भारतीय प्रौद्योगिकी संस्थान खड़गपुर
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
Department of Computer Science and Engineering

CS 60038: Advances in Operating System Design **Mid Semester Examination**

Full Marks: 50 Time: 2 hours Autumn, 2022

Note:

(i) There are THREE questions in this paper. Answer all the questions. The answers should be precise and to-the-point. Marks will be deducted for unnecessary texts.

(ii) Write down the assumptions clearly, if any. No clarifications will be given during the exam hours.

1. (a) Explain how microkernel architecture differs from exokernel architecture (with a figure). What additional benefits does exokernel provide over the microkernel? Do you think this benefit comes at the cost of some other overhead? – explain. [3+2+1]

   (b) Who does implement the resource revocation mechanism in exokernel – the kernel or the libOS? How is the resource protected from unauthorized revocation during the phase of resource revocation in exokernel? [2+2]

   (c) Consider the following C code.

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>
4.
5.  int main() {
6.      char *arr1, *arr2;
7.      arr1 = (char *)malloc(15*sizeof(char));
8.      strcpy(arr1, "I Love CS60038");
9.      arr2 = (char *)realloc(arr1, 30);
10.     strcat(arr2, " taught in CSE");
11. }
```

   Consider that the above code snippet is executed over a libOS. Mention the line numbers in the code where the secure binding will be initialized in the exokernel. Also, write down what exactly will happen in terms of secure binding while executing those statements. Mention the line numbers where the exokernel will check the secure binding. [5]

2. (a) A user task wants to make a `write()` call to the file system to write 48 bytes of binary data. Considering the Linux process state diagram, clearly write down the steps that will be executed during this process. Mention whether and when the process will move from the user mode to the kernel mode and return to the user mode again. [5]

(b) Do you need a context switch when a process changes its mode from user mode to kernel mode (consider Linux processes)? Why does a user mode process call to the stub of a kernel function rather than directly calling the kernel function? [2+2]

(c) Linux implements the process queues as doubly connected linked lists. Do you think a doubly connected link list is the most optimized data structure for all different processes? [2]

(d) In Linux, what is the difference between normal and lightweight processes? Give a scenario when lightweight processes are useful. Is there any possible danger if a lightweight child process is created through the `vfork()` system call, and then the parent and the child are executed simultaneously in parallel? [1+1+2]

3. (a) Why do we use two arrays in the `runqueue` data structure during O(1) scheduling – one active array and one expired array? How does the system handle process aging using these two arrays? [2+2]

(b) For multiprocessor scheduling, why is a spin lock required for process scheduling in Linux? [2]

(c) Consider three tasks $T1$, $T2$, and $T3$, with nice values -10, 0, and $+10$, respectively. Assume that these tasks are forked at time $t = 0$, $t = 50$, and $t = 100$, respectively, and they are added to the runqueue immediately. There are no other tasks in the runqueue at that time. Assume that all the times are in jiffies (the time unit in Linux Kernel, one jiffy = 10ms).

Consider that these three tasks need 300 jiffies to execute all their instructions. These tasks are batch types (no I/O instructions are executed in between). Further, they do not change their priorities in between (due to aging or by direct user interventions). Assume that all other events (context switch, mode change, etc.) take negligible time.

i) Show the execution steps of these three tasks when Linux O(1) scheduler is used. Draw a rough timeline to show when each process is executed and in what order. Indicate the points of context switches and show the states of the active and inactive arrays in the runqueue.

ii) Show the execution steps of these three tasks when Linux Completely Fair Scheduling (CFS) is used. Assume that the scheduling period is 500 jiffies. Consider the following mapping from priorities to weights for the CFS scheduling.

```
static const int prio_to_weight[40] = {
/* -20 */     88761,     71755,     56483,     46273,     36291,
/* -15 */     29154,     23254,     18705,     14949,     11916,
/* -10 */      9548,      7620,      6100,      4904,      3906,
/*  -5 */      3121,      2501,      1991,      1586,      1277,
/*   0 */      1024,       820,       655,       526,       423,
/*   5 */       335,       272,       215,       172,       137,
/*  10 */       110,        87,        70,        56,        45,
/*  15 */        36,        29,        23,        18,        15,
};
```

Figure 1: Mapping from Priorities to Weights (The numbers in the comments give the nice value of the first entry of each row.

NB: You do not need to show the exact execution ordering, but try to show an approximate ordering based on the algorithms. There would be many intermediate events in practice; assume those events take zero time. You can assume any other numbers you feel you need, but mention them clearly in the answer script. [6+8]