CS342 Operating Systems - Fall 2020
Homework #2

Assigned: Nov 9, 2020, Sunday.
Due date: Nov 21, 2020, Saturday, 23:59 Document version: 1.0

1. Describe the difference(s) between micro-kernel approach and monolithic kernel approach.

2. Why do we have a boot block on a disk?

3. When a system call is made and kernel starts, how does the kernel find the address of the respective system call routine to execute?

4. Why can red-black tree be a good data structure to use in kernel in some cases?

5. Why can we prefer using multiple child processes in developing an application?

6. How many processes will be created (including the initial process) by the following pseudo-code?

```
main()  {
        int i, n;
        n = fork();
        for (i=0;  i < 10;  ++i) {
                If (i % 2 == 0)  {
                        fork();
                }
        }
}
```

7. How many processes (including the initial process) will be created by the following pseudo-code?

```
main() {
        int x;
        x  = fork();
        if (x == 0) {
                fork();
                execve(….);
                fork();
        }
        fork();
}
```

8. Assume there are 16 cores in a computer. The fraction of an application that has to run serially is 0.2. The application creates 16 threads, each running on a different core. What is the maximum speedup that can be obtained?

9. Assume we have the following C program. Assume we run this program once. Which pair of integers are printed out?

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

int x = 5000;
int y = 7000;

void *tfunction()
{
        int y = 500;
        x = 1000;
        printf ("x,y = %d %d\n", x, y);
}

int main()
{
        int ret;
        pthread_t tid;
        int pid;

        printf ("x,y = %d %d\n", x, y);
        pid = fork ();
        if (pid > 0) {
                wait();
                printf ("x,y = %d, %d\n", x, y);
                ret = pthread_create (&tid, NULL, &tfunction, NULL);
                ret = pthread_join(tid, NULL);
                printf ("x,y = %d, %d\n", x, y);
                exit(0);
        }
        else if (pid == 0) {
                x = 100;
                y = 200;
                printf ("x,y = %d %d\n", x, y);
                ret = pthread_create (&tid, NULL, &tfunction, NULL);
                ret = pthread_join(tid, NULL);
                printf ("x,y = %d, %d\n", x, y);
                exit(0);
        }
} // end of main()
```

10. We have the following processes.

	Arrival time (ms)	CPU Time (ms)
A	0	60
B	15	55
C	25	75
D	45	30
E	55	40

Find the finish time and waiting time of each process for the following scheduling algorithms: a) FCFS, b) SJF, c) SRJF, d) RR (q = 10 ms), In case of a tie, select the process with a letter ID that is earlier in the alphabet. A new process is added to the tail when applicable.

11. Assume there are N processes in a ready queue in decreasing order with respect to their lengths (burst length) and IDs. The length of process k is k time units (1 <= k <= N). Process N is head and process 1 is tail. Compute average waiting time (averaged over all processes) for each of the following scheduling algorithms: FCFS, SJF, RR(q=1 time unit). What is the maximum response time that a process can face when a) FCFS is used; b) SJF is used; c) RR(q=1) is used. Assume response time is not including executing time.

12. The following processes are being scheduled using a preemptive, round-robin scheduling algorithm.

	Priority	Burst	Arrival
A	40	20	0
B	30	25	25
C	30	25	30
D	35	15	60
E	5	10	100
F	10	10	105

Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. The length of a time quantum is 10 units. If a process is preempted by a higher priority process, the preempted process is placed at the end of the queue. Show the scheduling order of the processes using a Gantt chart. What is the waiting time of each process? *Note the following: i) a new arriving processes preempts the running process if the priority of the new process is higher; ii) processes with same priority level are served in a round robin manner; iii) the running process can run until it finishes its burst, provided that there is no process ready whose priority is higher than the priority of the running process during this time; iv) if a new process is to be added to queue (i.e., it is not run immediately) it is added to the tail.*

13. Schedule the following 3 periodic processes using earliest deadline first algorithm. Show the Gantt chart. Find out when the third burst of each process ends.

	Period (ms)	CPU Burst (ms)
A	80	20
B	100	30
C	60	20

14. Assume a system is estimating the next CPU burst length of a process using exponential running average method described in the textbook and class. The alpha parameter value is ¼ (0.25). Initial estimation (T0) is 20 ms. What is the estimation value after three bursts of lengths 30 ms, 20 ms, 40 ms, are executed in the given order?

15. Assume you have mutex and condition variables available in your system (implemented already). You will implement semaphores using them. Show the pseudo-code of the implementation of wait() and signal() operations of semaphores.

16. How many different integer sequences (order is important) may be printed out by the following pseudo-code? Write down all of them.

```
Semaphore X = 0; //shared by all processes
Semaphore Y = 0; //shared by all processes
Semaphore Z = 0; //shared by all processes
int n;

main() {
        n = fork();
        if (n==0) {
                signal(X)
                print (10);
                wait (Y);
                print (20);
                  wait(Z);
                  print(30);
                exit (0);
        }
        signal (Y);
        print (40);
        wait (X);
        print (10);
         print(50);
         signal(Z);
}
```

17. Assume a cigarette requires three ingredients (items) to smoke: Tobacco (t), Paper (p) and a Match (m). Assume there are 3 smokers around a table, each of whom has an infinite supply of one of the three ingredients (items) — one smoker has an infinite supply of tobacco, another has an infinite supply of paper, and the third has an infinite supply of matches. Assume there is also a non-smoking agent which has also an infinite supply of these items. The agent enables the smokers to make their cigarettes by randomly selecting and putting two items (out of three items) on the table. Then the smoker having the missing item will take the items from the table (in this way will make the table empty), will make his cigarette, and will be able to smoke for a while. When table becomes empty, agent again chooses two items in random and places them on the table. Another smoker can now smoke (or maybe the currently smoking smoker will take those items again and start smoking again after it has finished its current smoking). This process continues forever. Synchronize the agent and 3 smokers by use of semaphores to act in this way.