Solutions
Operating Systems (OS) UPC – 42344304
Sr.No. of Question Paper - 1703

1. (a) execution time binding
   (b)Short term scheduler or CPU scheduler
   (c) Bootstrap Program or Bootstrap loader
   (d) 7 (1 Child process is created by first fork(), 2 child processes are created by 2nd fork(), 4 child processes are created by 3rd fork() (1+2+4) **(1+1 for desc)**
   (e) Any two advantages of multiprocessor systems: **(1 + 1)**
   Increased Reliability:if one processor fails,the system will still work smoothly.
   Increased throughput: work done per unit time increases.
   Economy of Scale: Cost less than equivalent multiple single processor systems
   (f) (i) set value of timer- privileged **(1 + 1)**
   (ii) read the clock- non-privileged
   (g) page size=1KB=1024 Bytes **(1 + 1)**
   (i) 2378/1024=2.3, therefore address 2378 will be in page no. 3. And offset will be 330 (1024*2+330=2378).
   (ii) 9360/1024=9.14, therefore address 9360 will be in page no. 10 and offset will be 144 (1024*9+144=9360).
   (h) Command interpreter allows users to directly enter commands to be performed by the operating system. **(1 + 1)**
   Separating the command interpreter from the kernel provides flexibility, ease of replacements, ease of debugging and better security.
   (i) grep -c "hello" file1.txt **(2 + 2)**
   sort -r -k 2 file2.txt

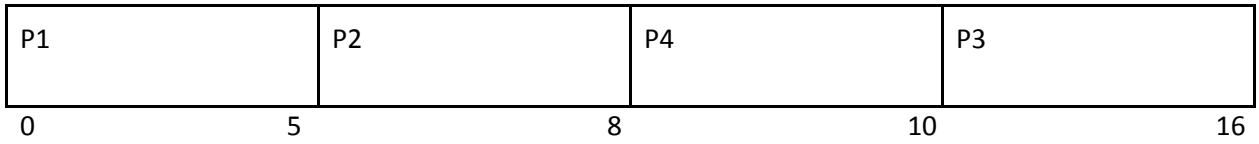
   (j) **(2)**
   (i)Multiprogramming increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute. The idea is to keep multiple jobs in main memory. If one job gets occupied with IO, CPU can be assigned to another job.
   Multi-tasking is a logical extension of multiprogramming. Multitasking is the ability of an OS to execute more than one task simultaneously on *a CPU machine.* These multiple tasks share common resources (like CPU and memory). In multi-tasking systems, the CPU executes multiple jobs by switching among them typically using a small time quantum (time sharing), and the switches occur so quickly that the users feel like interacting with each executing task at the same time.

   (ii)Renaming a file refers to changing the name of a file. **(2)**
   Copying a file refers to creating a duplicate copy of a file and its content.

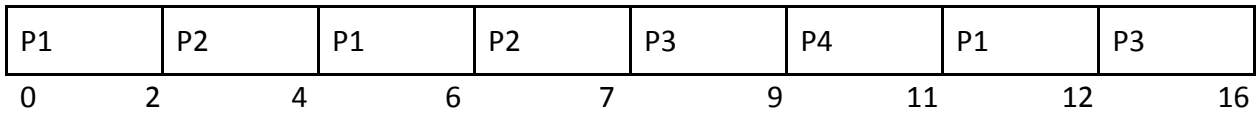
(k) **(1 X 4)**
   (i) Dispatch latency : the time it takes for the dispatcher to stop one process and start another running is known as dispatch latency.
   (ii) seek time: time necessary to move the disk arm to the desired cylinder
   (iii) Rotational latency : time necessary for the desired sector to rotate to the disk head.
   (iv) Response Time : Time from submission of a request until the first response is produced.

2.

SJF(non preemptive) Gantt Chart **(3)**



Round Robin (TQ=2) Gantt Chart **(3)**



For SJF(non preemptive) **(1 + 1)**

Wait time

P1=0
P2=3
P3=5
P4=2

Turn Around time

P1=5
P2=6
P3=11
P4=4

For Round Robin (Time Quantum=2) **(1 + 1)**

Wait time

P1=7
P2=2
P3=5
P4=3

Turn around time

P1=12
P2=5 P3=11 P4=5

3(a) (i) 200+200=400 nanoseconds **(2)**

(ii) 0.80(20+200) + 0.20(20+200+200) **(4)**

=0.80*220 + 0.20*420

= 176+84=260 nanoseconds

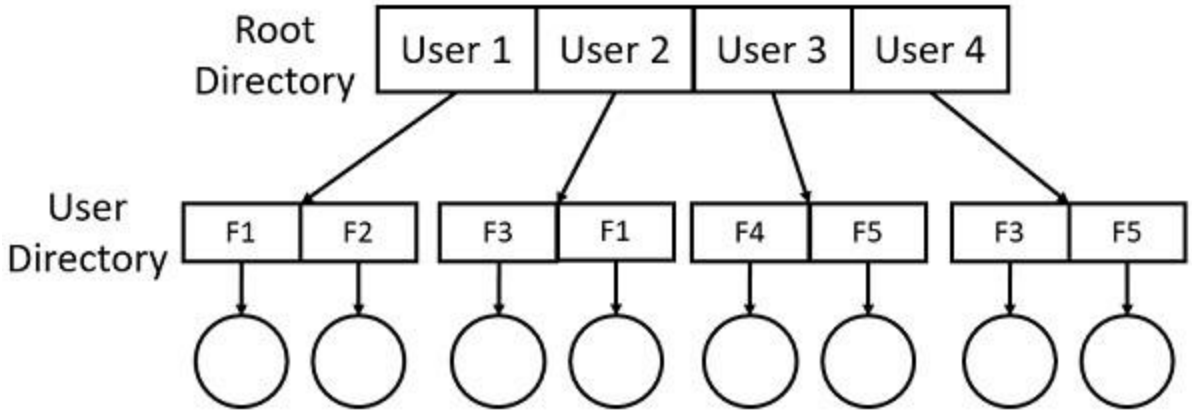
3(b) CPU-scheduling decisions may take place under the following four circumstances: **(1 X 4)**

- When a process switches from the running state to the waiting state (for example, as the result of an I/O request or an invocation of wait() for the termination of a child process)
- When a process switches from the running state to the ready state (for example, when an interrupt occurs)
- When a process switches from the waiting state to the ready state (for example, at completion of I/O)
- When a process terminates

4(a) An I/O-bound process is one that spends more of its time doing I/O than it spends doing computations. A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations. If all processes are I/O bound, the ready queue will almost always be empty, and the short-term scheduler will have little to do. If all processes are CPU bound, the I/O waiting queue will almost always be empty, devices will go unused, and again the system will be unbalanced. Thus it is important for OS to differentiate between CPU bound and IO bound processes. **(3)**

Long-term scheduler selects a good process mix of I/O-bound and CPU-bound processes. **(1)**

4(b)Two-level directory structure: In the two-level directory structure, each user has his own user file directory (UFD). The UFDs have similar structures, but each lists only the files of a single user. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. **(2 + 1 for diagram)**



Two-level Directory Structure

4 (c) (i) editor (ii) compiler (iii) loader **(1 X 3)**

5(a) Each process is represented in the operating system by a process control block (PCB)—also called a task control block. When a user requests to run a program, OS constructs PCB for that process. It stores resource management information, accounting information and execution snapshot. **(1)**

Information contained in PCB: **(3)**

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers

- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

5(b) (i) Process Management activities (any four): **(0.5 X 4)**

- Scheduling processes and threads on the CPUs
- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication

(ii) File Management activities(any four) **(0.5 X 4)**

- Creating and deleting files
- Creating and deleting directories to organize files
- Supporting primitives for manipulating files and directories
- Mapping files onto secondary storage
- Backing up files on stable (nonvolatile) storage media

5(c) Reasons (any two): **(1 X 2)**

- The child has exceeded its usage of some of the resources that it has been allocated.
- The task assigned to the child is no longer required.
- The parent is exiting, and the operating system does not allow a child to continue if its parent terminates.

6(a) **(4)**

```
read –p “Enter first number” n1
read –p “Enter second number” n2
if [ $n2 -lt $n1 ]
then
        m=$n2
        p=$n1
else
        m=$n1
        p=$n2
fi
r=`expr $p % $m`
while [ $r -ne 0 ]
do
    p=$m
    m=$r
    r = `expr $p % $m`
done
echo “GCD =” $m
```

6(b) **(0.5 +0.5 for example) X 3**

cat <*filename*>: Display the contents of a file
cat > <*filename*>: Create a file with given content
cat >> <*filename*>: Append the contents to a file

6(c) Main advantage of Demand Paging: **(1)**

With demand-paged virtual memory, pages are loaded only when they are demanded during program execution. Pages that are never accessed are thus never loaded into physical memory.

Computing Effective access time for a demand-paged memory: **(2)**

Let the memory-access time be denoted by *ma.* As long as we have no page faults, the effective access time is equal to the memory access time. If, however, a page fault occurs, we must first read the relevant page from disk and then access the desired word.

Let *p* be the probability of a page fault (*0 ≤ p ≤ 1*).

The effective access time is then computed as:

effective access time = (1 − p) × ma + p × page fault time

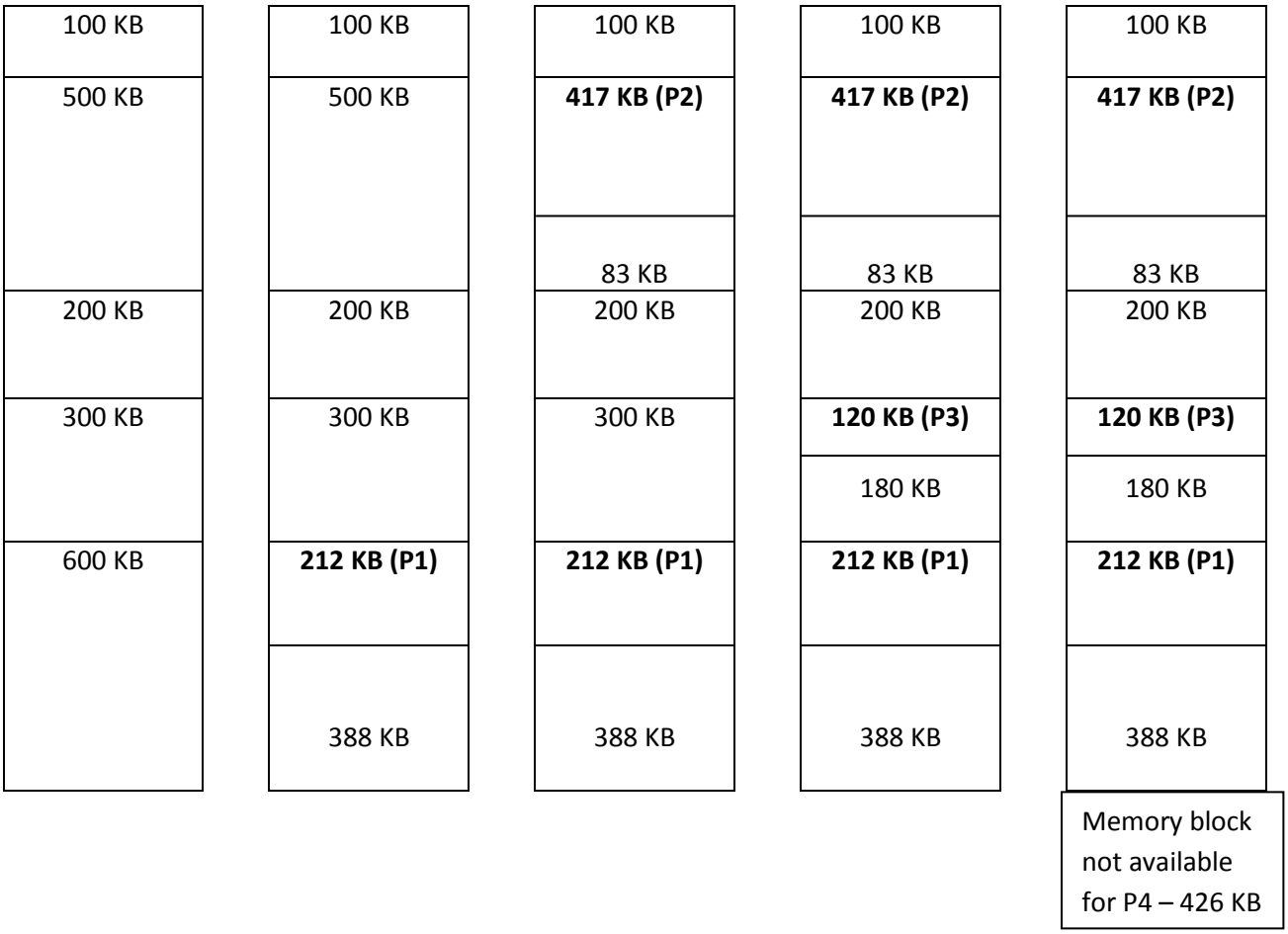
7(a)

100 KB	100 KB	100 KB	100 KB	100 KB
500 KB	500 KB	**417 KB (P2)**	**417 KB (P2)**	**417 KB (P2)**
		83 KB	83 KB	83 KB
200 KB	200 KB	200 KB	**120 KB (P3)**	**120 KB (P3)**
			80 KB	80 KB
300 KB	**212 KB (P1)**	**212 KB (P1)**	**212 KB (P1)**	**212 KB (P1)**
	88 KB	88 KB	88 KB	88 KB
600 KB	600 KB	600 KB	600 KB	**426 KB (P4)**
				174 KB

Best Fit: **(0.5 X 4 = 2)**

Internal Fragmentation: 100 +83 + 80 + 88 + 174 = 525 KB

Memory allocated to all processes.



Worst Fit: **(0.5 X 4 = 2)**

Internal Fragmentation: 100 +83 + 200 + 180 + 388 = 651 KB

Memory allocated to three processes, but memory block not available for fourth process.

Best Fit algorithm is better as memory allocated to all processes **(2)**

7(b) Two points of difference (**2 + 2)**

	External Fragmentation	Internal Fragmentation
1.	Small fragments (memory blocks) which remain unallocated	Unused memory inside a memory block allocated to a process
2.	Systems with variable-sized allocation units, such as the multiple-partition scheme and segmentation, suffer from external fragmentation.	Systems with fixed-sized allocation units, such as the single-partition scheme and paging, suffer from internal fragmentation

8(a) Size of logical address space = 8 X 1024=2^3 X 2^10=2^13 (**2)**

⇨ No. of bits in Logical address = 13

Size of physical address space = 32 X 1024=2^5 X 2^10=2^15 (**2)**

⇨ No. of bits in Logical address = 15

8(b) Advantages of Layered Approach over Simple Structure of OS: **(2)**

1. Simplicity of construction and debugging: Each layer uses services and functions of lower layers. Debugging can be done starting from the lowermost layer.
2. Information hiding: Each layer hides the existence of data structures and operations from higher level layers

Disadvantages of Layered Approach over Simple Structure of OS: **(2)**

1. Appropriate definition of various levels required in contrast to simple structure where all functions implemented in a monolithic structure
2. Less efficient than simple structure: Performance disadvantage as each layer adds to overhead for system calls

8(c) Swapping involves moving processes between main memory and a backing store. The backing store is commonly a fast disk. **(1)**

Mobile systems typically do not support swapping in any form: **(Any two)** **(2)**

i) Mobile devices generally use flash memory rather than more spacious hard disks as their persistent storage. The resulting space constraint is one reason why mobile operating-system designers avoid swapping.

ii) Limited number of writes can be tolerated by flash memory before it becomes unreliable

iii) Poor throughput between main memory and flash memory in these devices is another reason.