Solution to Selected Additional Problems related to Multi-tasking and Real-time Scheduling

Q. 1: What is the difference between turnaround time and response time.

Ans. 1: Turnaround time is the total time that a request spends in the system (waiting time plus service time. Response time is the elapsed time between the submission of a request until the response begins to appear as output.

Q. 2: What is the difference between Non-preemptive and Preemptive scheduling.

Ans. 2: Non-preemptive: A process is in the Running state, it continues to execute until (a) it terminates or (b) blocks itself to wait for I/O or to request some operating system service. Preemptive: The currently running process may be interrupted and moved to the Ready state by the operating system. The decision to preempt may be performed when a new process arrives, when an interrupt occurs that places a blocked process in the Ready state, or periodically based on a clock interrupt.

Q. 3: Explain how a preemptive priority scheduling system would work.

Ans. 3: In such a system, the highest priority process that is ready is run is always the one that is currently running. If a process becomes ready and has higher priority than the process currently running, then the current process is preempted and the higher priority process is allowed to run.

Q. 4: Consider the following C program for execution on a Linux system.

```
/* #define NOSYSCALL */
#ifdef     NOSYSCALL
int getpid() { return 55; }
#endif
int main( int argc, char * argv[] ) {
        int i, a, limit = atoi(argv[1]);
        for ( i = 0; i < limit; ++i )
        a = getpid();
}
```

The system call getpid performs almost no processing. It looks up your pid and returns it. All the time it takes is system call overhead that is present in a system call. Likewise, the procedure call getpid does nothing but return a value. All the time it takes is procedure call overhead that is present in every procedure call. Run this program for 2-3 million iterations and see how long it takes. Then uncomment out the #define and run it for 30-40 million iterations and see how long it takes. What do you conclude about the relative speed of a system call and a procedure call?

Ans 4: Try it yourself.

Q. 5: Suppose we run each of the following scheduling algorithms in a system that is very heavily overloaded. Describe how each of these algorithms act in the face of overloading. Discuss how this overloading affects the average waiting time of short jobs, medium jobs and long jobs (if they are affected differently). That is, discuss how the average waiting time changes (for short, medium and long jobs) for a lightly loaded system to a heavily loaded system. Make sure to discuss the overhead of extra context switches caused by the scheduling algorithm (if any).
FCFS, Round-Robin and Priority based Scheduling

Ans. 5:

- **First-come, First-served:** All jobs are treated the same. When the load goes up the average wait time will go up with it. All jobs have the same wait time. There are no context-switching costs with FCFS.
    - All jobs treated the same.
    - Average absolute wait time for all jobs is the same.
    - No context switching costs.
    - Jobs slowed down according to how long they are, longer jobs are slower down more.
    - No context switching costs.
- **Round-Robin Scheduling:** It treats all jobs equally so all jobs will be slowed down by the same factor. A job that was taking N seconds would take 10*N seconds if the average slowdown was a factor of ten. Moreover, round-robin causes extra context switches since it is a preemptive algorithm.
    - all jobs treated equally.
    - lots of content switches
    - this will perform the worst in an overloaded situation.
- **Priority based Scheduling:** The high priority jobs will still get to use the processor.
    - High priority jobs will be unaffected by the load.
    - Low priority jobs will get no service at all.
    - Favors short jobs the most.
    - Long jobs discriminated against.
    - No extra context switches.
    - This will perform the best in an overloaded situation.

Q. 6: Suppose a new process in s system arrives at an average of four processes per minute and each such process requires an average of 12 seconds of service time. Estimate the fraction of time the CPU is busy in a single processor system.

Ans. 6:
The fraction of time that a CPU is expected to be busy can be estimated if you know the arrival rate and service rate for all the processes. In this case, the arrival rate is 4 processes per minute (one process every 15 seconds), and the service rate is 12 seconds per process. The fraction of time CPU is expected to be busy is (1/15)/(1/12) = 12/15 = 80%.

Q. 7:

Consider the following processes are to be scheduled using, FCFS, Round Robin with time quantum 1 and 4.

	A	B	C	D	E
T_a	0	1	3	9	12
T_s	3	5	2	5	5

Ans 7: Try it yourself.

Q. 8: Assume you have the following processes to execute with one CPU.

Process	Arrival Time	Execution Time
0	0	75
1	10	40
2	10	25
3	80	20
4	85	45

Suppose a system uses RR scheduling with a time quantum of 15 and context switch time is five time units with RR scheduling. What is the turnaround time for process 3.

Ans. 8:

```
Time     Process
0-15     p_0
15-20    context switch
20-35    p_1
35-40    context switch
40-55    p_2
55-60    context switch
60-75    p_0
75-80    context switch
80-95    p_1
95-100   context switch
100-110 p_2
110-115 context switch
115-130 p_3
130-135 context switch
135-150 p_4
150-155 context switch
155-170 p_0
170-175 context switch
175-185 p_1
185-190 context switch
190-195 p_3
195-200 context switch
200-215 p_4
215-220 context switch
```

```
                    220-235 p_0
                    235-240 context switch
                    240-255 p_4
                    255-260 context switch
                    260-275 p_0


            p_3 turn around time = 115
```

Q. 9: Consider two jobs, A and B, in a deadline scheduling system. The deadline for A is before the deadline for B. Explain why we should run A before B, that is, show that if running A then B fails to meet some deadline then running B before A will also fail to meet some deadline.

Ans. 9:
Suppose we run A first and it fails to meet its deadline.. Running B first would mean that we would start A even later and so it would also fail to meet its deadline. Suppose we run A first and B fails to meet its deadline. This means that timeToRun(A) + timeToRun(B) > deadline(B). But since deadline(B) > deadline(A) we also have timeToRun(A) + timeToRun(B) > deadline(A). This means that if we run B first and then A, A will miss its deadline.

Q. 10, Q. 11, Q. 12 and Q. 13.

Discussed in the class on October 20, 2016.