CS 314 – Operating Systems
Midterm Exam (PARTIAL SOLUTIONS)
Spring 2024

9:30-10:45 A.M.
February 27, 2024

This exam is a closed-book exam. There are 5 questions in this exam. You have 75 minutes to finish the questions. Please write your answers on separated sheets of blank papers. To avoid grading problems, please staple your papers in the ascending order in the question number. All the questions in this exam are mandatory (no optional question).

Student ID (the last four digits): _____

QUESTION #1 (15 minutes) – 20 points (4 points each)

(1) What are the two primary problems in batch systems?

- Low processor utilization
- Processes that stop due to bugs will be kicked out of the short-term scheduling.

(2) If the processor time-slice in RR process scheduling is accidentally set very (i.e., extremely) long, which of the following algorithm does the RR algorithm will behave most similarly (chose the one that behaves most similarly to the RR)?

(a) FCFS

(b) SJF

(c) SRTF

(d) None of the above

Solution: (a)

(3) Many operating systems use "system calls". What are the reasons for using system calls in operating systems? Mention two different reasons.

① To switch the processor mode from USER mode to SYSTEM mode (or to prevent user processes from directly accessing the hardware resources)

② To provide user processes with high-level abstraction of a computer system (user processes can manipulate hardware components using high-level system calls ("abstraction").

(4) What is the primary difference between "VM (virtual machine)" and "multi-boot" (mention one essential – but the most essential - difference between them)?

In VM, users can switch OSes anytime while they use the OSes (and their applications), while users must shut down an OS before they switch to another OS (it implies that users can use only one OS before that OS is shut down) when "multi-boot" is used.

(5) What problem in "batch system" do "multi-programming (multitasking) OSes" fix and how?

What: It is "low processor utilization". Since batch systems load one program at a time to memory, when a process blocks itself for I/O requests, the process stops running since there is no other process to execute (since the blocked process is the only process in memory).

How: Multi-tasking operating systems solve this problem by loading multiple programs to the memory so that, whenever a process blocks (e.g., for I/O requests), the short-term scheduler switches the processor to another process (thus a processor is actively used as long as enough processes are loaded to the memory).

QUESTION #2 (15 Minutes) – 20 points (4 points each)

(1) How is "thrashing" caused?

Thrashing is caused by excess context switching.

In multitasking operating systems (especially those that use round-robin preemptive scheduling algorithm), the process spends most of its processor cycles just for performing context-switching when the degree of multitasking is too high.

When thrashing happens, processes will not make progress even though the processor utilization high.

(2) Are the following statements true or false (for each of the followings?

(a) A multitasking OS is a time-sharing OS. (FALSE)

(b) A time-sharing OS is a multitasking OS. (TRUE)

Note: for this question, you will earn the full credit only if you provide a correct solution for both (a) and (b). There is no partial credit for this question.

(3) Why can't processes in "ready state" switch to "blocked state (in the short-term scheduler)"?

For a process to switch to the blocked state (in the short-term scheduling), the process must perform an action (call to "wait" on a semaphore, makes an I/O request, and etc.) to an I/O device or a semaphore, which requires the process to be in the running state. Thus, processes in the ready state can not have a transition to the blocked state.

(4) What does “wait” system call to a semaphore exactly perform?

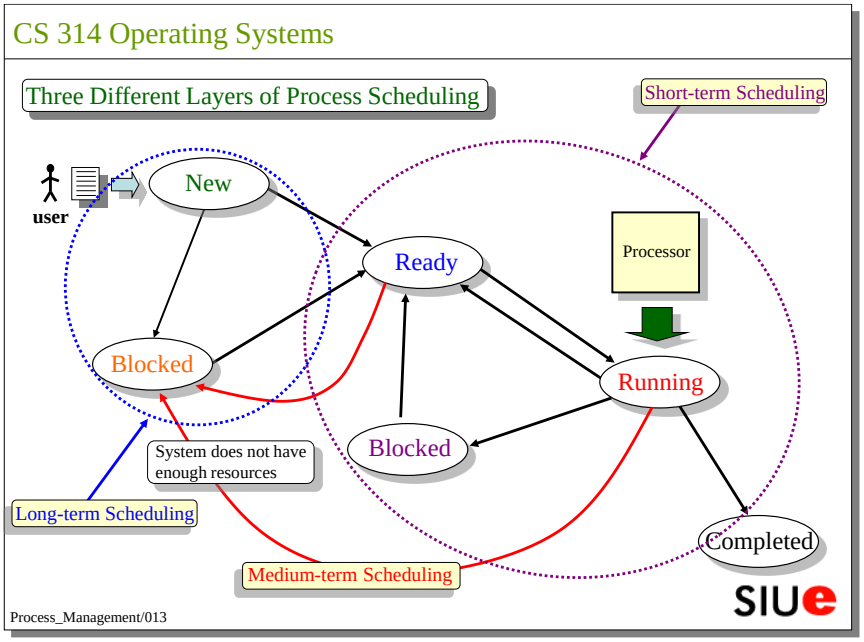
Solution:

Wait

- If S > 0, do S = S - 1 then proceed
- If S = 0, wait on this semaphore

(5) Show a sketch of the integration of the short-term, medium-term, and long-term process scheduling as a directed state-transition diagram (make sure to show all possible state-transitions).

Solution:



QUESTION #3 (10 minutes) – 20 points

Let us assume that you have a section in your program source code file as below:

```
pid1 = fork();

for (i = 0; i < NUM_REPEATS; i++)
{
     pid2 = fork();

     if (pid2 == 0)
     {   printf("I am a child \n");    }
     else
     {   printf("I am the parent \n");   }
}
```

If a label, NUM_REPEATS, is defined by a constant "4", how many times will you see "I am the parent" and "I am a child"? Explain how you found your solution.

QUESTION #4 (15 minutes) – 30 points

Any counting semaphore can be implemented using only binary semaphores. Implement a counting semaphore using only binary semaphore(s).

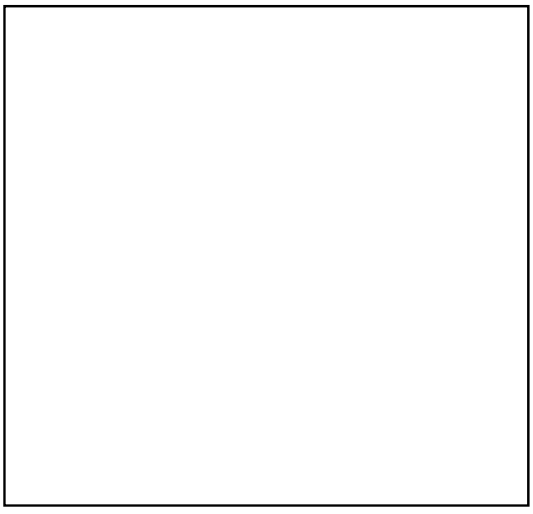
Assumptions and requirement:

- No busy wait (no busy wait) should be used in your implementation.
- More than one binary semaphore can be used (each binary semaphore must be initialized by "1").
- Deadlock should not happen.
- Starvation should not happen.
- As a counting semaphore would do, only the first N processes should be let in.

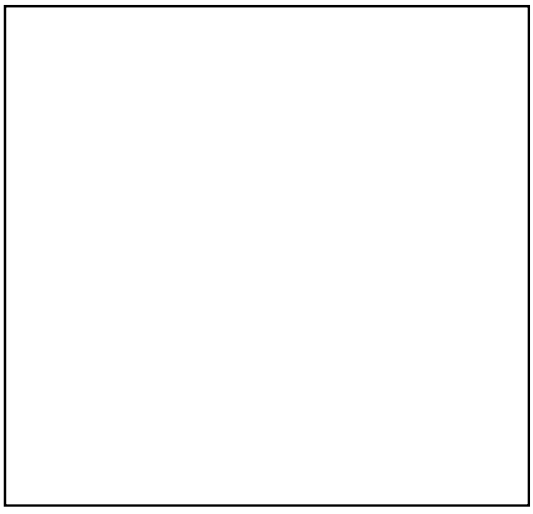
Implement *wait* and *signal* operations for a counting semaphore. Assume that it is your responsibility to make your idea clearly presented in your solutions. Provide your solution using to the following space holders (please do NOT provide your solutions here. Instead, copy the followings to your blank sheet of papers and provide your solutions there).

shared memory and semaphore(s)

"wait" operation



"signal" operation



#5 (20 minutes) – 20 points

When we discussed the technique of avoiding process deadlocks using the SAFE/UNSAFE concepts, we assume the same type of resources (but we had multiple instances of them). It is actually possible to extend the technique for multiple different resource types (while each resource type has multiple instance of the resource).

Assuming that we are going to use the same deadlock avoidance technique for the following four processes (**P_1** through **P_4**) for the five different types of resources (R_1 through R_5) with the following current resource allocation and requirements:

	ALLOCATED					**MAX NEEDED**				
	R_1	R_2	R_3	R_4	R_5	R_1	R_2	R_3	R_4	R_5
P_1	3	1	2	1	1	5	1	2	3	1
P_2	4	0	1	0	0	6	0	4	0	3
P_3	3	2	0	0	1	3	3	1	1	3
P_4	2	3	1	2	0	7	3	1	5	1

The existing vector, ***E***, is represented as: ***E*** (***v***, ***w***, ***x***, ***y***, ***z***), where ***v***, ***w***, ***x***, ***y***, and ***z***, mean the number of the total existing resource instances for R_1, R_2, R_3, R_4, and R_5 respectively.

Note: The existing vector, ***E***, represents the total number of the resource instances that exist for each resource type in the computer system (i.e., it does not mean "how many more available').

Question: what is the smallest* set of ***E*** (***v***, ***w***, ***x***, ***y***, ***z***) that makes the current situation a safe state?

Note*: the term, 'smallest' means the smallest value for (***v*** + ***w*** + ***x*** + ***y*** +***z***).

For anything not stated in this question, assume what we assumed in our discussions for deadlock avoidance in the classroom.

Show all your work as much as you can.