Ph.D. Qualifiers Exam Fall 2020
Operating Systems

Computer Science Department,
New Mexico State University

Exam time: 120 min. The exam contains a total of 6 problems.

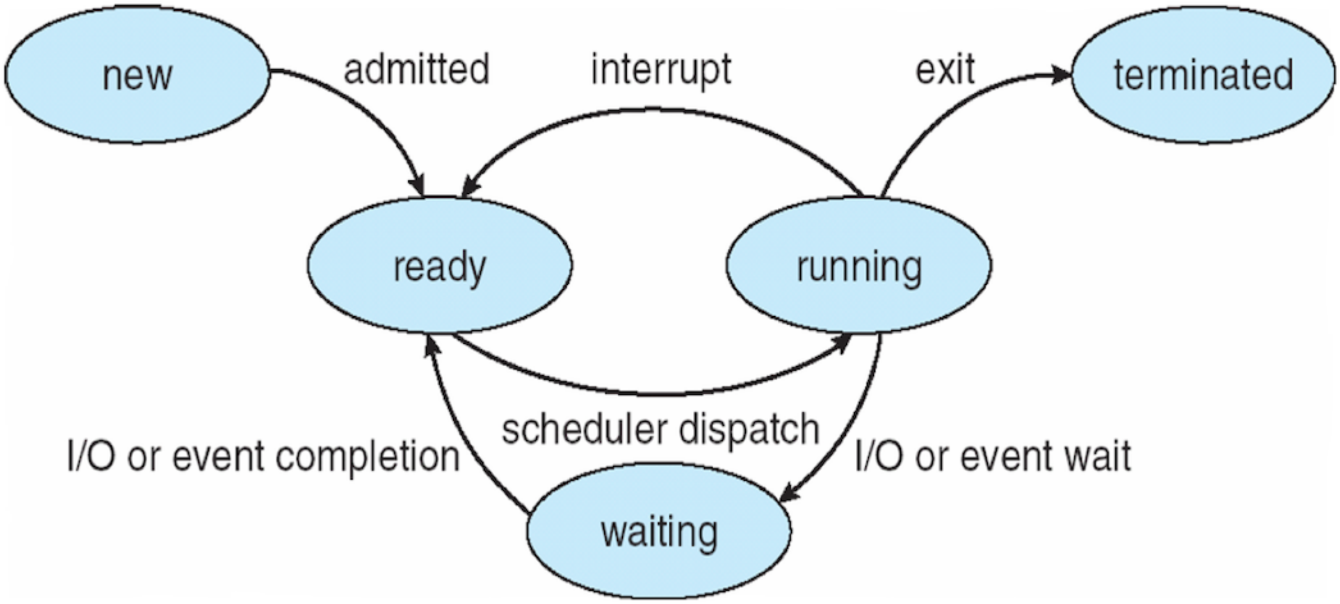
If you believe that you cannot answer a question without making some assumptions, state those assumptions in your answer.

Irrelevant verbosity will not gain you points. Clear and crisp answers will be appreciated.

This is a closed-book, closed-note exam.

Qn1. Answer following questions regarding process state transition.

- Draw clearly the state transition diagram of a process, clearly mark all the states and the event(s), which causes the transition.



- With respect to the above state-transition diagram, please answer the following questions clearly, yet briefly:
  1. Can a process go from the new to the terminated state? Why?
     No, the process has to get to the CPU first. *new → ready → running → terminated* **is the least number of states.**
  2. Can a process go from the waiting to the running state? Why?
     No, to run, the process must first put into ready queue.

- Specify if each of the following processes are I/O-bound, CPU-bound, both, or neither. **Explain in brief, why.**
  1. A file-transfer application.
     I/O bound, a lot of data transfer
  2. The scheduler of the OS.
     CPU bound, most of execution time is spent in solving the problem rather than reading/writing
  3. Video compression/decompression.
     Both, I/O-intensive when CPU is free, CPU-intensive when I/O finishes

Textbook source: Chapter 3.1.2 from page 107 to 109

Qn2. Assume you are given a uniprocessor system with one gigabyte of memory and a 300 gigabyte disk. The OS on the machine has a demand paged virtual memory system with a local page replacement policy and a multi-level feedback queue (MLFQ) CPU scheduler. On the system there are two compute-intensive jobs running: Job-A and Job-B. Job-A has a working set of 50 gigabytes while Job-B has a working set of 100 megabytes. Assume you left the system to run for a while until it reached a steady state with both jobs running.

1. Which job would you expect to have a higher CPU scheduling priority from the MLFQ scheduler?

   Job-A. Because the memory allocated to Job-A is less than its working set, it is thrashing and spending a major portion of its time in servicing the page faults, resulting in a higher priority in MLFQ.

2. Assume you add a second CPU to system, how would this affect the priorities of the jobs?

   It doesn't help. Adding a second CPU cannot relieve thrashing of Job-A. It doesn't affect their priorities.

3. Assume you switch from a local to a global page replacement policy, how does this change affect the priorities of the jobs?

   It will increase the priority of Job B. A global page-replacement algorithm replaces pages without regard to the process to which they belong, so both jobs are thrashing and have a high priority.

   Justify your answer and state any assumptions you make.

Textbook source: Chapter 6.3.6 from page 275 to 277, Chapter 9.6 from page 425 to 429

Qn3. Consider the following program:

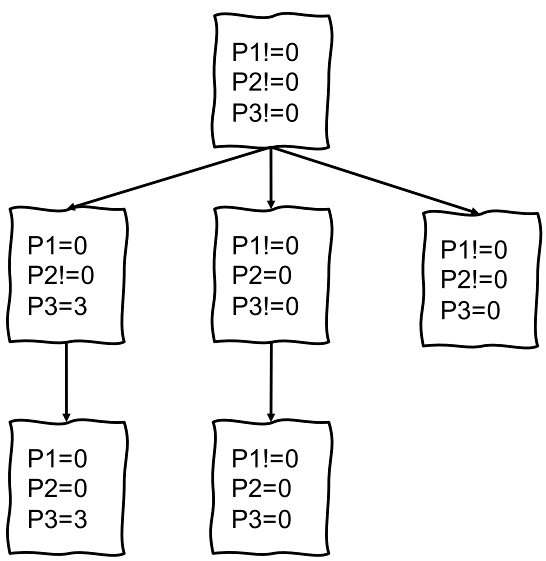
```
#include<stdio.h>
main()
{
        int p1=1, p2=2, p3=3;
        p1 = fork();
        if(p2>0) p2 = fork();
        if(p1>0) p3 =fork();
        if(p1==0) printf("type 1");
        if(p3!=0) printf("type 2");
        if(p2!=0) printf("type 3");
        if((p1>0)||(p2>0) || p3>0))
        printf("type 4");
        if((p2==0) && (p3==0))
        printf("type 5");
}
```

1. What is the return value of fork()?

   The function fork returns an integer equal to 0 to the child process and one different from 0 to the parent process.

2. How many processes are created, including the parent process? Draw a simple tree diagram to show the parent-child hierarchy of the spawned processes.

   6 processes



3. How many times will this program print the following?

   "type 1" **2**

   "type 2" **4**

   "type 3" **3**

   "type 4" **6**

   "type 5" **1**

Textbook source: Chapter 3.3 from page 116 to 119

Qn4. Consider the following snapshot of a system:

	Allocation				*Max*				*Available*			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	2	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

1. What is the content of the $Need$ matrix?

2. Is the system in a safe state? If the state is safe, illustrate the order in which the processes may complete.

3. If a request from process P_1 arrives for $(0, 4, 2, 0)$, can the request be granted immediately, why?

1.

Need			
A	**B**	**C**	**D**
0	**0**	**1**	**0**
0	**7**	**5**	**0**
1	**0**	**0**	**2**
0	**0**	**2**	**0**
0	**6**	**4**	**2**

2. The system is in a safe state. Safe order is $< P_0, P_2, P_1, P_3, P_4 >$**.**

3. The request cannot be granted immediately, because the resulting state is not safe.

Textbook source: Chapter 7.5 from page 331 to 333

Qn5. Consider a simple paging system with the following parameters: 2^{32} bytes of physical memory; page size of 2^{10} bytes; 2^{16} pages of virtual address space.

1. How many bits are in a virtual address?
2. How many bytes in a frame?
3. How many bits in the physical address specify the frame?
4. How many entries in the page table?
5. How many bits in each page table entry? Assume each page entry includes a valid/invalid bit and padding bits to make its size a power of 2.
6. What is the effect on the page table if the physical memory space is reduced by half?

1. 10+16=26 bits

2. 2^{10} **bytes**

3. 22 bits

4. 2^{16} **entries**

5. 32 bits including 22 bits to specify physical frame, 1 valid/invalid bit, 9 padding bits

6. 32 bits including 21 bits to specify physical frame, 1 valid/invalid bit, 10 padding bits

Textbook source: Chapter 8.5 from page 367 to371

Qn6. Consider the following program executed by two different processes P1 and P2. x is the shared variable between two processes. Initially it is set to 10.

```
while(1)
   {
    x=x-1;
    x=x+1;
    if (x!=10)
            printf("x is %d,x");
   }
```

Consider that the processes P1 and P2 are executed on a uniprocessor system. Note that the scheduler in a uniprocessor system would implement pseudoparallel execution of these two concurrent processes by interleaving their instructions, without restriction on the order of the interleaving.

1. What is a race condition? Is this situation an example of a race condition?
2. If there is a race condition, show a sequence (i.e., trace the sequence of interleavings of statements) such that the statement "x is 10" is printed.

   Suggested format: `Pi:  <instruction> <relevant new value>`
3. If there is a race condition, show a sequence that leads to the statement "x is 8" be printed.

   Hint: Remember that the increment/decrements at the source language level are not done atomically, e.g., the assembly language code below implements the single C increment instruction (x = x + 1).

```
register1 = x                    /* load register1 from memory location x */
register1 = register1 + 1        /* increment R0 */
x = register1                    /* store the incremented value back in x */
```

1. **A race condition is a situation, where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place.**

2. **P1: <x=x−1> <x=9>**
   P2: <x=x−1> <x=8>
   P1: <x=x+1> <x=9>
   P1: <if (x!=10)> <x=9, true>
   P2: <x=x+1> <x=10>
   P1: <printf("x is 10")>

3. **P1: <x=x−1> <x=9>**
   P2: <register2=x> <x=9>
   P2: <register2=register2 −1)> <x=9, register2=8>
   P1: <x=x+1> <x=10>
   P2: <x=register2> <x=8>
   P1: <if (x!=10)> <x=8, true>
   P1: <printf("x is 8")>

Textbook source: Chapter 5.1 from page 204 to 206