Roll Number: _____

CS 744 - Autumn 2019 - Quiz 1

1. What is the output of the following program? Justify. Assume the correct #include's are provided. (Hint: The *read (a, b, c)* reads ***c*** bytes from the file designated by file descriptor ***a*** into the buffer designated by pointer ***b***, and *write (a, b, c)* writes ***c*** bytes to the file designated by file descriptor ***a*** into the buffer designated by pointer ***b***. These operations read and write raw bits, without respect to data type). Assume the program load module exists in a file called "q1" in the current working directory. Assume the program is initially called as "q1" with no command-line arguments, *i.e., argc = 1*. Of course, if the *execve()* call in the program is executed, the *argc* value after that will be determined by the relevant parameter to *execve()*. **[2+2 = 4 marks]**

```
int n = 0;
int main(int argc, char *argv[]) {
      pid_t child;
      int status, fd[2];
      char *myargv[]={"q1","stop", NULL};
      char buf[10];
      n++;
      if (argc > 1) {
          read (0, &n, sizeof (n));
          printf ("n = %d\n", n);
          exit (3);
      }
      pipe (fd);
      child = fork();
      if (child != 0) {
         write (fd[1], &n, sizeof (n));
         waitpid (child, &status, 0);
         n = WEXITSTATUS (status);
      } else {
         n++;
         dup2 (fd[0], 0);
         execve ("q1", myargv, NULL);
      }
      printf ("n = %d\n", n);
      return 0;
}
```

Solution:

n = 1
n = 3

Assuming the first execution of "q1" has *argc = 1*, the original process will increment *n* from 0 to 1 and continue on to the *fork()* call without producing any output. It will write its current value of *n* (which is 1) to the pipe, and wait for the child to terminate. The child will increment its copy of *n* (to 2), redirect *stdin* to the pipe, and *exec* itself with the argument "stop". After the *exec*, the child process will start the program over from the beginning, with *argc = 2*. It will read into *n* the value (1) put into the pipe by the parent process, print "n = 1\n", and exit with value 3. The parent will wake up, and store into its copy of *n* the exit status value (3) provided by the child, print "n = 3\n", and exit.

:

Roll Number: _____

2. The village well is large enough so that ***four*** people can draw water at a time. However, if more than four try to do so a fight could break out. Give code for two procedures, **`WaitMyTurn(int pid)`** and **`Finished(int pid)`** to enforce these rules first using semaphores. The process id is an integer from 0... N-1, where N is the population of the village. (*Hint: Each procedure does not need more than one line of code).* **[2+2 = 4 marks]**

`Semaphore MaxPeople = 4; //counting`
 	
`WaitMyTurn(int pid) {`
 	
`MaxPeople.wait(); //or P`
 	
`}`
	`Finished(int pid) {`
	
	`MaxPeople.signal(); // V`
	
	`}`

3. Which of the following would normally be a reason for blocking a process, but not suspending it? *(Circle all the correct choices. 1 mark for each correct choice and -0.5 mark for each incorrect choice)*
   a. the process has been using too CPU time, and the system administrator has decided it should not run until the system is less loaded
   b. the system needs to allocate the memory of the process to another process
   c. the process is waiting for a disk write operation to complete
   d. a user is debugging the process, and stopped it from a terminal
   e. the process is waiting for a child process to terminate

Roll Number: _____

4. Suppose the following code is taken from the **consumer** side of a producer-consumer pair. Suppose the compilation context of the code is correct, so there are #includes for all necessary headers, and all variables are declared and initialized correctly. In particular, assume "*char *shared_buf[NITEMS};*" is declared and *NITEMS* is an integer > 1.

```
char *local_buf = (char *) malloc (N);;
...
pthread_mutex_lock (&M);
if while (next_out == next_in) {
   pthread_mutex_unlock (&M);
   pthread_cond_wait (&CV, &M);
   pthread_mutex_lock (&M);
}
strncpy (local_buf, shared_buf[next_out], N);
next_out = (next_out + 1) % NITEMS;
pthread_cond_signal (&CV-prod);
pthread_mutex_unlock (&M);
```

   What is wrong with the code? List at least three distinct major defects in design and/or coding, and explain or show how to correct each defect. Show corrections by crossing out and/or writing in bits of code above. If it would take more than just a simple change to correct a problem, use a sentence to explain what needs to be done below. (*Don't waste time trying to write out a complete correct solution!*) **[2 * 3 = 6 marks]**

There are more than three errors. The green text inserted above is an attempt to correct these errors, which we describe below:

1. No memory is allocated for the pointer variable *local_buf*.
2. The call to *pthread_cond_wait()* has no loop to test for the secondary condition.
3. The calls to *pthread_mutex_unlock()* and *pthread_mutex_lock()* around the call to *pthread_cond_wait()* are wrong, since waiting on a condition variable requires the mutex be held (already locked) by the calling thread, and on return from the wait operation the mutex is again held (locked) by the thread.
4. There is danger of buffer overflow with *strcpy*. It should be replaced by a use of *strncpy()*. (Note that the proposed solution shown above is not complete, since it does not check to see whether the entire string was copied.)
5. A call to *pthread_cond_signal()* is needed to wake up the producer, after *next_out* is incrementd. (Note that it would be wrong to say that the consumer needs to be awakened by this call, or to place the call inside the *while* loop or *if* statement.

Roll Number: _____

5. The following code is intended to implement one of the operations on a counting semaphore. **[1 + 3 marks]**

```
pthread_mutex_lock (&S.mut);
while (S.count == 0)
      pthread_cond_wait (&S.cond, &S.mut);
S.count --;
pthread_mutex_unlock (&S.mut);
```

The name of the operation implemented above is ___wait OR P____

The other operation (a V or signal) of the counting semaphore can be implemented as follows:

```
pthread_mutex_lock (&S.mut);
S.count++;
pthread_cond_signal (&S.cond);
pthread_mutex_unlock (&S.mut);
```