Operating System -

Name - Sradha Kedia
Roll no. - 20234757053

Ans 5. There are various solutions for critical section problem

i) Turn Variable / strict Alternation Method : It is a two process solution. Runs in user mode both the process share one variable - int turn

Algorithm :

```
while (true)
  {
    turn = i;
    while( turn == j);

    [Critical Section]

    turn = j;

    [remainder section]

  }
```

Eg :

```
Process P0
while(1)
{ while( turn != 0);
  { Critical section
  }
  turn = 1;
}
```

```
Process P1
while(1)
{ while( turn != 1);
  { critical section }
  turn = 0;
}
```

initially we can set turn = 0 or 1, but if turn = 0

then P_0 will execute first and if turn = 1 then P_1.

* Mutual Exclusion is achieved, but not progress.

ii) Shared Variables :

Algorithm - boolean flag[2];

- initially flag[0] = flag[1] = false
- flag[i] = true ⇒ P_i ready to enter its Critical section

Process P_0	Process P_1
while(1) {	while(1) {
flag[0] = true;	flag[1] = true;
while(flag[1]);	while(flag[0]);
Critical section	Critical section
flag[0] = false;	flag[1] = false;
remainder section	remainder section
}	}

* Mutual Exclusion is achieved, but not progress.

iii) Peterson Solution Method : It is also a two process solution.

it the two processes share two variable i) int turn
ii) boolean flag[2].

turn acts as same as first method, if turn is set to 0 then P_0 will execute first and if turn is set to 1 initially then

Date
Page

P_i executes first.

- flag array indicates if a process is ready to enter the CS (critical section).
  flag[i] = true ⇒ P_i is ready

Algorithm:

```
while(1){
    flag[i] = true;
    turn = j;
    while(flag[j] && turn == j);
    [Critical section]
    flag[i] = false;
    [remainder section]
}
```

eg:

Process P_0

```
while(1){
    flag[0] = true;
    turn = 1;
    while(flag[1]==1 &&
          turn==1);
    [critical section]
    flag[0] = false;
    [remainder section]
}
```

Process P_1

```
while(1){
    flag[1] = true;
    turn = 0;
    while(flag[0]==1 && turn==0);
    [critical section]
    flag[1] = false;
    [remainder section]
}
```

* Mutual Exclusion is satisfied, Progress is also satisfied & Bounding wait also.

Date
Page

iv) Semaphores : A method or tool to prevent race condition. It is an integer variable which is used in mutual exclusive manner by various concurrent cooperative process in order to achieve synchronization.

two types

- Counting ($-\infty$ to ∞)
- Binary (0, 1)

Two operations - 1) Wait / Down.
2) Signal / Up.

- wait(s) {
  while (s <= 0); // busy wait
  s--;
  }

signal(s) {
  s++;
  }

Busy wait :- the repeated execution of loop (infinitely).
- waiting to enter critical section.
- This continues to waste CPU cycles. CPU is not engaged in any real productive activity during this period.

Algorithm:
```
while (1) {
    wait(S);  ---> entry code
    [Critical section]
    signal(S); ---> exit code
    [Rimainder section]
}
```

entry and exit code is defined above.

eg: producer consumer solution

```
producer(item p)
{
    wait(empty);
    wait(s);
    Critical section
    increment 'in'
    signal(s);
    signal(full);
}
```

```
consumer()
{
    wait(full);
    wait(s);
    Critical section
    increment 'out'
    signal(s);
    signal(empty);
}
```

Ans 4: given logical address - 16 GB and 256 MB

we know, $1GB = 2^{30}$ and $1MB = 2^{20}$

$\therefore 16GB = 2^4 \times 2^{30} B$

$= 2^{34} B$

and $256MB = 2^8 \times 2^{20} B$

$= 2^{28} B$

To calculate page no. -

we use, page no. = $\frac{\text{logical Address or Process size}}{\text{Page size}}$

given, page size = 4 KB and 8 KB

$1KB = 2^{10} B$

$\therefore 4KB = 2^2 \times 2^{10} B = 2^{12} B$

also, $8KB = 2^3 \times 2^{10} B = 2^{13} B$

Date
Page

so, Displacement or offset of 4KB and 8KB pages are 12 & 13.

Now, page no. $= \frac{2^{34}B}{2^{12}B}$ for 4KB size pages

$= 2^{22}B$ or 4 Mb

page no. $= \frac{2^{34}}{2^{13}}B$ for 4KB size pages

$= 2^{21}B$ ~~or 4 MB~~.

for logical address = 256 mb

page number (4KB page size) $= \frac{2^{28}}{2^{12}} = 2^{16}B$

page number (8KB page size) $= \frac{2^{28}}{2^{13}} = 2^{15}B$

Ans 1: Process management — A running program needs to be able to halt its execution either normally (end) or abnormally (abort)

Types of process management system calls:

(1) fork() is a system call used for ~~terminate~~ the creating a new process called child process.
→ It takes no parameters and returns an integer value.

(2) exit() is a system call used to terminate the process normally.
→ It takes an integer i.e. status and return

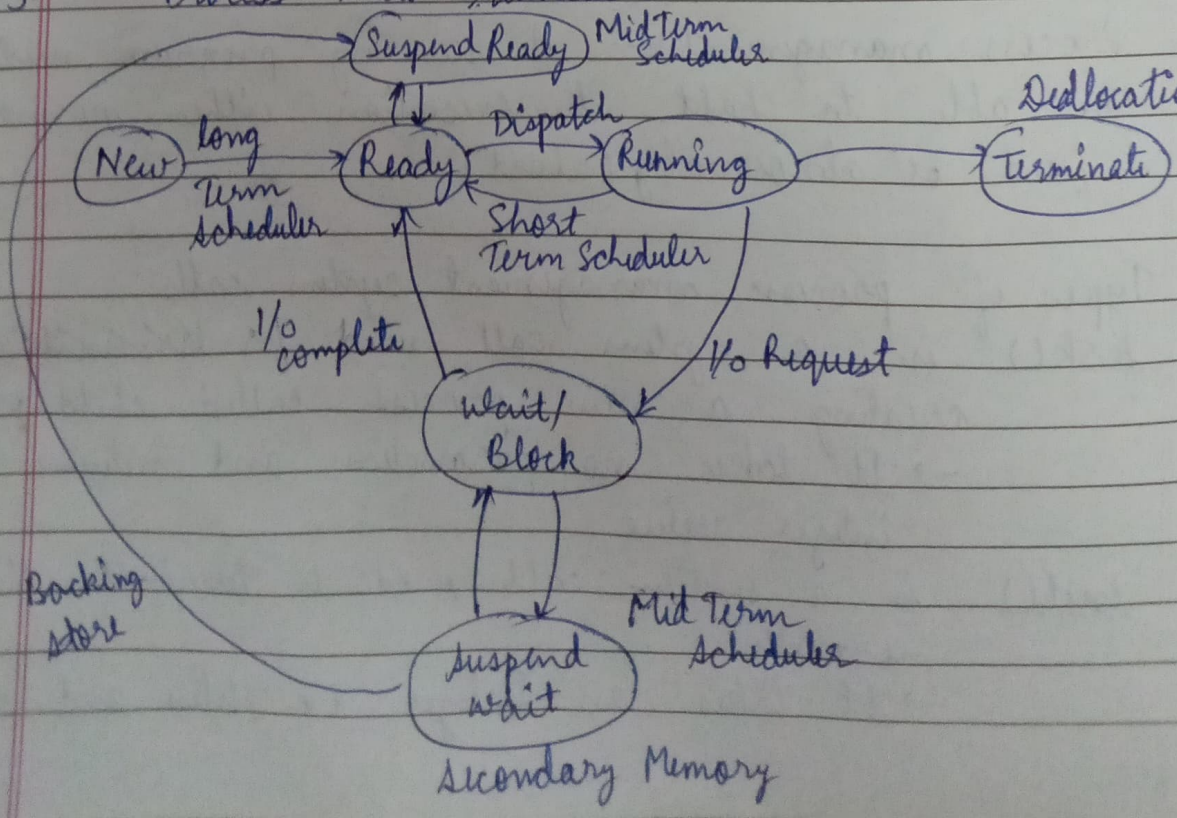
an integer exit status to the OS.

(3) abort(): This call terminates the process by raising a SIGABRT signal. it may not close files that are open.
→ It takes one argument status and returns a process ID of dead children.

→ It takes no argument and returns void.

(4) wait(): A call to wait() blocks the calling process until one of its child process exits or a signal is recieved.
→ It takes one argument status and returns a process ID of dead children.

Ans 3. Process State Transition



Date
Page

New - In this state, process is about to be created, but not the yet created, it is the program which is present in secondary memory that will be picked up by OS to creat process

Ready - here the process is ready and comes in ready queue inside the RAM. It waits for the CPU time for its execution.

Running - Here, Process is chosen by CPU for execution and the instructions within the process are executed by anyone of available CPUs.

Wait/Block - whenever the process requests access to I/O or needs input from the user, it enters the wait state.
once, I/O operation is completed, the process goes back to ready state.

Terminate - when the process execution is completed, process is killed as well as its process control block (PCB) is deleted.

Suspend Ready - Process that was initially in ready state but were swapped out of main memory and placed onto external storage by scheduler are said to be in suspend ready.

Suspend Wait - Similar to suspend ready but uses the

→ Functionality of OS.

ns. 2.

① Resource Management –
When we talk about single user working on a system then this functionality not come in much of the role but when multiple users work on a same system i.e. server level where multiple user requests & try to access ~~hard~~ resources then OS comes into role of when to give access & process request of a user & when to take back that access.

② Process Management –
When a user opens multiple processes i.e. browser also, File Explorer & TextEditor together then OS executes & manages all this processes by using CPU scheduling.
eg - there is a C file saved as .C now we need to execute this so OS executes this process in CPU by CPU scheduling (algorithms on how to ~~execute CPU~~ access the CPU and execute the process efficiently).

③ Storage ~~Resource~~ Management –
how the files are to be stored in hard disk is also done by OS.

④ Memory Management –
(RAM) all the tasks to be executed go in RAM first & after that ~~one~~ the task is given one by one to CPU. So, the allocation & deallocation of memory in RAM when tasks is to be done & then done respectively is managed by OS (This is important as RAM has

limited storage).

(5) Security & Privacy - When system is on user need to authenticate to use it. Or if there are multiple process & each are given indexes so no process can interfere data of other process. All this provided by OS.