F – 14371

Register No. | | | | | | | | |

BTech Degree Examination December 2022

Fifth Semester

Information Technology

20ITT52 – OPERATING SYSTEMS

(Regulations 2020)

Time: Three hours | Maximum: 100 marks

Answer all Questions

Part – A (10 × 2 = 20 marks)

1. A process executes the code [CO1,K3]

```
fork ( );
fork ( );
fork ( );
fork ( );
fork ( );
```

Identify the total number of processes created.

2. Specify any four services provided by an operating system. [CO1,K1]

3. List out the process states available. [CO2,K1]

4. Find the maximum number of processes that can be in the ready queue and running state in a single processor system with 'n' processes. [CO2,K3]

5. Consider three concurrent processes P1, P2 and P3 as shown below, which access a shared variable D that has been initialized to 100. [CO3,K3]

| P1 | P2 | P3 |
|---|---|---|
| . | . | . |
| . | . | . |
| . | . | . |
| D=D+20 | D=D-50 | D=D+10 |
| . | . | . |
| . | . | . |
| . | . | . |

The process is executed on a uniprocessor system running a time shared operating system. If the minimum and maximum possible values of D after the three processes have completed execution are X and Y respectively then, find the value of Y-X?

6. List any two examples of deadlocks that are not related to a computer system environment. [CO3,K1]

7. Consider a logical address space of 128 pages of 1,024 words each, mapped into a physical memory of 32 frames. How many bils are there in the logical address and in the physical address? [CO4,K3]

8. Sketch the memory hierarchy in computer architecture. [CO4,K1]

9. There are 200 tracks on a disc platter and the pending requests have come in the order 36, 69, 167, 76, 42, 51, 126, 12 and 199. Assume the arm is located at the 100 track and moving towards track 199. If required of disc access is 126, 167, 199, 12, 36, 42, 51, 69 and 76 then which disc access scheduling policy is used? [CO5,K3]

10. What are the most common schemes for defining the Logical Structure of a Directory? [CO5,K1]

Part – B (5 × 16 = 80 marks)

11. a. i) What are the various operating system services? Also provide overview about it. (8) [CO1,K1]

ii) Explain the role of linkers and loaders in an operating system. (8) [CO1,K2]

(OR)

b. i) What are the various system calls and their applications in operating systems? (8) [CO1,K1]

ii) Give a detailed analysis of various operating system structures. (8) [CO1,K2]

12. a. i) Illustrate two fundamental models of interprocess communication. (8) [CO2,K2]

ii) Consider the following set of processes, with the length of the CPU burst given in milliseconds. (8) [CO2,K3]

| Process | Execution time | Arrival time | Priority |
|---|---|---|---|
| P1 | 11 | 0 | 2 |
| P2 | 3 | 0 | 1 |
| P3 | 9 | 5 | 5 |
| P4 | 4 | 2 | 4 |
| P5 | 9 | 1 | 3 |

Draw Gantt charts and calculate turnaround time and waiting time for the following algorithms

(i) FCFS (ii) SIF (iii) SRTF (iv) Pre-emptive priority.

(OR)

b. i) Describe the actions taken by a kernel to context-switch between processes. (8) [CO2,K2]

ii) Consider the 3 processes P1, P2 and P3 shown in the table. (8) [CO2,K3]

| Process | Arrival time | Time units required |
|---|---|---|
| P1 | 0 | 5 |
| P2 | 1 | 7 |
| P3 | 3 | 4 |

Write the completion order of the 3 processes under the policies (i) FCFS (ii) SJF (iii) RR with quantum of 2 units (iv) SRTF

13. a. i) The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as S0=1, S1=0, S2=0. (8) [CO3,K3]

| Process P0 | Process P1 | Process P2 |
|---|---|---|
| While (true)<br>{<br>wait (S0);<br>print '0';<br>release (S1);<br>release (S2);<br>} | wait (S1);<br>release (S0); | Wait (S2);<br>Release (S0); |

Calculate how many times the process P0 will print'0'.

ii) The following two functions P1 and P2 that share a variable B with an initial value of 2 execute concurrently. (8) [CO3,K3]

```
P1( )                P2 ( )
{                    {
  C=B-1;               D=2*B;
  B=2*C;               B=D-1;
}                    }
```

Find the number of distinct values that B can possibly take after the execution.

(OR)

b. Consider the following snapshot of a system. (16) [CO3,K3]

| | *Allocation* | *Max* |
|---|---|---|
| | *A B C D* | *A B C D* |
| $P_0$ | 3 0 1 4 | 5 1 1 7 |
| $P_1$ | 2 2 1 0 | 3 2 1 1 |
| $P_2$ | 3 1 2 1 | 3 3 2 1 |
| $P_3$ | 0 5 1 0 | 4 6 1 2 |
| $P_4$ | 4 2 1 2 | 6 3 2 5 |

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

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

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

14. a. i) Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB and 125 KB ( in order) how would the First-fit, Best-fit and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 300 KB and 375 KB ( in order)? (8) [CO4,K3]

ii) Explain the techniques used to divide the page table into smaller units. (8) [CO4,K2]

(OR)

b. i) Consider the following page reference string: (8) [CO4,K3]

1, 5, 2, 3, 4, 5, 1, 2, 5, 1, 2, 3, 4, 5, 4

1. How many page faults would occur for the following replacement algorithms, assuming frame size is three? All frames are initially empty, so first unique pages will cost one fault each.

   a. FIFO b. Optimal c. LRU

2. Does this example exhibit Belady's anomaly?

ii) Draw segmentation hardware and explain its working. (8) [CO4,K2]

15. a. Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. If the disk head is start at 53, then find out the total head movement with respect to FCFS, SSTF, SCAN, C-SCAN and LOOK, C-LOOK scheduling. (16) [CO5,K3]

(OR)

b. Consider a file currently consisting of 100 blocks. Assume that the file-control block (and the index block, in the case of indexed allocation) is already in memory. Calculate how many disk I/O operations are required for contiguous, linked and indexed (single level) allocation strategies, for one block, the following conditions hold. In the contiguous-allocation case, assume that there is no room to grow at the beginning, but there is room to grow at the end. Also assume that the block information to be added is stored in memory. (16) [CO5,K3]

a. The block is added at the beginning

b. The block is added in the middle

c. The block is added at the end.

d. The block is removed from the beginning

e. The block is removed from the middle

f. The block is removed from the end.

| Bloom's Taxonomy Level | Remembering (K1) | Understanding (K2) | Applying (K3) | Analysing (K4) | Evaluating (K5) | Creating (K6) |
|---|---|---|---|---|---|---|
| Percentage | 14 | 27 | 59 | - | - | - |

**KONGU ENGINEERING COLLEGE, PERUNDURAI - 638 060**
**Deparment of Information Technology**
**BTechDegee Examination December 2022**
**20ITT52-OPERATING SYSTEMS-Answer Key**
**(Regulations 2020)**

.

## PART-A

1.$2^n=2^{5=}$32 process created

2. Operating system services: User interface, Program Execution, I/O operations, File system manipulation,Communication,Resourceallocation,errordetection,Accounting,Protection and security.

3. New, Ready, Running, waiting and terminated state.

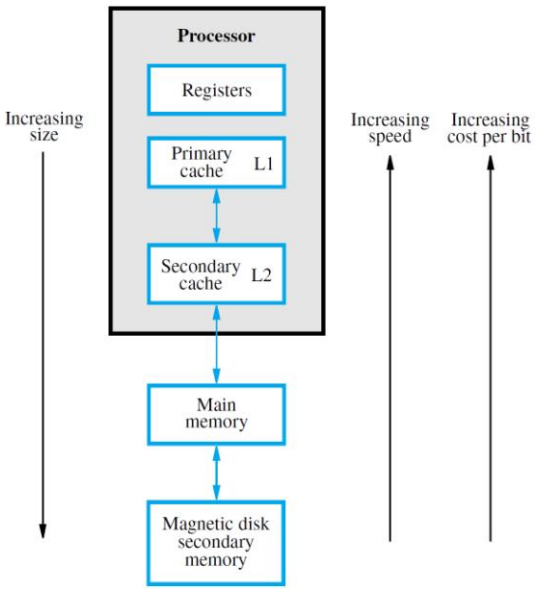
4. Independent of n

5. Y - X = 130 - 50 = 80

6. Two cars crossing a single-lane bridge from opposite directions.
A person going down a ladder while another person is climbing upthe ladder.
Two trains traveling toward each other on the same track.

7. Logical address space has 128 pages, size of each page, offset = 1024 words
Number if bits in page# field of the logical address = $\log_2$128 bits = 7 bits
Offset bits = $\log_2$1024 = 10 bits
So, **Logical address = 7 + 10 = 17 bits**
Physical memory has 32 frames, offset = 1024 words
Number of bits in frame# field of the Physical address = $\log_2$32 bits = 5 bits
**Physical address = 5 + 10 = 15 bits**

8. Memory Hierarchy

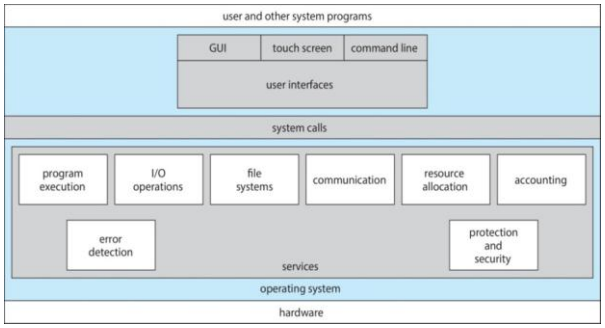


9. C-SCAN
10. Single level directory
    - Two-level directory
    - Tree structured directory
    - Acyclic graph directories

## PART-B

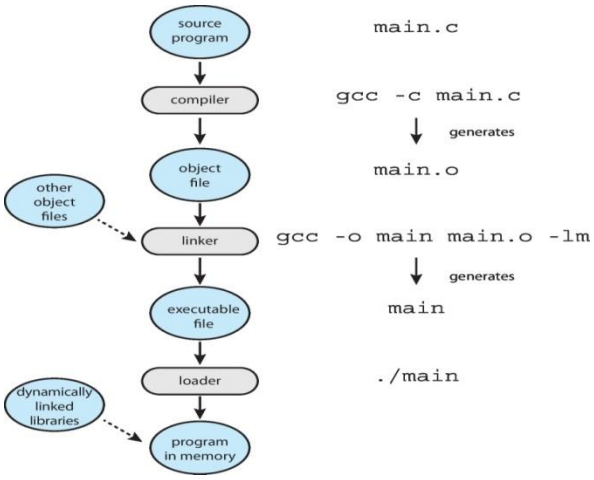
**11 a. i. Operating system services:**

- Operating systems provide an environment for execution of programs and services to programs and users
- One set of operating-system services provides functions that are helpful to the user:
    - **User interface** - Almost all operating systems have a user interface (**UI**).
        - Varies between **Command-Line** (**CLI**), **Graphics User Interface** (**GUI**), **touch-screen, Batch**
    - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
    - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
    - **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
    - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
        - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
    - **Logging** - To keep track of which users use how much and what kinds of computer resources
    - **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
        - **Protection** involves ensuring that all access to system resources is controlled
        - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts



**11a ii) Role of linkers and loaders**

- Source code compiled into object files designed to be loaded into any physical memory location – relocatable object file
- Linker combines these into single binary executable file
    - Also brings in libraries
- Program resides on secondary storage as binary executable
- Must be brought into memory by loader to be executed
    - Relocation assigns final addresses to program parts and adjusts code and data in program to match those addresses
- Modern general purpose systems don't link libraries into executables
    - Rather, dynamically linked libraries (in Windows, DLLs) are loaded as needed, shared by all that use the same version of that same library (loaded once)
- Object, executable files have standard formats, so operating system knows how to load and start them




**11 b.i) Various System calls**

- **Process control**
    - create process, terminate process
    - end, abort
    - load, execute
    - get process attributes, set process attributes
    - wait for time
    - wait event, signal event
    - allocate and free memory
    - Dump memory if error
    - Debugger for determining bugs, single step execution
    - Locks for managing access to shared data between processes

- **Information maintenance**
    - get time or date, set time or date
    - get system data, set system data
    - get and set process, file, or device attributes
- **Communications**
    - create, delete communication connection

- send, receive messages if message passing model to host name or process name
  - From client to server
- Shared-memory model create and gain access to memory regions
- transfer status information
- attach and detach remote devices

- **Protection**
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

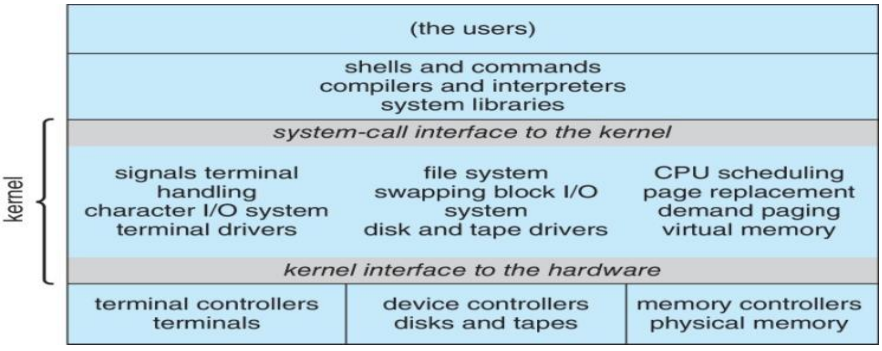
**11bii) Operating system structure**

- Simple structure – MS-DOS
- Monolithic Structure – UNIX
- Monolithic plus modular design – Linux
- Layered – an abstraction
- Microkernel – Mach

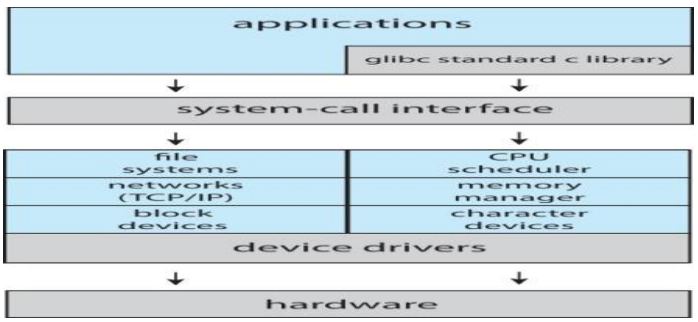
Monolithic Structure – Original UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

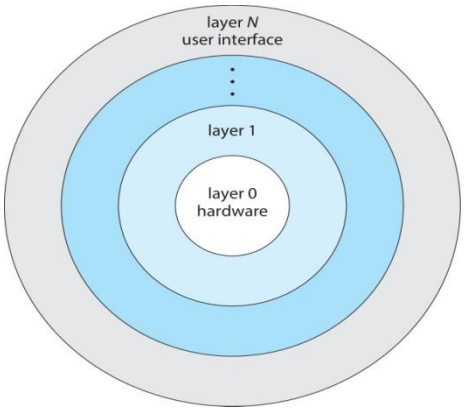
**Traditional UNIX Structure**




**Linux System Structure**



**Layered Approach**

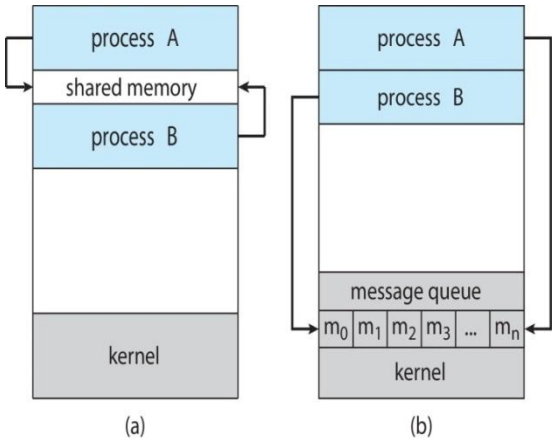



**12a.i) Inter process communication**

- Processes within a system may be *independent*or *cooperating*
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need interprocess communication (IPC)
- Two models of IPC
  - Shared memory
  - Message passing

**Communications Models**

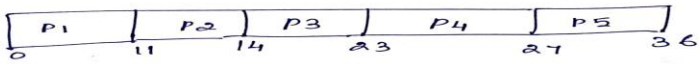
**a) Shared memory.** **(b) Message passing.**



12 a) ii)

FCFS scheduling

| Process | E.T | A.T | priority |
|---|---|---|---|
| P1 | 11 | 0 | 2 |
| P2 | 3 | 0 | 1 |
| P3 | 9 | 5 | 5 |
| P4 | 4 | 2 | 4 |
| P5 | 9 | 1 | 3 |

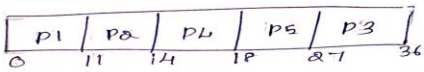



Waiting time (P1) = 0
W.T (P2) = 11
W.T (P5) = 27
W.T (P3) = 14
W.T (P4) = 23

Turn around time = ~~W.T + E.T~~ E.T − A.T

Turnaround (P1) = 11 − 0 = 11
T (P2) = 3 − 0 = 3
T (P3) = 9 − 5 = 4
T (P4) = 4 − 2 = 2
T (P5) = 9 − 1 = 8

ii) SJF scheduling

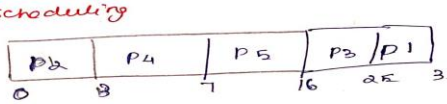



W.T (P1) = 0
W.T (P2) = 11 − 0 = 11
W.T (P3) = 27 − 5 = 22
W.T (P4) = 14 − 2 = 12
W.T (P5) = 18 − 1 = 17

T.T (P1) = E.T − Arrival time
= 11 − 0 = 11

P2 = 11
P3 = 22
P4 = 12
P5 = 17

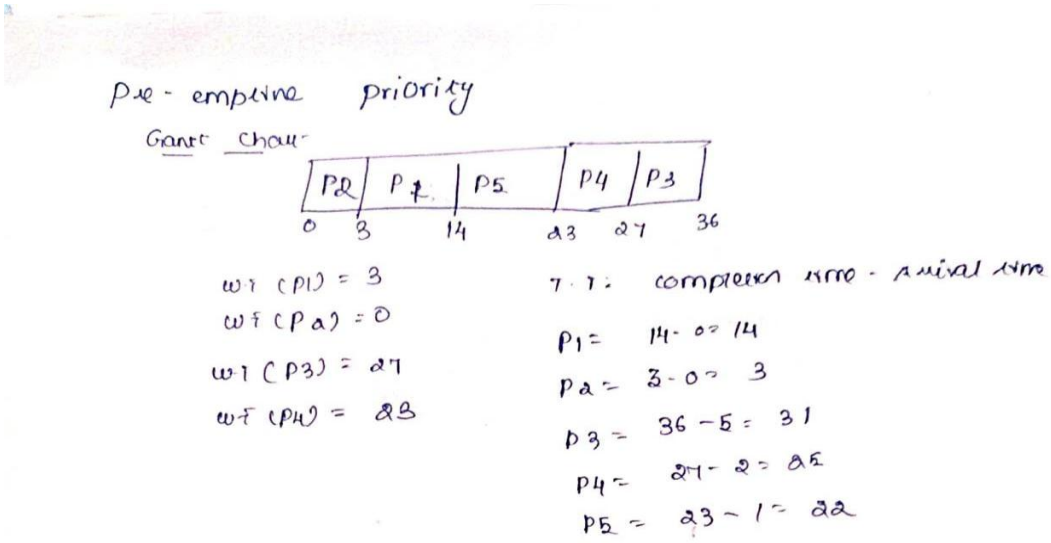
iii) SRTF scheduling




W.T (P1) = 25
W.T (P2) = 0
W.T (P3) = 16
W.T (P4) = 3
W.T (P5) = 7

T.T (P1) = completion time − Arrival time

T.T (P1) = 33 − 0 = 33
P2 = 3 − 0 = 3
P3 = 25 − 5 = 20
P4 = 7 − 2 = 5
P5 = 16 − 1 = 15




**12b).i. Action taken by a kernel to context switch between process**

- The OS must save the PC and user stack pointer of the currently executing process, in response to a clock interrupt and transfers control to the kernel clock interrupt handler

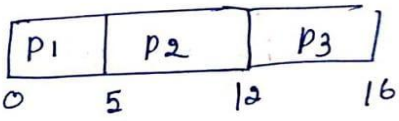
- Saving the rest of the registers, as well as other machine state, such as the state of the floating point registers, in the process PCB is done by the clock interrupt handler.

- The scheduler to determine the next process to execute is invoked the OS.

- Then the state of the next process from its PCB is retrieved by OS and restores the registers. The restore operation takes the processor back to the state in which the previous process was previously interrupted, executing in user code with user-mode privileges.

12 b) ii) FCFS Scheduling

Gantt chart

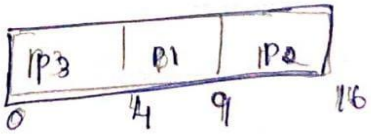



W.T(P1) = 0 W.T(P2) = 5 W.T(P3) = 12

order of execution = P1, P2, P3

SJF Scheduling

Gantt chart



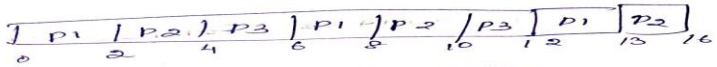

W.T(P1) = 4 - 0 = 4

W.T(P2) = 9 - 1 = 8

W.T(P3) = 3 - 0 = 3

order of execution is P3, P1, P2

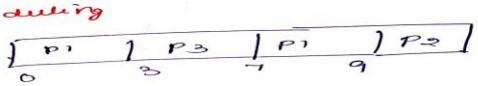
RR with quantum of 2




order of execution of process

P3, P1, P2

SRTF scheduling




order of execution of process

P3, P1, P2

**13a)i.** Initially only P0 can go inside the while loop as S0 = 1, S1 = 0, S2 = 0.
Minimum no. of time 0 printed is twice when execute in this order (p0 -> p1 -> p2 -> p0)
Maximum no. of time 0 printed is thrice when execute in this order (p0 -> p1 -> p0 -> p2 -> p0).
In semaphore, 0- hold and 1 – free

| Process | Semaphore | Initialization | Execution | Semaphore values | Comment |
|---|---|---|---|---|---|
| P0 | S0 | 1 | P0 executes, it print 0;<br>Now S0 value is 1.It release S1 and S2 , so the values of S1 and S2 changed as 1 | 0 | Now, both P1 and P2 have semaphore values as 1; If P1 is executing it change S0 as 1; And P1 complete its operation |
| P1 | S1 | 0 | 1 | 1 | |
| P2 | S2 | 0 | 1 | 1 | |

**If the execution order is P0 -> P1 -> P2 -> P0, then**

| Process | Sema phore | Sema phore values | Execution | Sema phore values | Comment |
|---|---|---|---|---|---|
| P0 | S0 | 1 | Now, both P0 and P2 have semaphore values as 1; If P2 is executing it change S0 as 1(already S0 as 1 because of the Process P1); And P2 complete its operation | 1 | Now P0 is executes, it print 0; **So, two times 0 is printed** |
| P1 | S1 | 0 | | 0 | |
| P2 | S2 | 1 | | 0 | |

**If the execution order is P0 -> P1 -> P0 -> P2 -> P0**

| Process | Semaphore | Sema phore values | Comment | Sema phore values | Comment | Sema phore values | Comment |
|---|---|---|---|---|---|---|---|
| P0 | S0 | 1 | Now, both P0 and P2 have semaphore values as 1; If P1 is executing it print 0; So far P2 is not executed | 0 | Now P2 is executes, it set S0 as 1 and P2 completes execution | 1 | Now P0 is executes, it print 0; **So, three times 0 is printed** |
| P1 | S1 | 0 | | 0 | | 0 | |
| P2 | S2 | 1 | | 1 | | 0 | |

**13a)ii.**

1. Execute P2 process after P1 process, then B = 3
2. Execute P1 process after P2 process, then B = 4
3. Pre-empting P1 and executing P2 processes B=3
4. Pre-empting P2 and executing P1 processes B=2

So, total no. of distinct values that B can possibly take after the execution is 3. (OR)

13.b. a. Not safe. Processes P2, P1, and P3 are able to finish, but no remaining processes can finish.
b. Safe. Processes P1, P2, and P3 are able to finish. Following this, processes P0 and P4 are also able to finish.

**14 a)i.)**
**First -Fit:**

a. 115 KB is put in 300-KB partition, leaving 185 KB, 600 KB, 350 KB, 200 KB, 750 KB, 125 KB

b. 500 KB is put in 600-KB partition, leaving 185 KB, 100 KB, 350 KB,200 KB, 750 KB, 125 KB

c. 358 KB is put in 750-KB partition, leaving 185 KB, 100 KB, 350 KB,200 KB, 392 KB, 125 KB

d. 200 KB is put in 350-KB partition, leaving 185 KB, 100 KB, 150 KB,200 KB, 392 KB, 125 KB

e. 375 KB is put in 392-KB partition, leaving 185 KB, 100 KB, 150 KB,200 KB, 17 KB, 125 KB

**Best-Fit**

a.115 KB is put in 125-KB partition, leaving 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, 10 KB

b. 500 KB is put in 600-KB partition, leaving 300 KB, 100 KB, 350 KB,200 KB, 750 KB, 10 KB

c. 358 KB is put in 750-KB partition, leaving 300 KB, 100 KB, 350 KB,200 KB, 392 KB, 10 KB

d. 200 KB is put in 200-KB partition, leaving 300 KB, 100 KB, 350 KB, 0KB, 392 KB, 10 KB

e. 375 KB is put in 392-KB partition, leaving 300 KB, 100 KB, 350 KB, 0KB, 17 KB, 10 KB

**Worst Fit:**

a. 115 KB is put in 750-KB partition, leaving 300 KB, 600 KB, 350 KB, 200KB, 635 KB, 125 KB

b. 500 KB is put in 635-KB partition, leaving 300 KB, 600 KB, 350 KB,200 KB, 135 KB, 125 KB

c. 358 KB is put in 600-KB partition, leaving 300 KB, 242 KB, 350 KB,200 KB, 135 KB, 125 KB

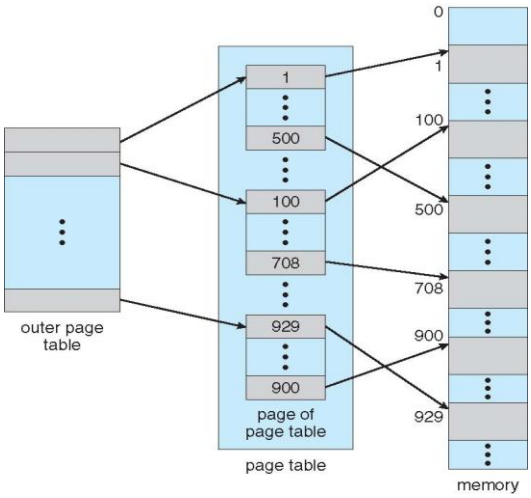
d. 200 KB is put in 350-KB partition, leaving 300 KB, 242 KB, 150 KB,200 KB, 135 KB, 125 KB

e. 375 KB must wait

**14a ii) Structure of the Page Table**

- Memory structures for paging can get huge using straight-forward methods
  - Consider a 32-bit logical address space as on modern computers
  - Page size of 4 KB ($2^{12}$)
  - Page table would have 1 million entries ($2^{32} / 2^{12}$)
  - If each entry is 4 bytes ➔ each process 4 MB of physical address space for the page table alone
    - Don't want to allocate that contiguously in main memory
  - One simple solution is to divide the page table into smaller units
    - Hierarchical Paging
    - Hashed Page Tables
    - Inverted Page Tables

**Hierarchical Page Tables:** Break up the logical address space into multiple page tables
A simple technique is a two-level page table




**Two-Level Paging Example**

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits
  - a page offset consisting of 12 bits

**Three-level Paging Scheme**

| outer page | inner page | offset |
|---|---|---|
| $p_1$ | $p_2$ | $d$ |
| 42 | 10 | 12 |

where $p_1$ is an index into the outer page table, and $p_2$ is the displacement within the page of the inner page table

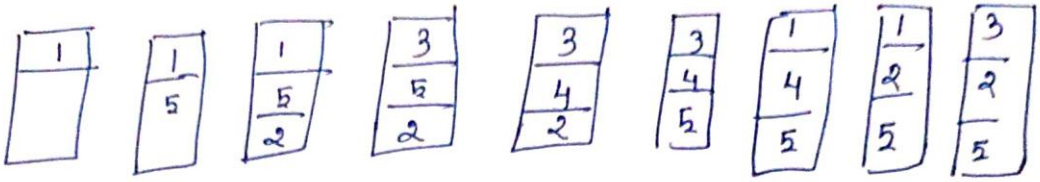
- Known as **forward-mapped page table**

| 2nd outer page | outer page | inner page | offset |
|---|---|---|---|
| $p_1$ | $p_2$ | $p_3$ | $d$ |
| 32 | 10 | 10 | 12 |

14 (b) i) FIFO page replacement

1, 5, 2, 3, 4, 5, 1, 2, 5, 1, 2, 3, 4, 5, 4

NPF NPF NPF NPF NPF



No of page faults = 10

Optimal page replacement

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

1 5 2 3 4 5 1 2 5 1 2 3 4 5 4

NPF NPF NPF NPF NPF NPF

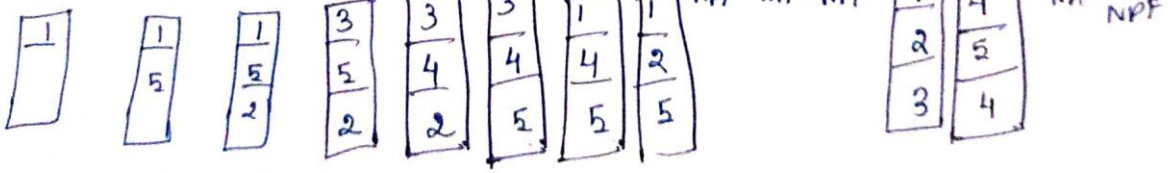
No of page fault = 8

LRU page replacement

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

1 5 2 3 4 5 1 2 5 1 2 3 4 5 4

NPF NPF NPF NPF NPF NPF



No of page fault = 10

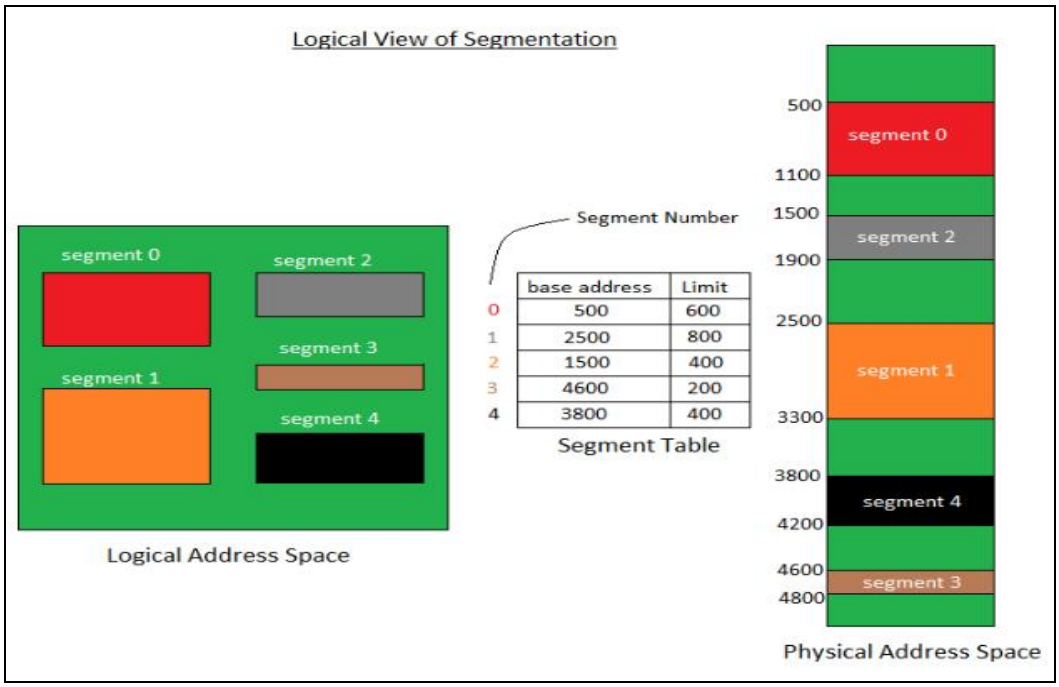
.

**2. Yes, this example exhibit Belady's anomaly.**

**14b ii) Segmentation Hardware**

Segmentation is **a memory management technique which divides the program from the user's view of memory**. That means the program is divided into modules/segments, unlike paging in which the program was divided into different pages, and those pages may or may not be loaded into the memory simultaneously

- **Segment base address**: Addresses specify both the name of the segment and the offset inside the segment. The user, thus, determines each address by two amounts: a segment name and an offset.
- **Segment Limit**: This indicates the length of the segment.




**15a)** Apply the FCFS, SSTF ,SCAN, C-SCAN, LOOK and C-LOOK disk scheduling algorithm for the disk queue with request for I/O blocks on cylinders as follows assuming disk head is at 53
Queue=98,183,37,122,14,124,65,67
FCFS: 640
SSTF: 236
SCAN:236(towards 0) or 331 (towards 199)
C-SCAN:382(towards 199) or 386 (towards 0)
LOOK: 208(towards 0)or 299(towards 199)
C-LOOK: 322(towards 199)or 326 (towards 0)
C-LOOK: 322(towards 199)or (towards 0) (OR)

15.b
**a. The block is added at the beginning.**
**contiguous allocation :** Each block must be shifted over to the next block. This involved one read and one write per block. Then the new block must be written. So, **201 I/O operations**.
**linked allocation:** Just write the new block making it point to the next block. Update the *first block* pointer in memory. So, only **1 I/O operation.**
**indexed allocation:** Just write the new block and update the *index* in memory. So, only **1 I/O operation.**

**b. The block is added to the middle.**
**contiguous allocation:** 50 blocks (the second half) must be shifted over one block and the new block must be written. As before, the shift takes one read operation and one write operation So, **101 I/O operations**.
**linked allocation:** Read 50 blocks to find the middle. Then , write new block somewhere with the *next block* pointing to the block after the 50th block. Then , write the 50th block to point to this new block. So, **52 I/O operations**.
**indexed allocation**: Just write the new block and update the *index* in memory. So, only **1 I/O operation.**

**c. The block is added at the end**
**contiguous allocation:** Just write the new block. So, only **1 I/O operation.**
**linked allocation: [**Assuming that in order to modify the a block's *next block* pointer, first read the whole block in, then write the whole block out just with that pointer changed]. First read the last block as given by the *last block* pointer, then write new block, then write the last block back modifying its *next block* pointer to point to new block, then update *last block* pointer in memory. So, **3 I/O operations.**
**indexed allocation**1. Just write the new block and update the *index* in memory. So, only **1 I/O operation.**

**d. The block is removed from the beginning.**
**contiguous allocation:** Just point the *first block* pointer to the second block. So, there is **'0' I/O operation.**
**linked allocation:** Read in the first block to get the second block's pointer and then set the *first block* pointer to point to the second block. So, only **1 I/O operation.**
**indexed allocation:** Just remove the block from the index in memory. So, there is **'0' I/O operation**

**e. The block is removed from the middle**
**contiguous allocation:** Assuming that removing the 51st block, then move 49 blocks. The 49 blocks takes a read and a write operation for each block. So, there is **98 I/O operations.**
**linked allocation:** It takes 51 reads to find the pointer to the 52nd block, then update the 50th block's *next block* pointer with this value. So, there is **52 I/O operations.**
**indexed allocation:** Just remove the block from the index in memory. So, there is **'0' I/O operation.**

**f. The block is removed from the end.**
**contiguous allocation:** Just update the *length* information stored in memory. So, there is **'0' I/O operation.**
**linked allocation:** update the *last block* pointer with the second to last block and the only way to get the second to last block is to read the preceding 99 blocks. Then probably mark the 99th block (the new last block) as having a null pointer and this would give the 100th operation. Save the new last block in *last block* pointer in memory. So, there is **100 I/O operations.**
**indexed allocation:** Just remove the block from the index in memory. So, there is **'0' I/O operation.**

The results are:

| | Contiguous | Linked | Indexed |
|---|---|---|---|
| a. | 201 | 1 | 1 |
| b. | 101 | 52 | 1 |
| c. | 1 | 3 | 1 |
| d. | 198 | 1 | 0 |
| e. | 98 | 52 | 0 |
| f. | 0 | 100 | 0 |