CSCI 4210 — Operating Systems
Sample Exam 1 Questions and Prep

Overview

- Exam 1 will be in class on **Monday, February 24, 2020** from **2:00-3:45PM** (please arrive early and sit with empty seats next to you on both sides).
- **We will be using room DCC 308! Not DCC 318. Do not come to 318.**
- Exam 1 will be 105 minutes; therefore, if you have 50% additional time, you will have 160 minutes; if you have 100% additional time, you will have 210 minutes.
- You may bring one double-sided or two single-sided 8.5"x11" crib sheets containing anything you would like; crib sheets will not be collected.
- No calculators, laptops, books, phones, etc.
- Exam 1 will be scanned and graded in Submitty.
- Exam 1 will cover everything we have done thus far this semester through lecture on Thursday, February 20.
- Use the sample questions and suggested topics below to prepare. Also study the quizzes as part of your preparation. **We will review answers to many of these questions in class on Thursday, February 20, so please try to answer as many as you can before class.**

Sample Exam 1 Questions

1. Given the following C program, what is the **exact** output? If multiple outputs are possible, succinctly describe all possibilities. Assume all system calls complete successfully. Also assume that the parent process ID is `128`, with any child processes numbered `768`, `769`, `770`, etc.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
  int rc;
  printf( "ONE\n" );
  rc = fork();
  printf( "TWO\n" );
  if ( rc == 0 ) { printf( "THREE\n" ); }
  if ( rc > 0 ) { printf( "FOUR\n" ); }
  return EXIT_SUCCESS;
}
```

2. Given the following C program, what is the **exact** terminal output? If multiple outputs are possible, succinctly describe all possibilities. Assume that all system calls complete successfully. Further, assume that the parent process ID is `128`, with any child processes numbered `768`, `769`, `770`, etc.

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

int main()
{
  int x = 150;
  printf( "PARENT: x is %d\n", x );

  printf( "PARENT: forking...\n" );
  pid_t pid = fork();
  printf( "PARENT: forked...\n" );

  if ( pid == 0 )
  {
    printf( "CHILD: happy birthday\n" );
    x *= 2;
    printf( "CHILD: %d\n", x );
  }
  else
  {
    wait( NULL );

    printf( "PARENT: child completed\n" );
    x *= 2;
    printf( "PARENT: %d\n", x );
  }

  return EXIT_SUCCESS;
}
```

How would the output change if the `wait()` system call was removed from the code?

How would the output change if variable `x` started at -150 instead?

3. Given the following C program, what is the **exact** terminal output? If multiple outputs are possible, succinctly describe all possibilities. Assume that all system calls complete successfully.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main()
{
  char * a = "POLYTECHNIC";
  char * b = a;
  char * c = calloc( 100, sizeof( char ) );

  printf( "[%s][%s][%s]\n", a + 10, b + 9, c + 8 );

  char ** d = calloc( 100, sizeof( char * ) );
  d[7] = calloc( 20, sizeof( char ) );
  d[6] = c;
  strcpy( d[7], b + 5 );
  strcpy( d[6], b + 4 );

  printf( "[%s][%s][%s]\n", d[7], d[6], c + 5 );

  float e = 2.71828;
  float * f = calloc( 1, sizeof( float ) );
  float * g = f;
  float * h = &e;

  printf( "[%3.2f][%2.2f][%2.1f]\n", *f, *g, *h );

  return EXIT_SUCCESS;
}
```

Add code to deallocate all dynamically allocated memory.

How does the output change if `malloc()` is used instead of `calloc()`?

4. Given the following C program, what is the **exact** terminal output? If multiple outputs are possible, succinctly describe all possibilities. Assume that all system calls complete successfully.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
  printf( "ONE\n" );
  fprintf( stderr, "ERROR: ONE\n" );
  int rc = close( 2 );
  printf( "==> %d\n", rc );

  printf( "TWO\n" );
  fprintf( stderr, "ERROR: TWO\n" );
  rc = dup2( 1, 2 );
  printf( "==> %d\n", rc );

  printf( "THREE\n" );
  fprintf( stderr, "ERROR: THREE\n" );

  return EXIT_SUCCESS;
}
```

5. Given the following C program, what is the **exact** terminal output? Further, what is the **exact** contents of the `output.txt` file? If multiple outputs are possible, succinctly describe all possibilities. Assume that all system calls complete successfully.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main()
{
  int fd;
  close( 2 );
  printf( "HI\n" );

#if 0
  close( 1 );  /* <== add this line later.... */
#endif

  fd = open( "output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0600 );

  printf( "==> %d\n", fd );
  printf( "WHAT?\n" );
  fprintf( stderr, "ERROR\n" );

  close( fd );

  return EXIT_SUCCESS;
}
```

How do the output and file contents change if the second `close()` system call is uncommented?

Add code to redirect all output on `stdout` and `stderr` to the output file.

6. Given the following C program, what is the **exact** output? Further, what is the **exact** contents of the `output.txt` file? If multiple outputs are possible, succinctly describe all possibilities. Assume that all system calls complete successfully. Further, assume that the parent process ID is `128`, with any child processes numbered `768`, `769`, `770`, etc.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>

int main()
{
  int fd;
  close( 2 );
  printf( "HI\n" );

#if 0
  close( 1 );  /* <== add this line later.... */
#endif

  fd = open( "output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0600 );

  printf( "==> %d\n", fd );
  printf( "WHAT?\n" );
  fprintf( stderr, "ERROR\n" );

  int rc = fork();

  if ( rc == 0 )
  {
    printf( "AGAIN?\n" );
    fprintf( stderr, "ERROR ERROR\n" );
  }
  else
  {
    wait( NULL );
  }

  printf( "BYE\n" );
  fprintf( stderr, "HELLO\n" );
  close( fd );
  return EXIT_SUCCESS;
}
```

How do the output and file contents change if the second `close()` system call is uncommented?

7. Given the following C program, what is the **exact** output? If multiple outputs are possible, succinctly describe all possibilities. Assume that all system calls complete successfully. Further, assume that the parent process ID is `128`, with any child processes numbered `768`, `769`, `770`, etc.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
  int rc;
  int p[2];
  rc = pipe( p );
  printf( "%d %d %d\n", getpid(), p[0], p[1] );

  rc = fork();

  if ( rc == 0 )
  {
    rc = write( p[1], "ABCDEFGHIJKLMNOPQRSTUVWXYZ", 26 );
  }

  if ( rc > 0 )
  {
    char buffer[70];
    rc = read( p[0], buffer, 8 );
    buffer[rc] = '\0';
    printf( "%d %s\n", getpid(), buffer );
  }

  printf( "BYE\n" );

  return EXIT_SUCCESS;
}
```

8. Given the table of process burst times, arrival times, and priorities shown below, calculate the turnaround time, wait time, and number of preemptions each process experiences using the following scheduling algorithms: FCFS; SJF; SRT; RR with a time slice of `3` ms; and Priority scheduling with FCFS as secondary scheduling algorithm (note that the lower the priority value, the higher the priority).

PROCESS	BURST TIME	ARRIVAL TIME	PRIORITY
P1	7 ms	0 ms	2
P2	5 ms	0 ms	3
P3	5 ms	1 ms	1
P4	6 ms	4 ms	2

9. Write pseudocode for the scheduling algorithms above; for each, determine the computational complexity of adding a process to the ready queue, of selecting the next process, and of removing a process from the queue. **(Note that this is a practice question and is too general to appear on the exam.)**

10. For each scheduling algorithm above, describe whether CPU-bound or I/O-bound processes are favored. Remember that a CPU-bound process is a process with long CPU burst times, whereas an I/O-bound process is a process with short CPU burst times and long I/O bursts.

11. For each scheduling algorithm above, describe how starvation can occur (or describe why it never will occur). What about indefinite blocking?

12. Given actual burst times for process P shown below and an initial guess of `6` ms, calculate the estimated next burst time for P by using exponential averaging with alpha (α) set to `0.5`; recalculate with α set to `0.25` and `0.75`.

    Measured burst times for P are: `12` ms; `8` ms; `8` ms; `9` ms; and `4` ms.

13. Describe what happens when a child process terminates; also, what happens when a process that has child processes terminates?

14. In a shell, how can pipe functionality be implemented? Describe in detail.

15. Why might `fork()` fail? Why does a "fork-bomb" cause a system to potentially crash and how can a "fork-bomb" be avoided by an operating system?

16. Why is it important to use free() to deallocate dynamically allocated memory?

17. Describe (using code snippets) three ways to cause a segmentation fault.

18. Describe (using code snippets) three ways to cause a memory leak.

19. Describe (using code snippets) three ways to cause a buffer overflow.

20. Why is output to `stdout` and other file descriptors buffered by default?