CMPUT 379, Assignment 1, Winter 2019

University of Alberta / Department of Computing Science

Instructor: Ioanis Nikolaidis (nikolaidis@ualberta.ca)

(UNIX signals, timers, error detection)

Objective

During the course, we will study numerous forms of inter-process communication (IPC). There are system utilities (like `netstat`) that allow one to observe the IPC activity between processes. There exists however a form of very crude IPC, Unix *signals*, that is not expected to be used for meaningful message exchanges. As such, signal activity usually "flies under the radar" of typical IPC inspection tools, and, as a result, can be used as a covert communication channel. Your assignment is to use signals to create a channel for communicating messages between two processes. The processes are symmetric, i.e., each process sends to, and receives from, the other process.

Specification

You are asked to write a C program that produces an executable that can send and receive messages to/from another process. We will use two processes that are running the same executable code. You are going to use conditional compilation flags to allow you to create two versions of this program. One (the default) uses two signals (`SIGUSR1` and `SIGUSR2`) and the other (conditionally compiled as `-DSINGLE`) uses only one signal (`SIGUSR1`)

When invoked the program reports its own process ID and then expects to read from standard input the process ID of the process with which it will communicate. Subsequently, each line of input is treated as a separate message. For example:

```
% covertsigs
Own PID: 15310
14252
This is my first message
This is my second message
.
%
```

The name of the executable is `covertsigs`. Here `14252` is the first line typed by the user which is the PID of the process to communicate with. Then the messages are entered one-at-a-time. A line with a single period terminates the program (note that it does *not* terminate the execution of the other process (`14252`) with which we were communicating). The receiving process reports the received messages prefixing them with a `!` exclamation mark symbol if it is confident it received it correctly and with a `?` question mark symbol if it has detected an error (yes it is possible to have errors -- read on). For example

```
% covertsigs
Own PID: 14252
15310
! This is my first message
! This is my second message
```

but it could have been (due to errors in the first message):

```
% covertsigs
Own PID: 14252
15310
? ThIWE%@
! This is my second message
```

or any other combination that might results from errors and such. The reason we could have a message in error is because, as you will discover, the signals are not always reliable and also because it depends on how you decide to code the messages using signal(s).

In the interest of readability, the example presented above had one process sending and one receiving. You are expected though to have, at each process, a mix of messages received and messages sent. Consider for example a session transcript like this:

```
% covertsigs
Own PID: 13333
13338
! Hi!
Hi to you too
! how's the weather
Fine
? same h3254dsg 453
Did you say "same here"
! yes
```

Essentially it is a two person chat channel (but generally of low speed and subject to errors).

Design Requirements

The most strict requirement is that the processes are not allowed to use any other form of communication between them except for the specified signals.

A performance-related requirement is that your throughput of delivering messages between the processes should be, on average, 80 characters per minute or better.

Many aspects are left under your own control. Each one is a corresponding design decision that you need to explain and report. You are expected to report on at least the following:

1. How do you code messages for the one signal option (only `SIGUSR1` used)?
2. How do you code messages for the two signals option (both `SIGUSR1` and `SIGUSR2` used)?
3. How do you represent message boundaries?
4. How do you check for errors?
5. What errors may you not be able to catch?
6. How do you handle the interleaving of input/output?
7. Is there any other signal you had to use (and why)? -- such signal is of course not allowed for communication
8. How do you ensure that you do not spend unnecessary CPU cycles?

Deliverables

You should submit your assignment as a single compressed archive file (`zip` or `tar.gz`) containing:

The `Makefile` needed to compile your program. The use of `make` and `Makefile` specifications will also be a requirement in subsequent assignments. The executable produced should be named `covertsigs` and your `Makefile` should at the very least provide targets `covertsigs.single` `covertsigs.double` and `clean`. Depending on whether you specified `covertsigs.single` or `covertsigs.double` as the target, the compilation will use (or not use) the `-DSINGLE` flag when producing the `covertsigs` executable.

You must provide all of the source files needed to produce the executable. Remember that this must be C code *only*, using the C99 style.

A file `readme.md` (plain text or Markdown markup) with your answers to the design questions. At the end of the `readme.md` you must include a section describing how you balanced the work across the two group members. Also indicate whether your code was tested (and running) on the official VM OR on the physical hosts in lab CSC 219.

Monday, January 7, 2019