Problem 1 (0 Points)
This problem considers the Forward Forking (Method 1) of creating a pipeline where each new process creates the next process in the pipeline. In the description below, we use the notation in Homework 4, Problem 3 where \( x[0] \) is the interactive shell. Suppose that the shell is required only to handle the case of only one pipeline and no stdin/stdout redirection; i.e., \( "C[0] | C[1]" \). Furthermore, suppose that the pipeline evaluation function has the following interface:

```c
void eval_pipe( char **argv[] );
```

where `argv[i]` is a pointer to the ith pipeline command stored in argv format; i.e., `argv[i][0]` is the command, `argv[i][1]` is the first argument, etc. of the ith command.

a) The diagram for Method 1 should indicate that the interactive shell \( x[0] \) should wait for the first command \( C[0] \) to complete but not the other commands in the pipeline. Why doesn't the interactive shell in Method 1 wait on all the child processes as in Method 3 where the interactive shell creates the entire pipeline?

b) In Method 1, can \( x[0] \) create all pipes as in Method 3? Explain.

c) Is it simpler for \( x[0] \) to create all pipes or for \( x[1] \) to create pipe \( i - 1 \)? Explain.

d) Draw a diagram of the file descriptor table for each of the processes executing commands \( C[0] \) and \( C[1] \).

e) Give the C/C++ code for evaluating the pipeline assuming that there is always one pipe and all commands are external commands.

Problem 2 (0 Points)
The following is a sequence of commands (with line numbers) entered in rapid succession to the bash shell:

```bash
1  cat /etc/passwd | sleep 60 &
2  ps -o pid,ppid,pgrp,tpgid,cmd
3  ctrl-c
4  fg
5  ctrl-c
```

a) What relationships will be shown by the output from line 2 regarding the process hierarchy, the process group(s), and the controlling terminal?

b) What will be the effect of line 3?

c) What will be the effect of line 4?

d) What will be the effect of line 5?
**Problem 3 (2 Points)**
Consider the following parallel program:

```c
int X = 0; // Global (shared)
int want[10] = {0, ... , 0}; // Global (shared)
void sum() { for (int n=0; n<10; n++) if ( ! want[n] ) { X += n; want[n] = 1; } }
void main() { X=0; parbegin sum(); sum() pend; print(X); }
```

The construct `parbegin S1; S2; ... pend` with statements S1, S2, ... means that the statements can execute in parallel subject to any synchronization primitives. In the above case, there is no synchronization between the two instances of the `inc` function.

Determine the smallest and largest value of the shared variable `X` that will be printed and explain how you arrived at this answer. Assume processes can execute at any relative speed and that a value can be incremented/decremented after it has been loaded into a register.

**Problem 4 (0 Points)**
This problem considers Peterson’s 2-process algorithm given in class.

a) Explain how the algorithm prevents one process from monopolizing the critical section; i.e., prevent starvation?

b) Explain how the algorithm guarantees freedom from deadlock?

**Problem 5 (6 Points)**
Consider the following software-only mutual exclusion algorithm for two processes with id’s of 0 and 1:

```c
boolean blocked[2];
int who;
void P (int id) {
    while (TRUE) {
        blocked[id] = TRUE;
        while (who != id) {
            while (blocked[1-id]) who = id;
        }
        // --| critical section goes here |--
        blocked[id] = FALSE; // exit section
        // --| ... other processing ... |--
    }
}
void main () {
    blocked[0] = blocked[1] = FALSE;
    who = 0;
    parbegin ( P(0), P(1) );
}
```
a) If the processes execute at the same rate as much as possible, in what order will the processes enter the critical section.

b) Summarize how the algorithm attempts to guarantee mutual exclusion in the general case.

c) The algorithm contains some flaws. Give an example where the algorithm exhibits starvation.

d) Give an example where the algorithm exhibits livelock and explain why your example exhibits livelock.

e) In what sense is the algorithm speed-sensitive?