.6/3 =>2.53	20.6/3=>6.86
Remarks	AAT	ART	AWT	ATAT

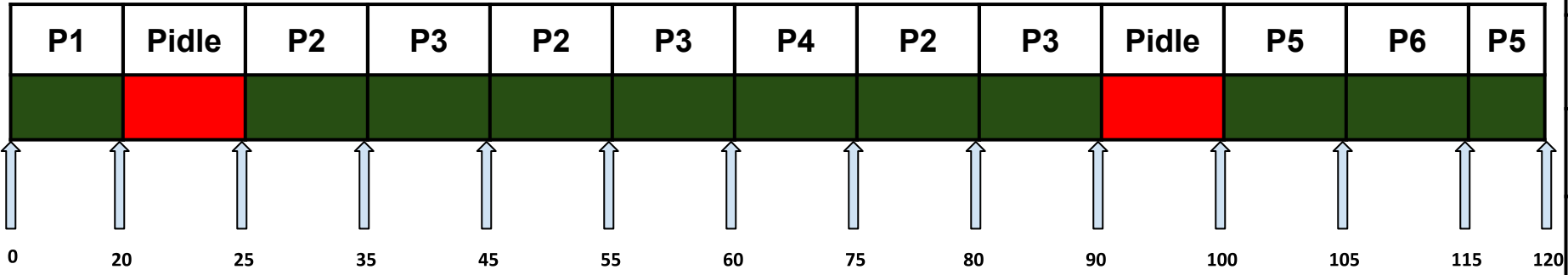
Review_of_ISA_1: Q2b

The following processes are being scheduled using a pre-emptive, round-robin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an idle task (which consumes no CPU resources and is identified as Pidle). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue 1. What is the turnaround time for each thread? 2. What is the waiting time for each thread? 3. What is the CPU utilization rate?	5 Marks

Thread	Priority	Burst	Arrival
P1	40	20	0
P2	30	25	25
P3	30	25	30
P4	35	15	60
P5	5	10	100
P6	10	10	105

Review_of_ISA_1: Q2b

Thread	Priority	Burst	Arrival
P1	40	20	0
P2	30	25	25
P3	30	25	30
P4	35	15	60
P5	5	10	100
P6	10	10	105



Process ID	Arrival Time	Response Time	Waiting Time	Turn Around time
P1	0	0	0	20
P2	25	0	75-25-20 => 30	80-25=> 55
P3	30	35-30=5	80-30-15=>35	90-30=> 60
P4	60	60-60=>0	60-60=> 0	75-60=> 15
P5	100	100-100=>0	115-100-5=> 10	120-100=> 20
P6	105	105-105=>0	105-105=> 0	115-105=> 10
Average	53.33	0.83	12.5	30
Remarks	AAT	ART	AWT	ATAT

3. (120-Pidle)/120 = 87.5 percent.
Pidle => 5+10 =>15
(120-15)/120 =>87.5

1. What is the turnaround time for each thread?
2. What is the waiting time for each thread?
3. What is the CPU utilization rate?

Review_of_ISA_1: Q3a

Precisely and Clearly Explain Four necessary Conditions to occur simultaneously for Deadlock	4 Marks

A deadlock situation can arise if the following four conditions hold simultaneously in a system

- **Mutual exclusion:** At least one resource must be held in a nonsharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
- **Hold and wait:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
- **No Preemption:** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular Wait:** A set {P0 , P1 , ..., Pn } of waiting processes must exist such that P0 is waiting for a resource held by P1 , P1 is waiting for a resource held by P2 , ..., Pn−1 is waiting for a resource held by Pn , and Pn is waiting for a resource held by P0 .

Review_of_ISA_1: Q3b

Precisely and Clearly Explain primary benefits of multithreading and its applications	3 Marks

The benefits of multithreaded programming can be broken down into four major categories

- **Responsiveness:** Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. This quality is especially useful in designing user interfaces. For instance, consider what happens when a user clicks a button that results in the performance of a time-consuming operation. A single-threaded application would be unresponsive to the user until the operation had completed. In contrast, if the time-consuming operation is performed in a separate thread, the application remains responsive to the user

- **Resource Sharing:** Processes can only share resources through techniques such as shared memory and message passing. Such techniques must be explicitly arranged by the programmer. However, threads share the memory and the resources of the process to which they belong by default. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.

- **Economy:** Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads. Empirically gauging the difference in overhead can be difficult, but in general it is significantly more time consuming to create and manage processes than threads.

- **Scalability:** The benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores. A single-threaded process can run on only one processor, regardless how many are available.

Review_of_ISA_1: Q3c

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

Yes, this system is deadlock-free. Proof by contradiction. Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

Review_of_ISA_1: Q4a

Give the correct and concise structure of reader and write process using semaphores for the first reader writer problem where no reader be kept waiting unless a writer has already obtained permission to use the shared object. Obviously, readers can access the shared object simultaneously.	5 Marks

```
do {
   wait(rw_mutex);
      . . .
   /* writing is performed */
      . . .
   signal(rw_mutex);
} while (true);
```

The structure of a writer process.

```
do {
   wait(mutex);
   read_count++;
   if (read_count == 1)
      wait(rw_mutex);
   signal(mutex);
      . . .
   /* reading is performed */
      . . .
   wait(mutex);
   read_count--;
   if (read_count == 0)
      signal(rw_mutex);
   signal(mutex);
} while (true);
```

The structure of a reader process.

Review_of_ISA_1: Q4b

Give a clear, correct and complete algorithm for deadlock detection among n processes and m resources type with multiple instances. Mention the complexity of the algorithm.	5 Marks

1. Let ***Work*** and ***Finish*** be vectors of length m and n, respectively. Initialize ***Work*** = ***Available***. For $i = 0, 1, ..., n-1$, if ***Allocation***$_i \neq 0$, then ***Finish***$[i]$ = ***false***. Otherwise, ***Finish***$[i]$ = ***true***.
2. Find an index i such that both
   a. ***Finish***$[i]$ == ***false***
   b. ***Request***$_i \leq$ ***Work***

   If no such i exists, go to step 4.
3. ***Work*** = ***Work*** + ***Allocation***$_i$
   Finish$[i]$ = ***true***
   Go to step 2.
4. If ***Finish***$[i]$ == ***false*** for some i, $0 \leq i < n$, then the system is in a deadlocked state. Moreover, if ***Finish***$[i]$ == ***false***, then process P_i is deadlocked.

This algorithm requires an order of $m \times n^2$ operations to detect whether the system is in a deadlocked state.

Review_of_ISA_1: Q5a

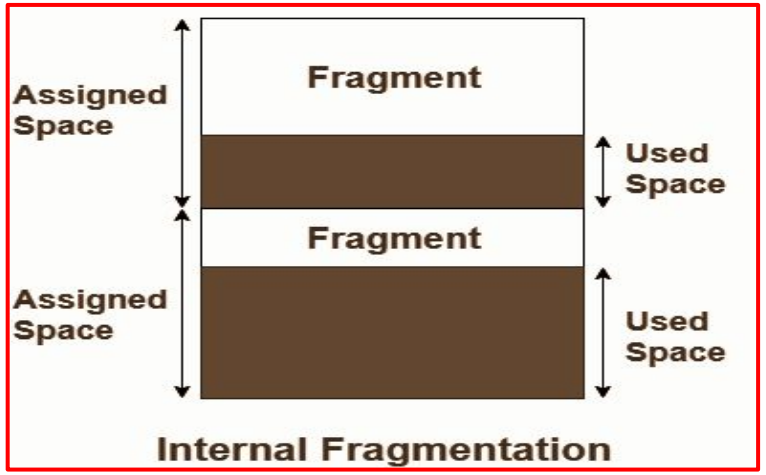
Clearly Explain all the Steps to Service a Page Fault in a demand paging based system	4 Marks

1. Trap to the operating system.
2. Save the user registers and process state.
3. Determine that the interrupt was a page fault.
4. Check that the page reference was legal and determine the location of the page on the disk.
5. Issue a read from the disk to a free frame:
   a. Wait in a queue for this device until the read request is serviced.
   b. Wait for the device seek and/or latency time.
   c. Begin the transfer of the page to a free frame.
6. While waiting, allocate the CPU to some other user (CPU scheduling, optional).
7. Receive an interrupt from the disk I/O subsystem (I/O completed).
8. Save the registers and process state for the other user (if step 6 is executed).
9. Determine that the interrupt was from the disk.
10. Correct the page table and other tables to show that the desired page is now in memory.
11. Wait for the CPU to be allocated to this process again.
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction.

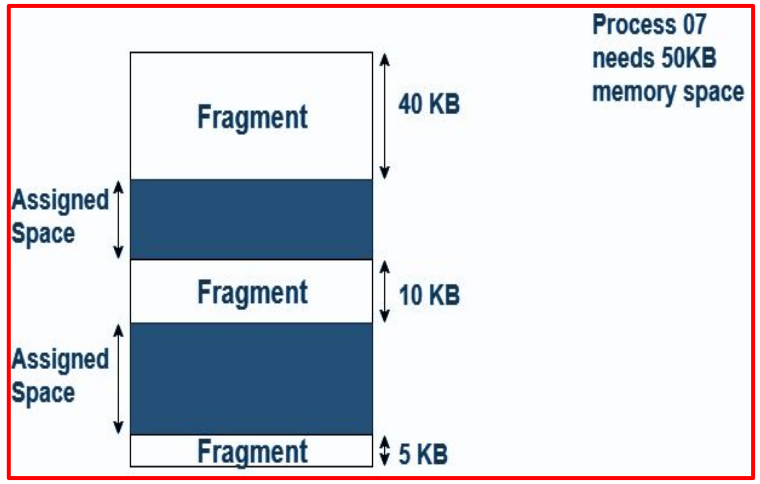
Review_of_ISA_1: Q5b

Clearly Differentiate between internal fragmentation and external fragmentation with example	3 Marks

Internal Fragmentation:
Internal fragmentation happens when the memory is split into mounted sized blocks. Whenever a method request for the memory, the mounted sized block is allotted to the method. just in case the memory allocated to the method is somewhat larger than the memory requested, then the distinction between allotted and requested memory results in the Internal fragmentation.



External fragmentation:
Occurs when there's a sufficient quantity of area within the memory to satisfy the memory request of a method. However the process's memory request cannot be fulfilled because the memory offered is in a non-contiguous manner.

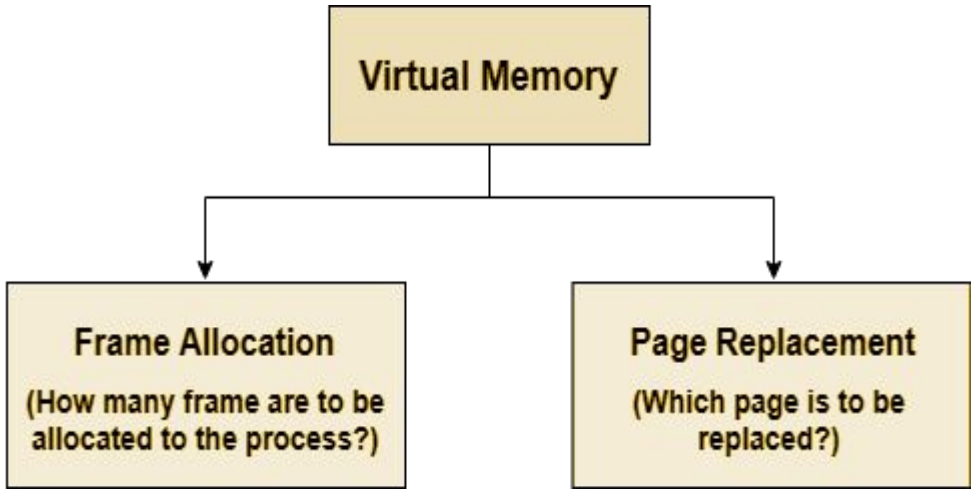


Review_of_ISA_1: Q5c

Clearly explain and Differentiate between Frame Allocation and Page replacement	3 Marks

Frame allocation is all about how many frames are to be allocated to the process while the page replacement is all about determining the page number which needs to be replaced in order to make space for the requested page.

The **Page Replacement** decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault)



Review_of_ISA_1: Q6a

Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how112 KB, and 426 KB (in order) ? Which algorithm makes the most efficient use of memory? would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB,1 12 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory ?	**5 Marks**

First- fit

212 K is put in 500 K partition.

417 K is put in 600 K partition.

112 K is put in 288 K partition. (New partition 288 K = 500 K - 212 K)

426 K must wait.

Best-fit

212 K is put in 300 K partition.

417 K is put in 500 K partition.

112 K is put in 200 K partition.

426 K is put in 600 K partition.

Worst-fit

212 K is put in 600 K partition.

417 K is put in 500 K partition.

112 K is put in 388 K partition. (600 K - 212 K)

426 K must wait.

In this example **Best-fit** is the **best solution**.

Review_of_ISA_1: Q6b

Consider the segment table as follows :

Segment	Base	Limit
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1352	96

What are the Physical address for the following logical address

Seg	Offset
0	430
1	10
2	500
3	400
4	112

5 Marks

1. 219 + 430 = 649
2. 2300 + 10 = 2310
3. illegal reference, trap to operating system
4. 1327 + 400 = 1727
5. illegal reference, trap to operating system

- # Review of ISA-1

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com