EECS 678 - Fall 2020

Midterm Preparation Questions

1 Chapter 1

1. Explain the User's and System's view of the operating system.
2. Explain the operating system goals.
3. What is a multiprogramming OS? How does it differ from a batch OS? How does it differ from a time-sharing OS?
4. How does an OS ensure protection between two user processes?
5. How does an OS ensure its own protection as well as that of the other system resources?
6. Sort the following levels of storage hierarchy by their access time and size.
   (a) Registers, (b) cache, (c) disk
7. What factors restrict the number of architected registers?
8. What factors restrict the size of the cache? Why have multiple levels of caches?
9. Explain if true or false:
   (a) The architecture registers are managed by the hardware.
   (b) The user can directly manage the disk.
10. The OS provides a virtualization. Explain.

2 Chapter 2

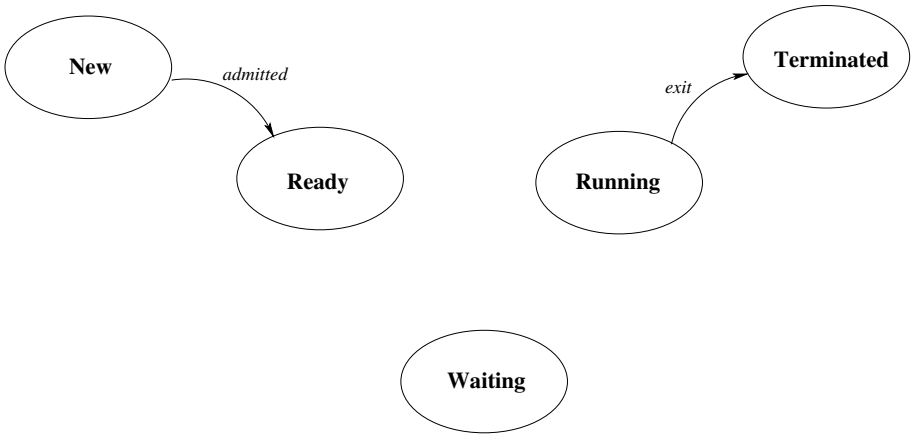
1. What are *system calls*? How do they differ from other function calls?
2. What is an API? How is an API call different from a system call? Which one would you prefer to use in your high-level program?
3. What are the limitations of passing function arguments in registers? How are these limitations overcome?
4. Discuss the definition of both policy and mechanism during software design and implementation.
5. What are the characteristics (in 1-2 lines), advantages and disadvantages of the following kinds of OS designs:
   (a) monolithic design
   (b) layered approach
   (c) microkernel approach.
6. What is virtualization? How is virtualization different from abstraction?
7. What are the benefits of virtualization?

3 Chapter 3

1. Define process. How is a process different from a *program*?

2. Illustrate (draw) the process address space. What component of your high-level C program goes into each section of the process address space?

3. The process state diagram below shows all the possible process states, but only two process transitions. Complete the figure by illustrating how the process changes states on the following events: (a) interrupt, (b) scheduler dispatch, (c) I/O or event completion, and (d) I/0 or event wait.



4. Which of the following typical kinds of CPU instructions should be privileged? Explain your reasoning briefly.
   (a) Set value for hardware timer
   (b) Read hardware clock
   (c) Set a value in the shared memory region.
   (d) Switch from user to OS execution mode
   (e) Turn off hardware interrupts.

5. Describe what a kernel must do to switch context between two processes. What are the benefits and drawbacks of context switching for an OS?

6. What is the purpose of the `fork` and `exec` system calls?

7. Using appropriate system calls, have the program below create a new process, and execute the program `/bin/ps`. It is not important to get all the function arguments and their positions correct.

```
int main()
{
   pid_t pid;

   /*create a new process*/
   pid =
   if(pid == 0){
```

```
   }
   else if(pid < 0){



   }
   else if (pid > 0){



   }
}
```

8. Compare the shared memory and message passing IPC models as regards: (a) characteristics (b) efficiency (c) ease of use. Explain

9. Consider the program below:

```
int main()
{
   pid_t pid;
   int shared_num = 0;

   /* create a pipe */
   ...

   /* create a process */
   pid = fork();
   if(pid > 0){
      /* parent process */
      /* write numbers to the shared variable*/
      int i;
      for(i=0 ; i<10000 ; i++)
         shared_num += 1;

      wait();
      fprintf(stdout, "Shared variable = %d\n", shared_num);
   }
   else{
      /* child process */
      int i;
      for(i=0 ; i<10000 ; i++)
         shared_num -= 1;
   }
}
```

   Use pipes to synchronize access to the shared variable so that the final result printed in the parent process is 0. (You need to first understand why the answer printed may be different from 0).

10. Compare shared memory based IPC with message passing based IPC.
11. What are the advantages and limitations of pipes IPC? How do some other IPC mechanisms address these drawbacks?
12. List and explain in a line the system calls used to setup a shared memory segment.

4 Chapter 4

1. Draw the process address space of a process with two threads. Indicate the stack, heap, text, and data space of the two threads. How is the process address space different if we had two processes instead of two threads?
2. What the (a) advantages and (b) disadvantages of threads over processes?
3. Differentiate between (a) user-level and (b) kernel-level threads. What are their advantages and disadvantages over each other?
4. What is the main disadvantage of the many-to-one multithreading model?
5. What *system call* is used to create threads in Linux?
6. Write a simple program that contains two global variables `num1` and `num2`. The program then: (a) creates two threads, (b) Thread-1 performs `num1+num2` and returns the result, (c) Thread-2 performs `num1-num2` and returns the result, (d) Main thread displays the two returned results and exits.
7. How do system calls *fork* and *exec* operate if invoked from multi-threaded programs? (You may use Linux as an example.)
8. Explain the two mechanisms of thread cancellation.
9. How are *synchronous* and *asynchronous* signals handled in a multi-threaded program?

5 Chapter 5

1. What is a critical section? What are the three conditions to be ensured by any solution to the critical section problem?
2. Consider the following set of processes P1 and P2:

```
P1: {                          P2:{
  shared int x;                  shared int x;

  x = 7;                         x = 7;
  while (1) {                    while (1) {
    x--;                           x--;
    x++;                           x++;
    if (x != 7) {                  if (x != 7) {
      printf(``x is %d'',x)          printf(``x is %d'',x)
    }                              }
  }                              }
}                              }
```

Note that the scheduler in a uniprocessor system would interleave the instructions of the two processes executing, without restriction on the order of the interleaving.

Show an execution (i.e., trace the sequence of inter-leavings of statements) such that the statement 'x is 7' is printed.

3. Understand Peterson's solution to the the synchronization problem.

4. Provide the pseudo code definitions of the following hardware atomic instructions:
(a) TestAndSet

5. A general synchronization solution using locks looks as follows:

```
int mutex;
init_lock(&mutex);

do {
   lock(&mutex);

   // critical section

   unlock(&mutex);

   // remainder section
}while(TRUE);
```

Provide the definitions of *init_lock*, *lock*, and *unlock* when using (a) TestAndSet