1(i) FCFS

Turn around Time (TAT) = Completion Time – Arrival Time

Gantt Chart on next page

Process	Ar. time	Ex Time	Completion Time	TAT
P1	0	4	4	4
P2	2	7	11	9
P3	3	3	14	11
P4	7	6	20	13

Average process TAT = (4+9+11=13)/4 = 9.25 seconds

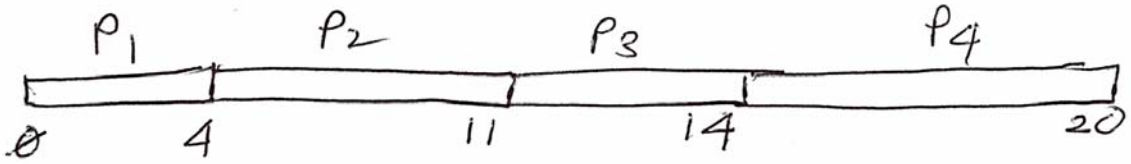
(ii) Name of Scheduling Policy is: Shortest Job First

(SJF) Gantt Chart on next page

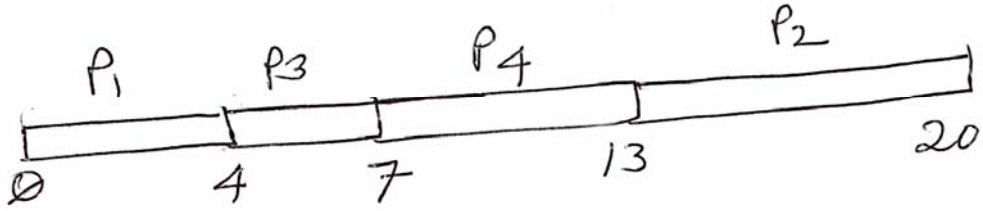
Process	Ar. time	Ex Time	Completion Time	TAT
P1	0	4	4	4
P2	2	7	20	18
P3	3	3	7	4
P4	7	6	13	6

Average process TAT = (4+18+4+6)/4 = 8 seconds

(i) FCFS



(ii) SJF



[15 marks] **2.** Fill up **the blank** (gap) in each of the following sentences with the appropriate word/phrase (selected from the set provided below the sentence). That is, for each sentence (i) – (x) select the best possible choice (from a, b, c etc.) and insert it (a/b/c….) above the line (__).

[NOTE: NO justification needs to be provided for answers to this question]

(i) When a critical section is used for protecting shared variables, the variable can be accessed simultaneously by ____(a)_____

(a) multiple reader processes (b) multiple writer processes (c) both reader and writer processes (d) only one process.

(ii) If a process performs a wait operation on variable x then the process is blocked unconditionally when x is a _____(c)_____

(a) semaphore (b) a simple binary variable (c) a condition variable (d) an integer variable

(iii) A process waiting to enter the critical section for a long time when other processes are going in and out of the critical section frequently is the indication of ____(d)_____

(a) a deadlock (b) a livelock (c) a violation of the mutual exclusion requirement (d) a violation of the bounded waiting requirement

(iv) The term _____(c)_____ refers to the continuous looping of a process executing an instruction or a set of instructions that tests the appropriate variable to gain entrance to a critical section.

(a) direct message passing (b) indirect message passing (c) busy waiting (d) racing

(v) Monitor is a language construct used to _____(c)_____

(a) observe the progress of a process execution (b) aid the programmer to display the program output (c) control access to the critical section (d) achieve spinlocks

(vi) A system in which the output must be generated before the expiry of a deadline is called a _____(b)_____

(a) soft real time system (b) a hard real time system (c) a time sharing system (d) a multiprogrammed system

(vii) The CPU scheduling algorithm that is expected to provide the smallest mean turnaround time for jobs is _____(b)_____

(a) Round Robin (b) Shortest Remaining Processing Time First (c) Multi Level Feedback Queue (d) Largest Processing Time First

(viii) The "running" state of a process can be reached directly from the ___(c)_____ state

(a) new (b) waiting (c) ready (d) terminated

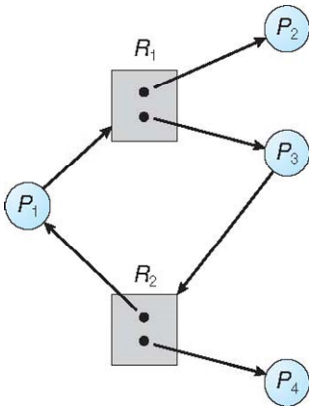
(ix) Indirect message passing among processes requires a _____(b)_____

(a) shared variable (b) mailbox (c) semaphore (d) monitor

(x) In a multi-threaded process any of the threads making a blocking system call blocks the entire process when the _____(a)_____ is used to describe the relationship between user threads and kernel threads.

(a) the many-to-one model (b) the one-to-one model (c) the many-to-many model

[5 marks] **3.** Consider the following resource allocation graph. Is there a deadlock on the system?
[Justify your answer]



Ans: There is no dadlock
Justification: Even though there is a cycle in the resource allocation graph there is no deadlock. The rationale is provided below

- When process P4 releases an instance of R2 it can be allocated to P3 that can now complete (because it will have both resources it needs).
- When P3 completes it will release an instance of R1 which can the be allocated to Process P1 that can now complete (because it will have both resources it needs).
- So all processes will complete – therfore no deadlock

Note that a similar set of arguments can be made based on P2 relaesing an instance of resource R1 and so on.

[5 marks] **4.** Consider a process P1 executing on a Linux environment executing a fork() system call for creating a child process P2. What are the values returned to P1 and P2 after the system call is made? How is the difference in these values utilized by the program?

i. Value returned to P1: PID of the new (child) process created
ii. Value returned to P2 is 0
iii. The returned value is checked by each process. The difference in value is used to make P1 and P2 execute different sets of instructions after the return from the fork system call.

[8 marks] **5.** In this problem you will be concerned with synchronization of processes with the help of a **monitor** called synchronize. Consider a system in which 20 processes P1 .... P20 run concurrently on the system. Each process performs some computation. Before exiting the system each process must however make sure that all the other processes have also completed their computations. The typical operations performed by a process are outlined below.

Process Pi {

1. Perform computation.
2. Call function "done" in the monitor synchronize.
   {A return from the monitor must imply that all other processes have completed their computations and Pi can now exit.}
3. exit

} // end of Pi

An unfinished structure of the monitor "synchronize" is provided.

monitor synchronize {

variable declarations

function done ....

.....

.....

{Initialization Code}

}

Your Job: Write the pseudo-code (algorithm) for the monitor for solving the synchronization problem described above. You only need to complete the variable declaration part, write the pseudo code for ``done`` and the initialization code. Note that you do not need any other functions and your variable declaration CAN NOT include a variable of type semaphore.

GO TO NEXT PAGE

Ques 5. Ans:

```
Monitor synchronize {

        int noDone; // variable that keeps track of number of processes completed
        condition queue; // condition variable used to make processes wait

    procedure done () {

        noDone++; // increment the number of processes completed
        if (noDone < 20) {
            queue.wait (); // Wait until all processes complete
        }
        queue.signal(); // Wake up another waiting process
    }

     {  //initialization code
        noDone = 0;
      }
}
```