1. Consider the following function:

```
void f ( int n )
{
   if (n <= 0) return;
   while (n--) {
      printf("PID = %d, PPID = %d, n = %d\n", getpid(), getppid(), n);
      fflush(stdout);
      fork();
   }
}
```

How many lines (as a function of n) will be printed by f(n)? What is the potential problem if you do not have the `fflush` statement?

Solution

The number $T(n)$ of lines printed satisfies the following recurrence relation.

$T(1) = 1$
$T(n) = 1 + 2\ T(n-1)$ for $n \geq 2$

The closed-form solution for this recurrence is

$T(n) = 2^n - 1$ for all $n \geq 1$.

Child inherits a copy of the parent's `stdout` buffer. The buffer should be empty when the child starts running. Otherwise some prints pending for parent will also be pending for child.

2. Consider the following recursive function.

```
void f ( int n )
{
   if (n <= 0) return;
   while (n--) {
      printf("PID = %d, PPID = %d, n = %d\n", getpid(), getppid(), n);
      fflush(stdout);
      if (!fork()) f(n);
   }
}
```

a) Explain why this function prints more lines (in general) than the function of Exercise 1.

Solution

The child prints all the lines as in Exercise 1. It prints additional lines by the recursive call f(n).

b) Prove or disprove: The number $T(n)$ of lines printed by this function satisfies the following recurrence relation.

$$T(1) = 1$$
$$T(n) = 3\ T(n-1) + 1 \quad \text{for} \quad n \geq 2$$

Solution

False. By the above recurrence, $T(2) = 4$. But the actual number of lines printed by f(2) is 5. Visualize why.

3. [*Convoy effect*]

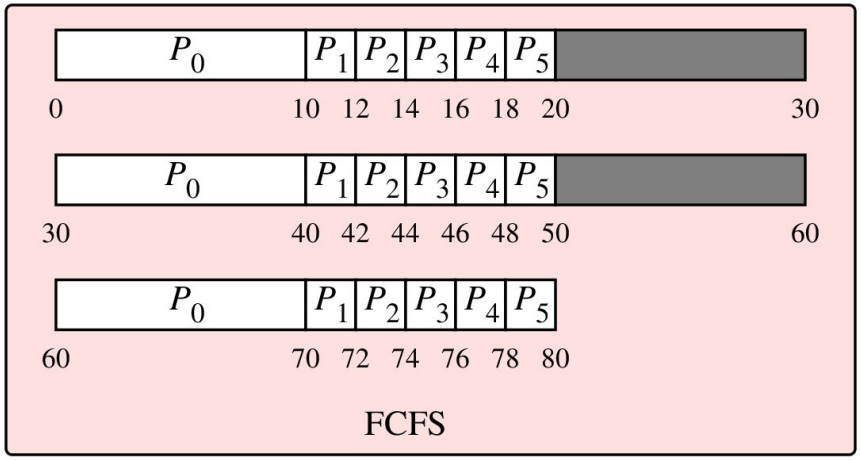
There are six processes all available at time $t = 0$. P_0 is a CPU-bound process with each CPU burst taking 10 ms, whereas P_1, P_2, P_3, P_4, and P_5 are IO-bound processes with each CPU burst taking 2 ms. Assume that each IO burst takes 20 ms time. Each of these six processes do three CPU bursts and two interleaving IO bursts. Draw the Gantt charts for the following scheduling algorithms. From the Gantt chart, deduce the average wait times of these processes.

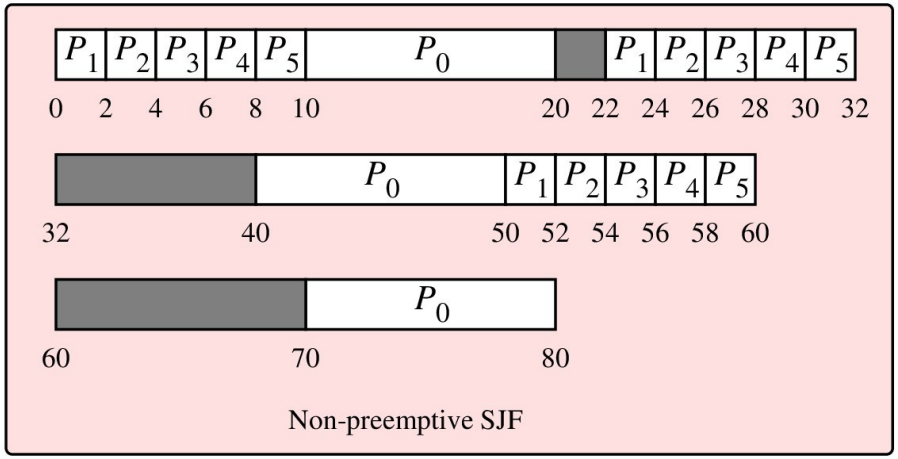
(a) FCFS
(b) Non-preemptive SJF
(c) Preemptive SJF
(d) Round Robin with quantum $q = 5$

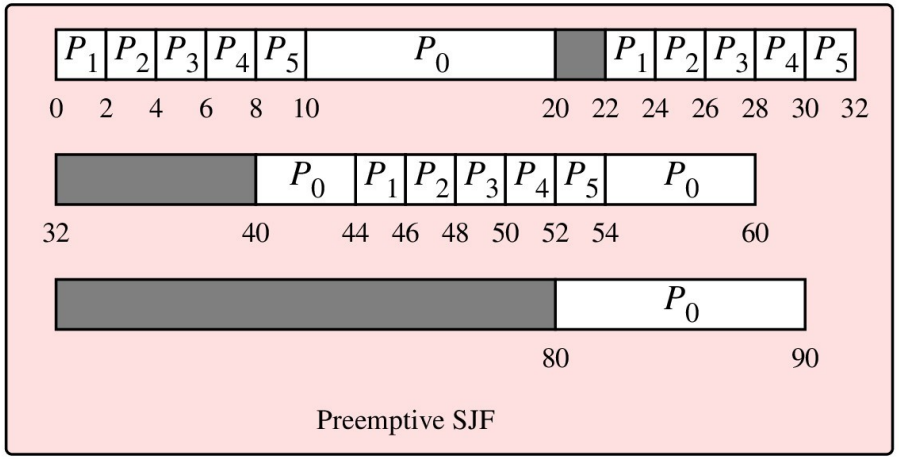
Solution

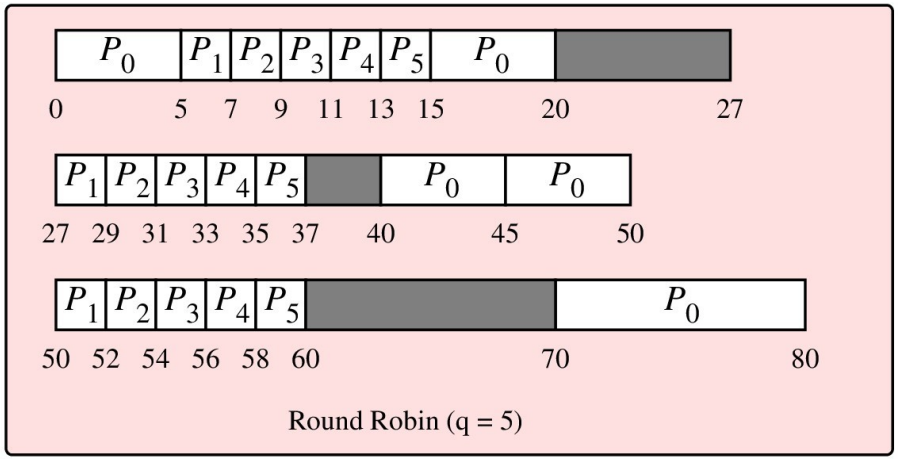
The Gantt charts are on the next page. Total time needed for P_0 is $10 \times 3 + 20 \times 2 = 70$. For the other processes, this time is $2 \times 3 + 20 \times 2 = 46$. From this the average wait times can be caluclated.

(a) FCFS: $[\,(70 - 70) + (72 - 46) + (74 - 46) + (76 - 46) + (78 - 46) + (80 - 46)\,] / 5 = 30$
(b) Non-preemptive SJF: $[\,(80 - 70) + (52 - 46) + (54 - 46) + (56 - 46) + (58 - 46) + (60 - 46)\,] / 5 = 12$
(c) Preemptive SJF: $[\,(90 - 70) + (46 - 46) + (48 - 46) + (50 - 46) + (52 - 46) + (54 - 46)\,] / 5 = 8$
(d) RR: $[\,(80 - 70) + (52 - 46) + (54 - 46) + (56 - 46) + (58 - 46) + (60 - 46)\,] / 5 = 12$


P_0 P_1 P_2 P_3 P_4 P_5
0 10 12 14 16 18 20 30
P_0 P_1 P_2 P_3 P_4 P_5
30 40 42 44 46 48 50 60
P_0 P_1 P_2 P_3 P_4 P_5
60 70 72 74 76 78 80
FCFS


P_1 P_2 P_3 P_4 P_5 P_0 P_1 P_2 P_3 P_4 P_5
0 2 4 6 8 10 20 22 24 26 28 30 32
P_0 P_1 P_2 P_3 P_4 P_5
32 40 50 52 54 56 58 60
P_0
60 70 80
Non-preemptive SJF


P_1 P_2 P_3 P_4 P_5 P_0 P_1 P_2 P_3 P_4 P_5
0 2 4 6 8 10 20 22 24 26 28 30 32
P_0 P_1 P_2 P_3 P_4 P_5 P_0
32 40 44 46 48 50 52 54 60
P_0
80 90
Preemptive SJF


P_0 P_1 P_2 P_3 P_4 P_5 P_0
0 5 7 9 11 13 15 20 27
P_1 P_2 P_3 P_4 P_5 P_0 P_0
27 29 31 33 35 37 40 45 50
P_1 P_2 P_3 P_4 P_5 P_0
50 52 54 56 58 60 70 80
Round Robin (q = 5)

4. Consider ***three-level feedback queue scheduling***. The three queues Q1, Q2, and Q3 support the following scheduling algorithms: RR(q = 5), RR(q = 10), and preemptive FCFS. Each process at arrival or after IO completion joins Q1. If it cannot finish in its next time quantum, it is enqueued at the back of Q2. When the process is scheduled from Q2, the process runs for 10 ms, and if it is still unable to complete its CPU burst, it joins Q3, and can be scheduled only when Q1 and Q2 are empty. Q3 scheduling is preemptive, that is, if a process in Q3 is being executed and a new process arrives, the running process is preempted and goes back to Q3. Assume that there is no aging policy in the scheduling algorithm.

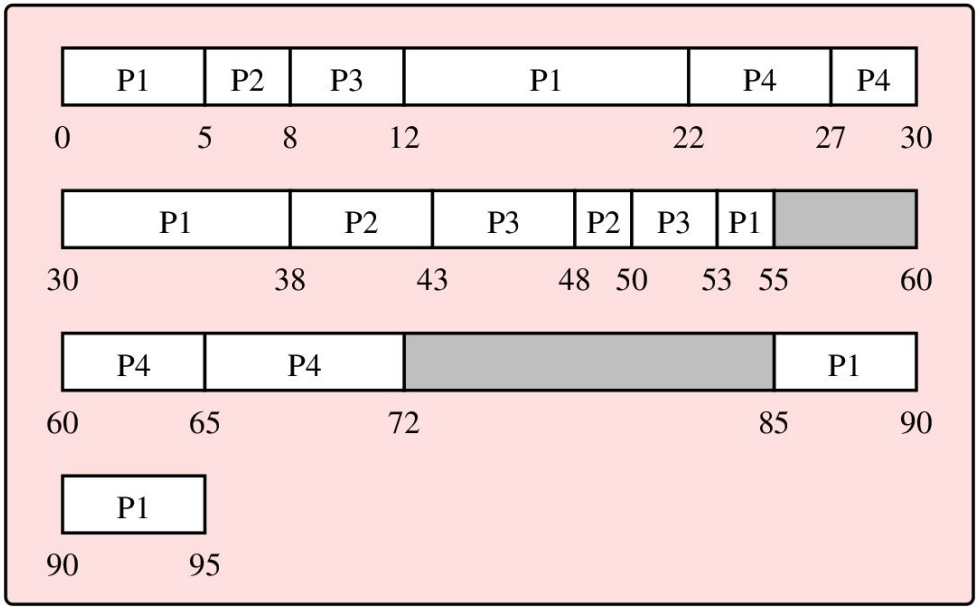
Four processes arrive at the system, each with two CPU bursts and one intermediate IO burst. The burst and arrival times of the processes are given in the table below. All times are in ms.

Process	CPU Burst 1	IO Burst	CPU Burst 2	Arrival Time
P1	25	30	10	0
P2	3	30	7	2
P3	4	30	8	7
P4	8	30	12	15

Draw the Gantt chart with detailed explanation how the three queues are populated at different times. Also compute the turnaround times and wait times of the four processes.

Solution

The Gantt chart is given on the next page (without explanations).



Process	Running Time	Turnaround time	Wait time
P1	25 + 30 + 10 = 65	95 – 0 = 95	95 – 65 = 30
P2	3 + 30 + 7 = 40	50 – 2 = 48	48 – 40 = 8
P3	4 + 30 + 8 = 42	53 – 7 = 46	46 – 42 = 4
P4	8 + 30 + 12 = 50	72 – 15 = 57	57 – 50 = 7

5. [*Multi-core scheduling*]

Solve Exercise 1 of previous year's MidSem test.

Note: The solution posted in the website is not correct.

6. Consider the following variant of Peterson's algorithm. The algorithm is given for Process P_i. The other process is called P_j.

```
flag[i] = true;
while ( (flag[j]) && (turn == j) ) ;

CRITICAL SECTION

flag[i] = false;
turn = j;

REMAINDER SECTION
```

Prove/Disprove: This variant will ensure mutual exclusion.

Solution

False. Assume that both the processes run the above code in a loop. Then so far as the synchronization statements are considered, `turn` is set to `j` before `flag[i]` is set to `true`. So the effect will be similar to a swap of these two instructions in the original variant of Peterson's algorithm, implying that both the processes may enter their respective critical sections concurrently.