Name: ID#: Lab Time:

CS 4061: Practice Exam 2

Spring 2019

University of Minnesota

Exam period: 30 minutes

Points available: 40

Background: Sigblo C'Ker runs an application called `coordinated_changer` which makes changes to a single file in a safe way. According to the documentation for the code, any number of such processes can be run and they will be coordinated using a semaphore so no data will be lost. While running the program Sigblo accidentally hits the keystroke `Ctrl-c` and finds that `coordinated_changer` closes immediately but on trying to re-run it, Sigblo finds that he cannot get any more instances to run: all seem to "hang" immediately on starting. Looking at the source code for `coordinated_changer`, Sigblo would like to alter it so that `Ctrl-c` will kill `coordinated_changer` safely.

```
1  // rough code for coordinated_changer.c
2  int main(){
3    sem_t *file_lock = sem_open(..);
4
5    perform_setup();
6
7    sem_wait(file_lock);
8    modify_file_for_a_while();
9    sem_post(file_lock);
10
11   perform_cleanup();
12   return 0;
13 }
```

Problem 1 (5 pts): Based on the provided source code, explain why killing one instance of `coordinated_changer` at the wrong time causes all others to stall.

Problem 2 (10 pts): Advise Sigblo on what changes should be made to prevent deadlock in `coordinated_changer`.

Problem 3 (5 pts): Pam Elif is writing a small database system. She would like to support multiple client programs reading and writing the database system simultaneously so is thinking of using a shared memory segment such as is provided by POSIX `shm_open()`. She also would like the database to be backed up by a disk file which a daemon process will occasionally copy from shared memory to disk but is finding the whole arrangement to seem overly complex.
Suggest a simpler mechanism that Pam can use which allows multiple processes to share memory that is automatically written to disk periodically.

Problem 4 (10 pts): Contrast FIFOs and POSIX Message Queues as means for inter-process communication. Describe at least 3 aspects that are similar or different between them.

Background: Consider the small application setup given in the nearby code. The intent is for the program to read commands interactively from a prompt or to allow the program to be launched in the background and read commands from a FIFO that is created. Answer the following questions about the program which reads input from two different sources.

Problem 5 (5 pts): Explain why the `select()` system call is used here rather than simply performing `read()` on the FIFO and standard input sources.

Problem 6 (5 pts): Curiously the FIFO called `input.fifo` is opened in Read/Write mode at line 4 despite the program only reading from it. What problems does this approach avoid?

```
1 int main() {
2   mkfifo("input.fifo", S_IRUSR | S_IWUSR);
3   int stdin_fd = STDIN_FILENO;
4   int altin_fd = open("input.fifo", O_RDWR);
5
6   while(quit ==0){
7     printf("prompt> "); fflush(stdout);
8
9     fd_set fdset;
10    FD_ZERO(&fdset);
11    FD_SET(stdin_fd, &fdset);
12    FD_SET(altin_fd, &fdset);
13    int maxfd = stdin_fd;
14    maxfd = (maxfd < altin_fd ? altin_fd : maxfd);
15    select(maxfd+1, &fdset, NULL, NULL, NULL);
16
17    char buf[1024];
18    if(FD_ISSET(stdin_fd, &fdset)){
19      int n = read(stdin_fd, buf, 1024);
20      buf[n-1] = '\0';
21      execute_command(buf);
22    }
23    if(FD_ISSET(altin_fd, &fdset)){
24      int n = read(altin_fd, buf, 1024);
25      buf[n-1] = '\0';
26      execute_command(buf);
27    }
28  }
29
30  close(altin_fd);
31  remove("input.fifo");
32  return 0;
33 }
```