CSC209H Worksheet: Signals Activity

This is not a paper worksheet in the style of the previous in-class activities but more of a guided exercise that we will work through together in the lecture time. The purpose is to explore and play with signals and signal handlers. See the back of this page for some resources.

1. The first task is to write a program called `greeting` that does nothing except run an infinite loop. Work with someone in the classroom who has a computer so that you can compile and run your code.
2. Run your code in one window in the foreground using `./greeting`. Kill it by pressing `Ctrl-c` in the same window. Run it again in the background (`./greeting &`). When running it the background, it shouldn't look like it is running from the prompt. Use `ps` to see that indeed it is still running. Use `fg` to move it to the foreground (`fg ./greeting`).
3. Open another window (on the same machine) and run `ps` again (`ps aux | grep <username>`). What is the pid of your infinite loop program? Now that you know the pid use the `kill` command to kill it. Write the command you used here. There are a number of possibilities that will work and the command will need the pid of your running process.

4. We are going to write a function in your program called `sing` that will eventually be used as a signal handler. What signature is required for that?

5. Write the `sing` function so that it prints the lines of "Happy Birthday" and then returns.
6. Change your program so that it expects one command-line argument that will hold the name of the birthday person. Now change your `sing` function so that it sings using the actual name. Hmmm – You can't change the signature. Why not? Add a global variable for the name.
7. When you run your greeting program, it doesn't sing the song because the sing function never gets called. Write the code to install `sing` as the handler for the `SIGUSR1` signal. You may want to look at the sample code with this week's videos for a reminder of the syntax for installing a signal handler.
8. Now run your compiled greeting program from one window, look up the PID from another and send it a `SIGUSR1` signal. Did it sing? If not, go back and check your code.
9. Add a `sleep(10);` line to the middle of your singing to simulate taking longer to actually sing the song. Now compile and run and send your program a `SIGUSR1` signal from another window. Now, before it finishes the singing, send it a `SIGINT` signal from the other window. What happened? Why?
10. Change your program so that the `SIGINT` signal is **not** delivered to the program in the middle of the birthday song. Use `sigaddset` (this was not in the PCRS videos, so read the man page to learn about it). Repeat the actions from above to confirm that the song is finished before the program is killed.
11. One more experiment: Run your final greeting program in the foreground and kill it with `Ctrl-c` from the same window. Run it again and from another window, send it the `USR1` signal to start the singing. While it is still singing, try to kill your `greeting` program using `Ctrl-c` from its own window (like you just did already.) What happens? Why?

Happy birthday to you,
Happy birthday to you,
Happy birthday dear Michelle,
Happy bithday to you!

CSC209H Worksheet: Signals Activity

C function prototypes

```
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
int sigemptyset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);

struct sigaction {
    void (*sa_handler)(int);
    sigset_t sa_mask;
    int sa_flags;
};
```

In addition to the man pages for each function listed above, see also:

```
man 7 signal
```