Birla Institute of Technology & Science Pilani, Hyderabad Campus
First Semester 2019-2020
CS F372: Operating Systems
Mid Semester Examination (Regular)

Type: Closed Book **Time: 90 minutes** **Max Marks: 100** **Date: 30/09/2019**

Instructions:

- **Answer all questions. All parts of the same question should be answered together.**
- **There are total 4 pages in the question paper. Answer the questions according to the marks allotted.**
- **Calculators are allowed.**

1.a) Write short notes on microkernel and layered kernel. **[2.5 + 2.5 = 5]**

1.b) Explain the different parts of a PCB. **[7]**

1.c) Explain in detail all the steps that are involved in system boot. **[4]**

1.d) Explain the problem associated with asynchronous thread cancellation. How is it solved in deferred cancellation, especially in the Pthreads API? **[4]**

2.a) Explain dual mode of OS operation with a diagram. **[4]**

2.b) What are zombie and orphan processes? Explain using code snippets how they are created. **[6]**

2.c) Explain the difference between parallelism and concurrency. **[3]**

2.d) Explain process context switching. Your answer should contain proper explanation as well as a diagram explaining context switching. **[4]**

2.e) Explain direct and indirect message passing. **[3]**

3.a) Write a C program to implement the following scenario. A parent process creates a child process and the child process creates a grandchild process. The parent creates a pipe that is shared between the parent and the child. The parent writes a message to the pipe and the child reads it from the pipe. The child creates another pipe which is shared between the child and the grandchild. Note that this pipe is not available to the parent and is not the same as the one shared by the parent and the child. The grandchild writes another message to the pipe and the child reads it from the pipe. After obtaining the two messages from the pipes, the child prints the difference in lengths of the two messages. Moreover, each of the three processes prints the PID of its respective parent process. Note that, other than the messages mentioned above, processes are not allowed to exchange any other information. Also, the parent should wait for the child to terminate and the child should wait for the grandchild to terminate. You cannot use `sleep()` to make any of the processes wait. You can assume the contents of the two messages and these need not be taken as user inputs. Your code should include all requisite header files. **[12]**

3.b) Consider the C program given below. Assume all requisite header files are included.

```
int num1 = 6; //global variable

void meth(int y){
    static int x = 5;
    x = x + y;
    num1 = x;
}

int main(){
  int num2 = 10;
  if(fork()){  //Line p
     printf("\n%d %d\n", num1 + num2, getpid());
     meth(num2);
  }
  else if(fork() + fork() + fork()){
     meth(num2);
     meth(num2);
     printf("\n%d %d\n", num1, getpid());
  }
  else   printf("\n%d %d\n", num2, getpid());
}
```

(i) What is the output of the given program? Assume the PID of the main process (before `fork()` is called in **`Line p`**) to be 64137 and the PID of a child process is always just 1 greater than the PID of its parent. Also assume that apart from the processes formed from the given code, no other process is created in the system. Note that the sequence of the output is not important in this case. **[4.5]**

(ii) Explain with proper justification the reason behind obtaining the output. **[7.5]**

4.a) Write a C program for the following scenario using the Pthreads API. Assume that you are working on a Linux OS and have been given a text file named as `a.txt`. The main thread creates a child thread and the child thread creates a reverse copy of `a.txt` and names it as `b.txt`. Here, reverse copy implies that the contents of `a.txt` will be copied into `b.txt` in reverse order of the words. Assume that the command to accomplish reverse copy on the Linux OS that you are working on is `rcp` and the syntax for the `rcp` command is `rcp original_file_name copy_file_name`. The file names should contain the extensions as well. The path to the `rcp` command is `/bin/rcp`. You are not allowed to open and read `a.txt` in your program. Your program should make appropriate use of the `rcp` command. You have to initialize the thread attributes to the default values. The main thread should wait for the child thread to terminate. You are not allowed to use `sleep()` to make the main thread wait. All requisite header files have to be included by your program. **[6]**

4.b) Consider the modified version of the hardware instruction `test_and_set()` and another hardware instruction `reset()` as defined below.

```
boolean test_and_set(boolean *p, boolean *q){
   *p = true;
   *q = !(*q); //! is the NOT operator
   boolean temp = *p XOR *q;
   return temp;
}
```

```
void reset(boolean *p, boolean *q){
   *p = false;
   *q = false;
}
```

Next, consider the following solution to the critical section problem (pseudocode) using the hardware instructions described above and two shared boolean variables `lock` and `enter`. Initially, both `lock` and `enter` are set to `false`. Note that `lock` and `enter` are shared by all the processes.

```
do{
   while(test_and_set(&lock, &enter));
   //critical section
   reset(&lock, &enter);
   //remainder section
}while(true);
```

Assume the two hardware instructions defined above to be atomic. This means if one process is modifying the values of `lock` and `enter` then no other process will be allowed to modify the values of these variables even if the two processes are executing on different processors. Moreover, `test_and_set()` and `reset()` have to be executed without any interruption. Explain in detail how the first process enters the critical section. Are mutual exclusion and progress satisfied by the above mentioned critical section problem solution? Justify your answer with proper explanations. **[6]**

4.c) Suppose there are two processes **`P0`** and **`P1`** accessing three binary semaphores - **`X`**, **`Y`** and **`Z`**. The values of **`X`**, **`Y`** and **`Z`** are initially set to 1. Assume that **`P0`** and **`P1`** are executing in parallel on two different processors whose speeds are same. Consider the following operations performed by **`P0`** and **`P1`** and the corresponding timestamps at which each process intends to perform a particular operation shown in the **Table 1**. A "**–**" in a table cell indicates that the corresponding process performs computational task instead of **`wait()`** or **`signal()`**. Two operations performed by two different processes at the same timestamp are assumed to be simultaneous. Assume the classical definitions of **`wait()`** (with busy waiting) and **`signal()`**. Does this execution sequence lead to a deadlock? Explain your answer by clearly mentioning the states of the semaphores and the processes after each operation. **[2]**

Timestamp	Operation of P0	Operation of P1
t0	wait(X)	-
t1	wait(Y)	wait(Z)
t2	-	signal(Z)
t3	-	wait(Y)
t4	signal(X)	-
t5	wait(Z)	wait(X)
t6	signal(Z)	signal(Y)
t7	signal(Y)	signal(X)

Table: 1

4.d) Consider the two processes **P0** and **P1** and the three binary semaphores **X**, **Y** and **Z** initialized to 1 of **Q 4.c**. All the assumptions related to **Q 4.c** are applicable for this question also. Now consider the following sets of operations performed by **P0** and **P1** along with the timestamp of each operation shown in **Table 2**. Assume the classical definitions of **wait()** (with busy waiting) and **signal()**.

Timestamp	Operation of P0	Operation of P1
t0	wait(X)	-
t1	wait(Y)	-
t2	-	wait(Z)
t3	signal(Y)	-
t4	-	wait(X)
t5	wait(Z)	-
t6	-	signal(Z)
t7	-	wait(Y)
t8	signal(X)	-
t9	-	signal(Y)
t10	signal(Z)	signal(X)

Table: 2

i) Does this execution sequence lead to a deadlock? Explain your answer by clearly mentioning the states of the semaphores after each operation. **[2]**

ii) Would the situation have changed if in **Table 2**, at **t3**, **P0** performs **signal(X)** instead of **signal(Y)** and at **t8**, **P0** performs a computational task instead of **wait()** or **signal()**. All other operations remain same. Assume that there are no other processes in the system and no new processes will enter the system in future. Explain your answer. **[2]**

5.a) Consider the set of processes shown in **Table 3** below along with their arrival times and CPU burst cycles given in milliseconds. The processes are scheduled using the **SRTF** algorithm. In case of tie, the selection is done based on FCFS. Calculate the average waiting time. Your answer should contain a proper Gantt Chart and a detailed calculation. **[6]**

Process	Arrival Time	CPU Burst Cycle (milliseconds)
P1	0	5
P2	1	3
P3	3	2
P4	4	1
P5	6	8
P6	7	4

Table: 3

5.b) Consider the set of processes shown in **Table 4** below along with their arrival times and CPU burst cycles given in milliseconds.

Process	Arrival Time	CPU Burst Cycle (milliseconds)
P1	0	8
P2	0	4
P3	2	3
P4	3	7
P5	13	2
P6	15	17
P7	18	9

Table: 4

(i) The given processes are scheduled using **Round-Robin** scheduling algorithm. Assume the time quantum duration to be 3 milliseconds. Calculate the average waiting time. Your answer should contain a proper Gantt Chart and a detailed calculation. **[6]**

(ii) Assume that the system on which the given processes are scheduled requires 0.09 milliseconds for performing process context switch. Also, assume that none of the processes have any I/O burst and each process on submission to the system directly enters the main memory. Moreover, assume that even if a single process is present in the ready queue, context switch occurs after the expiration of time quantum. Calculate the total turnaround time for all the processes in case of scheduling done using the **Round-Robin** scheduling algorithm. Detailed calculation and steps have to be shown properly. **[4]**

5.c) Consider a system implementing multilevel queue scheduling algorithm for scheduling processes. The ready queue on this system is partitioned into two queues, **Q1** and **Q2**. **Q1** is for foreground processes and **Q2** is for background processes. Processes in **Q1** are scheduled using **non-preemptive SJF** algorithm and the ones in **Q2** are scheduled using **FCFS** scheduling algorithm. Moreover, foreground processes have higher priority than background processes. Consider the four processes - **P1**, **P2**, **P3** and **P4**. The nature, arrival times and CPU burst cycles (in milliseconds) of the processes are shown below in **Table 5**. Calculate the waiting time for **P4**. Your answer should contain a proper Gantt Chart and a detailed calculation. **[2]**

Process	Nature	Arrival Time	CPU Burst Cycle (milliseconds)
P1	Foreground process	0	15
P2	Foreground process	0	9
P3	Background process	0	12
P4	Background process	2	11

Table: 5

****