EEL 4733/5732 Advanced Systems Programming

Exam1

Student Name:

UFID:

Signature:

Important Note There are five questions. Please read all the questions carefully and write your name on all answer pages. Good luck.

Questions

1. (7 pts) Write the pseudocode of a program (process P1) that creates a child process (P2), which in turn creates its own child process (P3). Make sure that your program does not yield more than 3 processes when executed. Let P1 and P3 communicate through a pipe where P1 is the reader and P3 is the writer. P1 should keep reading from the pipe into a buffer of size N, where N is the value of command line argument 1. It continues reading until there won't be any more data. P3 should write "Hello, grandma" to the pipe each time and for a total of M number of times, where M is the value of command line argument 2. Make sure that each process closes the end of the pipe that it does not use. Also, the processes should terminate in the following order: P3, P2, and P1.

2. (9 pts) This question has two parts:

   (a) **Specify** two similarities and two differences regarding two synchronization primitives: condition variables and semaphores.

   (b) In each of the following pseudocode snippets that try to solve the multiple producer multiple consumer problem with infinite data, there is a logical error. **Clearly explain the issue for each part.** You can assume that all synchronization variables and data structures are properly initialized.

      i. Producer-Consumer solution attempt using mutexes and condition variables.

```
Data Declarations
===============
mutex mp, mc;
cond full, empty;
buffer_entry buffer[N];
int numItems=0;

Producer
========
while (true) {
  produce item
  mutex_lock(&mp);
  while  (numItems == N)
      cond_wait(&full, &mp);
  insert item to buffer
  signal_cond(&empty);
  mutex_unlock(&mp);
 }

Consumer
========
while (true) {
   mutex_lock(&mc);
   while (numItems == 0)
      cond_wait(&empty, &mc);
   remove one item from buffer
   signal_cond(&full);
   mutex_unlock(&mc);
   use item
}
```

ii. A Producer-Consumer solution attempt using semaphores.

```
Data Declarations
===============
sem m;
sem full, empty;
buffer_entry buffer[N];
init(m,1); // counter initialized to 1
init(full,N); // counter initialized to N
init(empty,0); // counter initialized to 0
Producer
========
while (true) {
  produce item
  sem_wait(&m);
  sem_wait(&full);
  insert item to buffer
  sem_post(&empty);
  sem_post(&m);
}
Consumer
========
while (true) {
   sem_wait(&m);
   sem_wait(&empty);
   remove one item from buffer
   sem_post(&full);
   sem_post(&m);
   use item
}
```

3. (6 pts) This question has two parts:

   (a) Which functions in the code fragment below cause system calls? Explain briefly.

   (b) What does the following code fragment do? Explain and show the side effect of any system calls on the file descriptor table and/or the file table. You can assume that mysterious.txt is the first file explicitly opened in the code below.

```
...
int fd = open("mysterious.txt",O_WRONLY);
dup2(fd, STDOUT_FILENO);
printf("Nice terminal!\n");
..
```

4. (4 pts) Compare and contrast the pipe() and mmap() system calls by explaining two similarities and two differences in terms of their use for inter-process communication.

5. (4 pts) This question has two parts:

   (a) Explain one similarity and one difference between the system call `wait` and the Pthreads library function `pthread_join`.

   (b) What is wrong with the following code snippet?

```
void *mystery(void *arg) {
   int x = 5;
   // ...
   return &x;
}

int
main(int argc, char *argv[])
{
    pthread_t t;
    int s;
    int *r;

    s = pthread_create(&t, NULL, mystery, NULL);
    s = pthread_join(t, &r);
    if (s != 0)
        errExitEN(s, "pthread_join");
    printf("return value = %d\n", *r);
    exit(EXIT_SUCCESS);
}
```