INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Mid-Spring Semester Examination 2022-23

Subject No.: CS30002 / CS31202 **Subject: OPERATING SYSTEMS**

Duration: 2 hours **Full Marks: 60**

Department/Center/School: Computer Science and Engineering

Specific charts, graph paper, log book etc., required: NA

Special Instructions (if any):

- **Attempt all questions. All parts of the same question should be answered together.**
- **Keep your answers brief and to-the-point. Marks may be deducted for unnecessarily long and unrelated answers.**
- **No clarifications can be provided during the exam. If necessary, make reasonable assumptions, and state any assumption made.**

Question 1 [2 + 3 + 5 = 10 marks]

The following code snippet is meant to implement this situation: a process P creates a shared memory for storing two integers. Then it forks a child process C. C writes a value (e.g., 42) as the first integer in the shared memory. After C writes its value, P reads the value written by C and writes double that value (e.g., 84) as the second integer in the shared memory. Both processes must take the necessary steps for the shared memory to be destroyed. Complete the code snippet by filling in one or more statements in the boxes (i), (ii) and (iii) below. You can assume that all system calls succeed (i.e., you need not check for failures of system calls).

```
int main(){
    int shmid;
    int *a, *b;
    // create a shared array to store two integers
    (i)

    if ( fork() == 0 ){
    (ii)
    }
    else{
    (iii)
    }
    return 0;
}
```

Question 2 **[2 + 4 + 2 = 8 marks]**

The following code snippet is meant to implement this situation: a process P creates a pipe and then forks two child processes C1 and C2. C1 writes the contents of the string *to_write* into the pipe, and C2 reads the contents from the pipe into the string *to_read* and then prints out the string *to_read*. Complete the code snippet by filling in one or more statements in the boxes (i), (ii) and (iii) below. All ends of the pipe must be closed before the program ends. You can assume that all system calls succeed (i.e., you need not check for failures of system calls).

```
int main(){
        char to_write[]="OS";
        char to_read[10];
        pid_t firstChild, secondChild;
        // create a pipe
        (i)

        firstChild = fork();
        if (firstChild > 0){
        (ii)
        }
        else{
        (iii)
        }
        return 0;
}
```

Question 3 **[3 x 4 = 12 marks]**

Draw a Gantt chart showing the execution order of the following processes on a single-CPU machine, and find the waiting time of each process considering the following scheduling algorithms: (a) Shortest Job First, (b) Shortest Remaining Time, (c) Round Robin with time quantum = 2 msec, and (d) Round Robin with time quantum = 1 msec.

Process	A	B	C	D
Arrival time (msec)	0.000	1.001	4.001	6.001
CPU burst (msec)	3	6	4	2

Question 4 **[7 x 2 = 14 marks]**

Write TRUE or FALSE for each statement with justification for your answer. **No marks will be awarded if no justification is given / the justification is not correct. The justification should be between 1-3 sentences.** Assume all function calls (pthread, file read/write etc.) succeed.

(a) The Swapped-Out states in a process state transition diagram are required to keep track of the processes that do not require CPU time.
(b) A SPOOL buffer has a fixed size (say 5 GB) for the printer connected to your unix system. If process A wants to print a very high-resolution image from Hubble Space Telescope of size 1 GB, then printing might still fail in this setting due to insufficient SPOOL buffer size.
(c) Using "sudo" or super user mode a user can delete system files (e.g., files in system folders like /usr/bin) in Unix. So, using "sudo" a user can run their processes (e.g., a process to delete system files) directly in kernel mode.
(d) Running the "sudo" command generates an interrupt.
(e) If a timer interrupt fires while executing a syscall in the context of a process, the process will continue till the syscall handler finish execution and then handle the timer interrupt by invoking code for handling the interrupt
(f) In Linux the kernel scheduler module schedules processes.
(g) Processes which use multiple threads are always faster than their single threaded counterparts.

Question 5 **[4 x 3 = 12 marks]**

For each question below, **give your answer within 1- 5 sentences.** Assume all function calls (pthread, file read/write etc.) succeed.

(a) Recall that for getting some service from the kernel we use the "syscall <num>" instruction where num is the syscall number. The syscall number is just the index in an array of function pointers. Each element in the function pointer array is the starting address of the right syscall handler function. Why doesn't C-runtime just allow using "syscall <starting memory address of right syscall handler>" directly, instead of using index of yet another array (and adding lookup time)?
(b) Rahul wrote a code where he called fork() to create 2 child processes. Then in the parent as well as in both child processes he ran the function veryFun() which defines local variable x and then prints the address of x using printf("%p",&x). Rahul noted the value printed in all three cases is 0x7fff538078bc. Since it is the same, can he use this unchanged memory address of x to update the value of x from parent to children? Why or why not?
(c) Rahul is now practicing threading using pthreads. What is the unique output of the code below (or if it throws an error, write the error message)? Justify your answer? (no marks without justification). Also note that a is a local variable of Thread 1.

```
#include<pthread.h>
#include<stdio.h>
int *addr;
void *thread(void *ptr)
 {
    int type = (int) ptr;
     if(type==1){int a=10; addr = &a;}
    while(1){
    if(*addr==10)*addr=20;
    else if(*addr==20)*addr=10;
    printf("%d%d\n",type, *addr);
  }
 return ptr;
}
```

```
int main(int argc, char **argv)
{
 pthread_t thread1, thread2;
 int thr = 1, thr2 = 2;
 pthread_create(&thread1, NULL, *thread, (void *) thr);
 pthread_create(&thread2, NULL,*thread, (void *) thr2);
 pthread_join(thread1,NULL);
 pthread_join(thread2,NULL);
 return 0;
}
```

(d) Write possible content of file.txt after the following program terminates in unix. Justify your answer (no marks without justification)

```
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
FILE* fd;
void* worker() {fprintf(fd, "%s", "1");exit(0);}
int main(int argc, char** argv) {
   fd = fopen("file.txt","w");
   pthread_t thread;
   pthread_create(&thread, NULL, worker, NULL);
   fprintf(fd, "%s", "2");
   pthread_exit(0);
}
```

Question 6 **[2 + 2 = 4 marks]**

A fork bomb is a rogue program that just forks infinitely and does nothing. It simply looks like int main(){while(1)fork(); return0;} Fork bombs are notorious for eating system resources (like memory) and crashing an OS. The kernel needs to allocate a PCB for each new process. Today, in Linux machines you have to just just reboot if you encounter a fork bomb. However, you are the chief architect of the company which creates BulletproofOS. Among other things BulletproofOS wants to give protection against fork bombs. So, you want to keep an extra data structure which will keep track of (i) how many successors does one process have (children, grandchildren etc.), (ii) how long they are running and (iii) #fork calls / #total instructions for each process and use them in your quest to defeat fork bombs. Design the following:

(a) A new data structure that will be part of PCB to store these values. Justify your design keeping in mind the memory footprint of PCB and the utility of the data structure.

(b) write pseudo code of an efficient algorithm using this data structure to identify if a process is a fork bomb for terminating it. Justify your design keeping in mind the time and space complexity of your algorithm.