長庚大學102學年度第一學期作業系統第二次期中測驗(總分106)

系級: 姓名: 學號:

1. (8%) Why should a web server not run as a single-threaded process?

Answer: For a web server that runs as a single-threaded process, only one client can be serviced at a time. This could result in potentially enormous wait times for a busy server.

2. (8%) Describe the difference between the **fork()** and **clone()** Linux system calls.

Answer: The **fork()** system call is used to duplicate a process. The **clone()** system call behaves similarly except that, instead of creating a copy of the process, it creates a separate process that might shares the address space of the calling process and might share some data structure by setting flags CLONE_FS, CLONE_VM, CLONE_SIGHAND, CLONE_FILES.

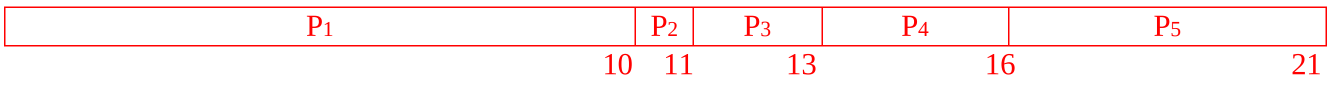
3. (18%) Consider the following processes, assume that the time unit is one millisecond, please (1) draw the scheduling charts for FCFS, SJF and RR (time slice=1), (2) derive the waiting time of each process, and (3) derive the average waiting time of each scheduling algorithm.

Process	Burst Time
P_1	10
P_2	1
P_3	2
P_4	3
P_5	5

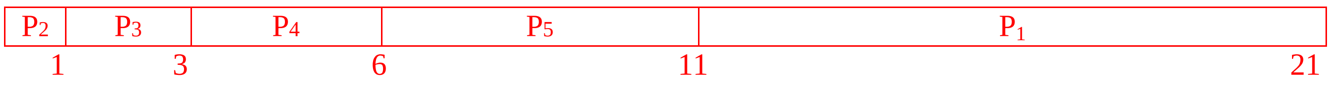
Answer:

(1)

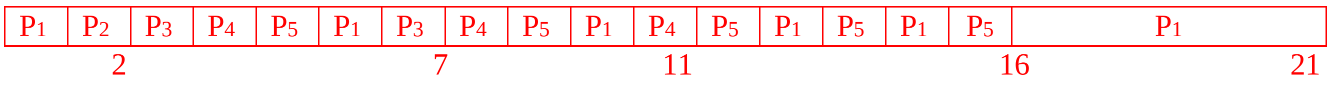
FCFS:



SJF:



RR



(2)

FCFS:	P_1: 10-10=0,	P_2: 11-1=10,	P_3: 13-2=11,	P_4: 16-3=13,	P_5: 21-5=16
SJF:	P_1: 21-10=11,	P_2: 1-1=0,	P_3: 3-2=1,	P_4: 6-3=3,	P_5: 11-5=6
RR:	P_1: 21-10=11,	P_2: 2-1=1,	P_3: 7-2=5,	P_4: 11-3=8,	P_5: 16-5=11

(3)

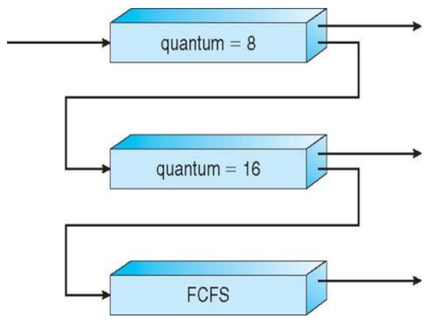
FCFS: (0+10+11+13+16)/5 = 10ms SJF: (11+0+1+3+6)/5 = 4.2ms RR: (11+1+5+8+11)/5 = 7.2ms

4. (12%) Please (1) define what is the “multilevel feedback queue scheduling,” *(hint: you can draw a diagram to better illustrate your ideas)* and (2) explain the advantage of the multilevel feedback queue scheduling.

Answer:

(1) A process can move between the various queues. For example, as shown in the figure, each task first goes to the top queue. If the process is not completed within time quantum 8, it is moved to the second queue. If the process is not completed with the further time quantum 16, it is moved to the FCFS queue.

(2) The scheduling is flexible. In the example, short CPU bursts should be completed in the first and second queues, and long CPU bursts are then handled by FCFS.



5. (8%) Explain the fundamental difference between asymmetric and symmetric multiprocessing.

Answer: In asymmetric multiprocessing, all scheduling decisions, I/O, and other system activities are handled by a single processor, whereas in SMP, each processor is self-scheduling.

6. (8%) What is the difference between hard real-time systems and soft real-time systems?

Answer:

- **Soft real-time systems** – do not guarantee when critical real-time process will be scheduled
- **Hard real-time systems** – always service each task before its deadline.

7. (12%)Please (1) define “Race Condition” and (2) provide an example for Race Condition.

Answer:

(1) A situation where the outcome of the execution depends on the particular order of process scheduling.

(2)

- One counter++ and one counter--

r1 = counter	r2 = counter
r1 = r1 + 1	r2 = r2 - 1
counter = r1	counter = r2

- Initially, let counter = 5
  1. P: r1 = counter
  2. P: r1 = r1 + 1
  3. C: r2 = counter
  4. C: r2 = r2 – 1 ➡ A Race Condition!
  5. P: counter = r1
  6. C: counter = r2 = 4
- The result can be 4, 5 or 6

8. (12%) What are the three fundamental requirements for solving critical-section problem? Please also briefly explain the meaning of each requirement.

Answer:

- **Mutual Exclusion** – Only one process can be in its critical section.
- **Progress** – If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
- **3. Bounded Waiting** – A waiting process only waits for a bounded number of processes to enter its critical section.

9. (20%) For the read-write problem, we have to protect the shared data. We allow multiple readers to read the data at the same time, or only single writer can access the data. Let's have the following semaphores and read counter:

- Semaphore **rw_mutex** initialized to 1
- Semaphore **mutex** initialized to 1
- Integer **read_count** initialized to 0

We now also have the writer as follows:

Writer:

```
wait(rw_mutex);
......
Writing is performed here;
......
signal(rw_mutex);
```

The reader is as follows:

Reader:

```
wait(mutex);
read_count++; //Increase the number of the counter of readers
if (read_count == 1) //If this is the first reader
    wait(rw_mutex); //Have to wait if some writer is working
signal(mutex);
......
Reading is performed here;
......
wait(mutex);
```

Answer:

```
read_count-- ;
if (read_count == 0)
    signal(rw_mutex);
signal(mutex);
```

Now, your mission is to ***fill the remaining part*** of the Reader to solve the read-write problem.