CS130 OS 1 Midterm Exam

Nov. 5th. 2020 8:15am-9:55am

Name					
ID					
Problem #	1	2	3	4	5
Points	20	20	25	25	10
Score					
Total					

General Information:

1. This is an open book with one single-side handwritten/ one double-sided printed cheat-sheet examination.
2. You have 100 minutes to answer as many questions as possible.
3. The number in parentheses at the beginning of each question indicates the number of points for that question.
4. You should read all of the questions before starting the exam, as some of the questions are substantially more time consuming.
5. Write all of your answers directly on this paper.

Good Luck!

1. (20 points) Multiple Choice Question (Only one answer is correct)

① (5 points) What is NOT a part of LINUX system

A. Kernel
B. System Library
C. System Utility
D. User Program

② (5 points) In order to boot a system, usually we will follow a few steps, which of the following is NOT a required step:

A. Initialization at a fixed memory location
B. Bootstrap loader
C. Loads kernel
D. Execute a user program

③ (5 points) Which of the following may NOT describe the roles of an OS?

A. Abstraction of hardware devices
B. Protected access to shared resources
C. Prevent the system and users from virus
D. Communication amongst logical entities

④ (5 points) Which of the following OS abstraction is Incorrect?

A. Processor: Thread
B. Memory: Address space
C. Machines: Users
D. Persistent Storage: Files

2. (20 points) **Process:**

a. (5 points) What's the major different component(s) between a PCB and a TCB?

b. (10 points) Read the following code

```
int i;
cpid = fork();
if (cpid > 0) {
   for (i = 0; i < 5; i++) {
       printf("Parent: %d\n", i);
   }
} else if (cpid == 0) {
   for (i = 0; i > -5; i--) {
    printf("Child: %d\n", i);
   }
}
```

What's output of this code?

c. (5 points) What if I want to force to print out all Parent and then print out all Child in the aforementioned code, what should I do?

3. (25 points) **Dinning-Philosoper:** Consider the following code that:

```
do {
        wait (chopstick[i]);
        wait (chopstick[(i+1) % 5]);
        //eat
        signal (chopstick[i]);
        signal (chopstick [(i+1) % 5]);
        //think
        …
} while (true);
```

Please answer the following questions.

a. (5 points) What's the problem with this algorithm?

b. (10 points) What if we add a new semaphore called mutex (the value of mutex is initialized to 1) and place **wait(mutex)** before **wait(chopstick[i])**, where do you put **signal(mutex)** to solve the deadlock problem?

```
do {
        wait(mutex);
        wait (chopstick[i]);
        wait (chopstick[(i+1) % 5]);
        //eat
        signal (chopstick[i]);
        signal (chopstick [(i+1) % 5]);
        //think
        …
} while (true);
```

c. (10 points) What if we force no more than FOUR philosophers to pickup their left-handed chopstick, can you fill the blanks in the following code to implement this solution? Please note that you DO NOT need to fill all the following 6 blanks, only need to fill the blanks where you think is necessary.

```
semaphore r=4;

do {
1.
_____
wait (chopstick[i]);

2.
_____
wait (chopstick[(i+1) % 5]);

3.
_____
//eat

4.
_____
signal (chopstick[i]);

5.
_____
signal (chopstick [(i+1) % 5]);

6._____
//think
...
} while (true);
```

4. (25 points) **CPU Scheduling:** Consider the following **single-threaded** processes, and their arrival times, CPU bursts and their priorities (a process with a higher priority number has priority over a process with lower priority number):

Process	CPU burst	Arrival Time	Priority
A	4	1	1
B	1	2	2
C	2	4	4
D	3	5	3

Please note:

- Priority scheduler is preemptive.
- Newly arrived processes are scheduled last for RR. When the RR quanta expires, the currently running thread is added at the end of to the ready list before any newly arriving threads.
- Break ties via priority in Shortest Remaining Time First (SRTF).
- If a process arrives at time x, they are ready to run at the beginning of time x.
- Ignore context switching overhead.
- The quanta for RR is 1 unit of time.
- Total completion time is the time a process takes to complete after it arrives.

Given the above information, please fill in the following table

Time	FIFO/FCFS	Round Robin	SRTF	Priority
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
Total Completion Time				

5. (10 points) **Synchronization:** Consider an online reading server that can support at MAX 500 readers concurrently to read. How can you add semaphores to the following code to ensure a strict limit of 500 readers connected at a time? Assume that this server can create semaphores and share them amongst the reader threads.

```
void read_session(struct server s){
        login{s};
        read{};
        logout{s};
}
```