

Exercise 5

Signal 1

Assume we want to create one child process and count how many times the user press Ctrl-C. Here is the code:

```
1 void handle_ctrlc(int sig) {
2     kill(getppid(), SIGUSR1);
3 }
4 void handle_usr1(int sig) {
5     static int cnt = 0;
6     printf("Ctrl-C pressed %d times\n", ++cnt);
7 }
8 int main() {
9     if (fork() == 0) {
10         signal(____, handle_ctrlc);
11         for (;;)
12     } else {
13         signal(SIGUSR1, handle_usr1);
14         for (;;) // stuck the main process
15     }
16     return 0;
17 }
```

1. Please fill the blank in line 10.
2. There are two bugs in the above code, please explain why the program won't work, what's the phenomenon and **TRY** to fix them.
(Note: actually there is a bug that you can't fix completely)

Signal 2

```
1. volatile sig_atomic_t pre_alarm = 0, post_alarm = 0;
2. void handler_alrm(int sig) {
3.     if (post_alarm != 0) return;
4.     if (pre_alarm != 0) {
5.         sio_puts("forth alarm: "); sio_putl(alarm(0));
6.         sio_puts("\n");
7.     } else {
8.         pre_alarm = 1;
9.         sio_puts("third alarm: "); sio_putl(alarm(1));
10.        sio_puts("\n");
11.        Kill(getppid(), SIGUSR1);
12.        sleep(5);
13.        Kill(getpid(), SIGUSR2);
14.        post_alarm = 1;
15.        sio_puts("alarm done\n");
16.    }}
17. volatile sig_atomic_t child_exit, pid;
18. void handler_usr1(int sig) { sio_puts("usr1\n"); }
19. void handler_usr2(int sig) { sio_puts("usr2\n"); }
20. void handler_chld(int sig) { sio_puts("chld\n"); child_exit = 1; }
21. int main() {
22.     sigset_t mask_all, prev_one;
23.     Sigfillset(&mask_all);
24.     Sigprocmask(SIG_BLOCK, &mask_all, &prev_one);
25.     Signal(SIGUSR1, handler_usr1);
26.     Signal(SIGUSR2, handler_usr2);
27.     Signal(SIGALRM, handler_alrm);
28.     Signal(SIGCHLD, handler_chld);
29.     if ((pid = Fork()) == 0) {
30.         printf("first alarm: %u\n", alarm(2));
31.         printf("second alarm: %u\n", alarm(3));
32.         Sigprocmask(SIG_SETMASK, &prev_one, NULL);
33.         Pause();
34.         exit(0);
35.     }
36.     printf("I'm parent\n");
37.     child_exit = 0;
38.     Kill(pid, SIGALRM);
39.     Sigprocmask(SIG_SETMASK, &prev_one, NULL);
40.     while (!child_exit)
41.         Pause();
42. }
```

Note: the kernel check the pending bits when it switches from kernel to user (e.g. returning from system call). The return of signal handler will trigger an **sigreturn** system call.

1. Please fill the following table:

For visibility, please write **T** if the line will always be printed, **F** if the line will never be printed, **M** if the line might be printed.

Line	Visibility	Possible return values of alarm()
5&6	[1]	[7]
9&10	[2]	[8]
15	[3]	-----
18	[4]	-----
19	[5]	-----
20	[6]	-----
30	T	[9]
31	T	[10]
36	T	-----

2. Please draw the dependency diagram of output:
Note: You could use line number to represent one output line in the diagram. And you do **NOT** need to show the values in the output.
3. Please show the maximum number of SIGALRM **sent** to child and **received** by child, and give one corresponding execution flow for each.