EECE.4810/EECE.5730: Operating Systems

Key Questions

Processes & Process Management (Lectures 2 & 3)

QUESTIONS

1. Explain the basic characteristics of a process. What is the difference between a process and a program?
2. What are the operating system's responsibilities with respect to managing processes?
3. What are the key components of a process?
4. What information does a process store in memory, and how is that information organized?
5. What are the possible states in which a process can exist?
6. How does the operating system track all necessary information about a process? When does that information get updated?
7. Describe the basics of how and why a process transitions from one queue to another.
8. What is a context switch?
9. Describe the general steps in process creation and the system calls commonly used to accomplish these tasks.
10. Describe the purpose of the various exec system calls.
11. Describe how processes are terminated.

EXAMPLES

1. Including the initial parent process, how many processes does the program below create? Draw a process tree to support your answer.

```
int main() {
   for (int i = 0; i < 4; i++)
      fork();

   return 0;
}
```

2. What does the program below print?

```
int nums[5] = {0,1,2,3,4};

int main() {
   int i;
   pid_t pid;

   pid = fork();

   if (pid == 0) {
      for (i = 0; i < 5; i++) {
         nums[i] *= -i;
         printf("CHILD: %d\n", nums[i]);
      }
   }
   else if (pid > 0) {
      wait(NULL);
      for (i = 0; i < 5; i++)
         printf("PARENT: %d\n", nums[i]);
   }
}
```

3. Describe the operation of this basic program, which ultimately represents two separate processes.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main() {
   pid_t pid;
   pid = fork();          // Create a child process

   if (pid < 0) {         // Error occurred
      fprintf(stderr, "Fork failed");
      return 1;
   }
   else if (pid == 0) {   // Child process
      printf("Child: listing of current directory\n\n");
      execlp("/bin/ls", "ls", NULL);
   }
   else {            // Parent process—wait for child to complete
      printf("Parent: waits for child to complete\n\n");
      wait(NULL);
      printf("Child complete\n\n");
   }
   return 0;
}
```