CSCI-342 Operating Systems
Quiz 6
Chapter 8.5.6 & 8.5.7
Synchronizing Flows & Explicitly Waiting for Signals

Name _____

1. What is a *race*?

2. Specify the race that occurrs in the shell program written in Figure 8.39.

3. Describe a problem that can occurs because of the race discussed above.

4. Describe a solution to the problem described above.

5. Write a spin loop in C.

6. Specify the problem in the shell program written in Figure 8.41.

7. A solution to the problem specified in problem 6 is to replace

```
while(!pid) ;
```

with

```
while(!pid) pause();
```

Does this remedy all problems with the code?

8. A solution to the problem specified in problem 6 is to replace

```
while(!pid) ;
```

with

```
while(!pid) sleep(1);
```

Does this remedy all problems with the code?

9. What is the sigsuspend() function equivalent to?

code/ecf/procmask1.c

```
1   /* WARNING: This code is buggy! */
2   void handler(int sig)
3   {
4       int olderrno = errno;
5       sigset_t mask_all, prev_all;
6       pid_t pid;
7
8       Sigfillset(&mask_all);
9       while ((pid = waitpid(-1, NULL, 0)) > 0) { /* Reap a zombie child */
10          Sigprocmask(SIG_BLOCK, &mask_all, &prev_all);
11          deletejob(pid); /* Delete the child from the job list */
12          Sigprocmask(SIG_SETMASK, &prev_all, NULL);
13      }
14      if (errno != ECHILD)
15          Sio_error("waitpid error");
16      errno = olderrno;
17  }
18
19  int main(int argc, char **argv)
20  {
21      int pid;
22      sigset_t mask_all, prev_all;
23
24      Sigfillset(&mask_all);
25      Signal(SIGCHLD, handler);
26      initjobs(); /* Initialize the job list */
27
28      while (1) {
29          if ((pid = Fork()) == 0) { /* Child process */
30              Execve("/bin/date", argv, NULL);
31          }
32          Sigprocmask(SIG_BLOCK, &mask_all, &prev_all); /* Parent process */
33          addjob(pid);  /* Add the child to the job list */
34          Sigprocmask(SIG_SETMASK, &prev_all, NULL);
35      }
36      exit(0);
37  }
```

code/ecf/procmask1.c

Figure 8.39 A shell program with a subtle synchronization error. If the child terminates before the parent is able to run, then addjob and deletejob will be called in the wrong order.

code/ecf/waitforsignal.c

```
1  #include "csapp.h"
2
3  volatile sig_atomic_t pid;
4
5  void sigchld_handler(int s)
6  {
7      int olderrno = errno;
8      pid = waitpid(-1, NULL, 0);
9      errno = olderrno;
10 }
11
12 void sigint_handler(int s)
13 {
14 }
15
16 int main(int argc, char **argv)
17 {
18     sigset_t mask, prev;
19
20     Signal(SIGCHLD, sigchld_handler);
21     Signal(SIGINT, sigint_handler);
22     Sigemptyset(&mask);
23     Sigaddset(&mask, SIGCHLD);
24
25     while (1) {
26         Sigprocmask(SIG_BLOCK, &mask, &prev); /* Block SIGCHLD */
27         if (Fork() == 0) /* Child */
28             exit(0);
29
30         /* Parent */
31         pid = 0;
32         Sigprocmask(SIG_SETMASK, &prev, NULL); /* Unblock SIGCHLD */
33
34         /* Wait for SIGCHLD to be received (wasteful) */
35         while (!pid)
36             ;
37
38         /* Do some work after receiving SIGCHLD */
39         printf(".");
40     }
41     exit(0);
42 }
```

code/ecf/waitforsignal.c

Figure 8.41 Waiting for a signal with a spin loop. This code is correct, but the spin loop is wasteful.