Question 1

The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: the higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

$$\text{Priority} = (\text{recent CPU usage} / 2 + \text{base})$$

where base = 60 and recent CPU usage refers to a value indicating how often a process has used the CPU since priorities were last recalculated.

Assume that recent CPU usage for process P1 is 40, for process P2 is 18, and for process P3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

Solution

New priority for P1 = 40/2+60 = 80
New priority for P2 = 18/2+60 = 69
New priority for P3 = 10/2+60 = 65

A CPU bound process is likely to have high CPU utilization. Thus the recent CPU usage would be high. Thus the priority number will be larger for CPU-bound processes. Thus the traditional UNIX scheduler will lower the relative priority of a CPU-bound process.

Question 2

What would be the output from the program at LINE C and LINE P?

Listing 1: C program for Exercise 4.19

```
1 #include <pthread.h>
2 # include <stdio.h>
3 int value = 0 ;
4 void * runner ( void * param ) ; / * the thread * /
5 int main ( int argc , char * argv [ ] )
6 {
7 pid_t pid ;
8 pthread_t tid ;
9 pthread _ attr _ t attr;
10 pid = fork( ) ;
11 if ( pid == 0 ) { /* child process * /
12 pthread _ attr _ init (& a t t r ) ;
13 pthread _ create (&tid ,&attr , runner , NULL ) ;
14 pthread _ join ( tid, NULL ) ;
15 printf ( ”CHILD: value = %d ” , value ) ; / * LINE C * /
16 }
17 else if ( pid > 0 ) { / * parent process * /
18 wait (NULL ) ;
19 printf ( ”PARENT: value = %d ” , value ) ; / * LINE P * /
20 }
21 }
22 void * runner ( void * param ) {
23 value = 5 ;
24 pthread_exit ( 0 ) ;
25 }
```

Solution

When we fork, we are creating a new process. Let's call that the child process. In that process, pid is 0. The variable “value” is a global variable in the child process as well. Child process has its own set of variables that it has copied from the parent process. Thus during process creation, there are two copies of the global variable “value”, one in the child process and one in the parent process.

Then in the child process, we are modifying the variable “value” by creating a thread. Since “value” is a global variable, the thread will modify the global variable. Thus the output at LINE C will be:

CHILD: value = 5

The copy of the global variable “value” in the parent process does not get modified. Thus the output at LINE P will be:

PARENT: value = 0

Question 3

Consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
P1	5	4
P2	3	1
P3	1	2
P4	7	2
P5	4	3

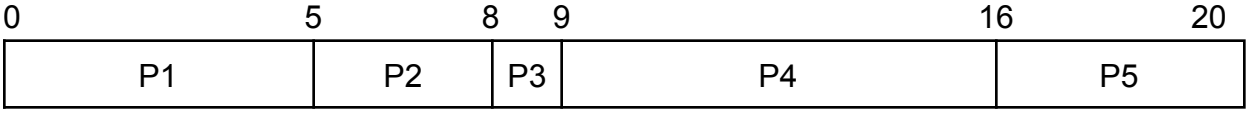
The processes are assumed to have arrived in the order P1, P2, P3, P4, P5 all at time 0.

A. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, non-preemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).
B. What is the turnaround time of each process for each of the scheduling algorithms in part a?
C. What is the waiting time of each process for each of these scheduling algorithms?
D. Which of the algorithms results in the minimum average waiting time (over all processes)?

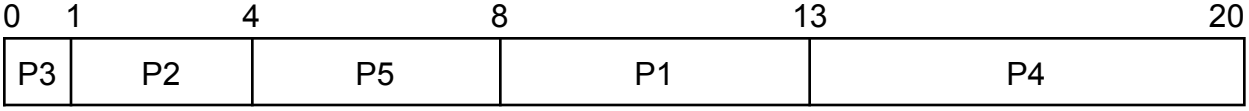
Solution

Part A

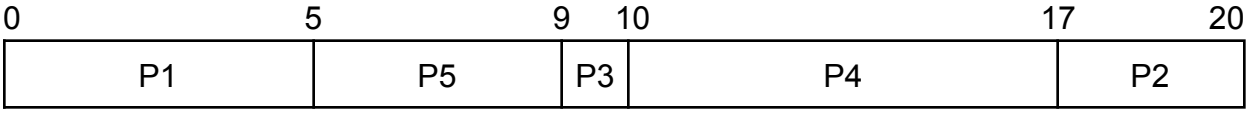
FCFS:



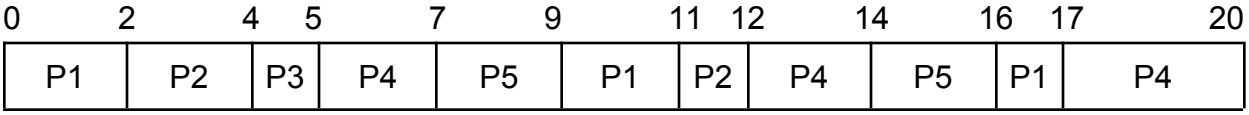
SJF:



Non-preemptive priority scheduling:



RR:



Part B

Process	FCFS	SJF	Priority	RR
P1	5	13	5	17
P2	8	4	20	12
P3	9	1	10	5
P4	16	20	17	20
P5	20	8	9	16

Part C

We can write waiting time as Turnaround Time - Burst Time. Therefore we will get the following table for waiting times of each process

Process	FCFS	SJF	Priority	RR
P1	0	8	0	12
P2	3	1	17	9
P3	8	0	9	4
P4	9	13	10	13

P5	16	4	5	12

Therefore the average waiting times are given by

Algorithm	Average Waiting Time
FCFS	7.2
SJF	5.2
Priority Scheduling	8.2
RR	10

Part D

Short Job first algorithm results in the minimum average waiting time.

Question 4

Consider a preemptive priority scheduling algorithm based on dynamically changing priorities. Larger priority numbers imply higher priority. When a process is waiting for the CPU (in the ready queue, but not running), its priority changes at a rate α. When it is running, its priority changes at a rate β. All processes are given a priority of 0 when they enter the ready queue. The parameters α and β can be set to give many different scheduling algorithms.

A. What is the algorithm that results from $\beta > \alpha > 0$?
B. What is the algorithm that results from $\alpha < \beta < 0$?

Solution

Part A

The algorithm that results because of $\beta > \alpha > 0$ is First Come First Served (**FCFS**). This is because since $\beta > \alpha$, the priority of the running process will keep increasing faster than that of processes waiting for the CPU. Since initial priority is 0, the running process will always have a higher priority. Thus there will be no preemption. In the ready queue, the priority of a process that comes first will always be higher than that of a process that comes later (because all processes in the ready queue have priorities increasing at rate α). Thus the process that comes first will be served first. Thus we have the FCFS algorithm.

Part B

The algorithm that results because of $\alpha < \beta < 0$ is Last In First Out (**LIFO**). This is because since $\beta > \alpha$, the priority of the processes waiting for the CPU will keep decreasing faster than that of the running process. Since initial priority is 0, the running process will always have a priority lower than a new process. Thus the new process will preempt the running process. In the ready queue, the priority of a process that comes first will always be lower than that of a process that comes later (because all processes in the ready queue have priorities decreasing at rate α and the process coming to the ready queue will come only when it was first running and then got preempted). Thus the process that comes last will be served first. Thus we have the LIFO algorithm.

Question 5

Consider two processes, P1 and P2 , where p1 = 50, t1 = 25, p2 = 75, and t2 = 30.

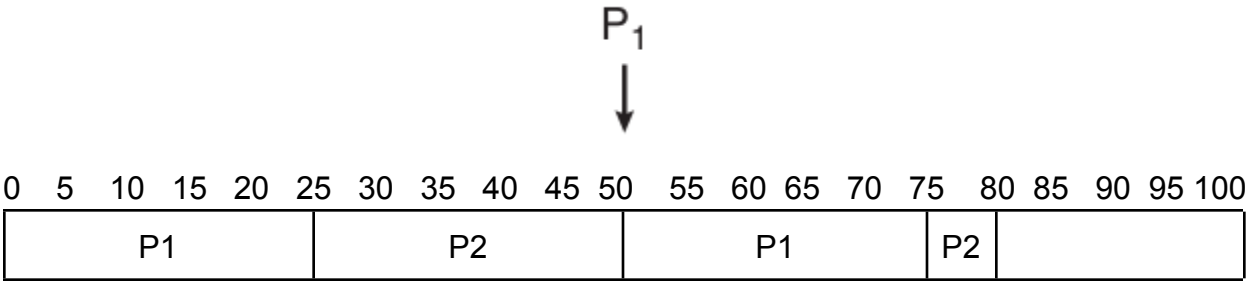
A. Can these two processes be scheduled using rate-monotonic scheduling? Illustrate your answer using a Gantt chart such as the ones in Figure 5.21–Figure 5.24.
B. Illustrate the scheduling of these two processes using earliest-deadline-first (EDF) scheduling.

Solution

Part A

No, these two processes cannot be scheduled using rate-monotonic scheduling. This is because total CPU utilization for these processes is t1/p1+t2/p2 = 90% but the upper bound on CPU utilization is 83% for rate-monotonic scheduling for two processes.

The same can be seen from the following Gantt chart



P2 is missing its deadline of 75 ms because P1 is preempting the execution of P2 because P1 has a higher priority.

Part B