CMSC 421: Principles of Operating Systems

Homework 2: Signal Your Intent

This homework is due on Thursday, September 27, at 11:59:59 PM (Eastern daylight time). You *must* use `submit` to turn in your homework like so: `submit cs421_jtang hw2 ballot.c candidate.c`

Your programs must be named `ballot.c` and `candidate.c`, and they will be compiled on Ubuntu 18.04 as follows:

```
gcc --std=c99 -Wall -O2 -o candidate candidate.c
gcc --std=c99 -Wall -O2 -o ballot ballot.c
```

(Note the above is *dash*, *dash*, *"std=c99"*, and the other flags likewise are preceded by dashes.) There must not be any compilation warnings in your submission; **warnings will result in grading penalties**. In addition, each code file must be properly indented and have a file header comment, as described on the coding conventions page.

> ***Due to UMBC policy, you may not use the GL systems for this assignment, because you will be using the `fork()` system call.***

In this homework, you are writing an e-voting system for the upcoming midterm elections. The system consists of two programs. The user interface is handled by `ballot`. The candidates that may be voted for are represented by multiple instances of `candidate`.

Part 1: Write Initial Candidate Code

The first task is to write `candidate.c`. This program takes a single command-line argument, the name of the candidate. If no name is given, display an error message and quit. Otherwise, display the name and the process's PID.

Add a global variable, representing the number of votes that candidate received. Set this global variable to zero at startup. Create a signal handler for `SIGUSR1`. Every time the process receives a `SIGUSR1`, increment the global variable. Then create another signal handler for `SIGUSR2`. When the process receive `SIGUSR2`, write **to standard error** the **32-bit unsigned little-endian integer** corresponding to the vote count. That is, **do not print an ASCII representation of the value**, but literally write 32 bits.

Finally, after `candidate.c` has set up the signal handlers, go into an infinite loop. Within the loop, read from standard input. When the return value indicates that standard input was closed, then exit from the loop. Print an exit message and terminate the program.

Now compile and test `candidate.c`. In Linux, while a program is running on the command line, pressing **Control-D** will close stdin to the active process. (This differs from **Control-C**, which sends a `SIGINT` signal.)

```
$ ./candidate 'George Burdell'
I am George Burdell at PID 6628.
(In a second terminal)$ kill -USR1 6628
$ kill -USR1 6628
$ kill -USR2 6628
(In first terminal) <garbage characters>
(Press Ctrl-D)
PID 6628 terminating.
```

The garbage characters are because the terminal is trying to render the byte sequence **0x02 0x00 0x00 0x00** as printable ASCII characters. This time, pipe standard error from `candidate` into a hex viewer:

```
$ ./candidate 'George Burdell' 2> >(xxd)
I am George Burdell at PID 6783.
(In a second terminal)$ kill -USR1 6783
$ kill -USR1 6783
$ kill -USR1 6783
$ kill -USR1 6783
$ kill -USR2 6783
(Press Ctrl-D in first terminal)
PID 6783 terminating.
00000000: 0400 0000                                ....
```

The bolded text above is the output from the very useful `xxd` utility. It shows the `candidate` wrote (to stderr) 4 bytes, and the first byte is the hexadecimal value **0x04**. Test `candidate` thoroughly, by sending varying amounts of `SIGUSR1` and `SIGUSR2` signals, and inspecting its standard output and standard error streams.

Part 2: Write Initial Ballot Code

The `ballot.c` program will be divided into several parts as follows. Begin by inspecting its command line arguments. There must be exactly four arguments (plus the program name itself); if not then display an error message and quit. These four arguments are the names of candidates to appear on the ballot.

Next, create a loop that executes 4 times. Within the loop, call `pipe()` function twice; call these pipes **P1** and **P2**. Store all 16 newly created file descriptors somewhere convenient. Then create another loop that executes 4 times. Within this loop, call `fork()` to create a child process. The first created child will be associated with first **P1** and **P2** pipes, the second child with second **P1** and **P2**, and so forth.

In each child process, after the call to `fork()`, **ensure the child process does not continue forking additional processes!** Call `dup2()` twice and `close()` several times:

1. Use `dup2()` to close stdin and duplicate the reading end of its **P1** pipe.
2. Close the reading end of its **P1** pipe.
3. Call `dup2()` again, to close stderr and duplicate the writing end of its **P2** pipe.
4. Close the writing end of its **P2** pipe.
5. Close the writing ends of all **P1** pipes, not just the **P1** pipes associated with that process.
6. Finally, close the reading ends of all **P2** pipes, not just the **P2** pipes associated with that process.

Then call `execlp()` to execute the `candidate` program. While executing the program, pass on its command line the candidate's name (from `ballot`'s command line.)

In the parent process, after the loop, close the reading end of all **P1** pipes, and the writing end of all **P2** pipes. For now, go into an infinite loop. Compile and test `ballot` like so:

```
$ ./ballot Clinton Bush Obama Trump
I am Clinton at PID 7242.
I am Bush at PID 7243.
I am Trump at PID 7245.
I am Obama at PID 7244.
```

Part 3: Add Ballot Menu Options

Within the parent process, in `ballot.c`, display the following menu after creating all child processes:

```
Main Menu:
  0. End program
  1. Cast ballot
  2. Show vote counts
```

Read from the user a menu option. Perform the requested action as below, then return to the main menu.

If the user selects menu option 0, then close each child's **P1** pipe's writing end. This should trigger each `candidate` process to terminate itself, as that its stdin is closing. Back in the parent process, call `waitpid()` with the child's PID, so as to reap the child. After all children have been reaped, then terminate `ballot`.

If the user selects menu option 1, display the four candidates' names in a submenu. Read from the user a number, zero through three. Send a `SIGUSR1` signal to the `candidate` process with that name.

If the user selects menu option 2, send a `SIGUSR2` to each `candidate` process. Then read from each child processes' **P2** pipe the 32-bit vote count. Display that value as a human-readable string.

Part 4: Add Voter Fraud

No modern election is immune to foreign hostile interference, and your homework is no different. The next step is to add a backdoor to your software.

In `ballot.c`, add a fourth menu option, "Set vote count". When the user selects this option, display the four candidates' names in a submenu. Read from the user a number, zero through three. Then prompt the user for a new vote count. Send the to target `candidate` a `SIGALRM`. Then write to that process's **P1** the new vote count, as a 32-bit unsigned little-endian value.

Add to `candidate.c` a handler for `SIGALRM`. When that signal arrives, interpret the next four bytes read from its stdin as a new vote count. Silently update its global vote count with the read value. If `candidate` does not receive a `SIGALRM`, it is to ignore any bytes read from its stdin. Test your updated `candidate` like so:

```
$ (killall -USR2 candidate; echo -e '\xab\xcd\x98\x87'; killall -USR2 candidate) | (./candidate 'George Burdell' 2> >(xxd))
I am George Burdell at PID 32138.
PID 32138 terminating.
00000000: 0000 0000 0000 0000                      ........
$ (killall -USR2 candidate; killall -ALRM candidate; echo -e '\xab\xcd\x98\x87'; killall -USR2 candidate) | (./candidate 'George Burdell' 2>
I am George Burdell at PID 32144.
PID 32144 terminating.
00000000: 0000 0000 abcd 9887                      ........
```

Observe how in the output from the first command, the vote count remains unchanged. In the second command, the count changes from zero to 0x8798cdab.

> ***The change to `candidate.c` is trickier than first appearance. If your `main()` is reading from standard input when `SIGALRM` arrives, the signal handler cannot also call `read()`. The correct approach is to set a global variable within that signal handler. Change your `main()` to ignore the read byte, or to store that byte into the global vote count.***

Part 5: Add Required Documentation

After you finish everything above, as an experiment temporarily modify `ballot.c` so that after the `fork()`, in the child processes do not close any of the writing ends of **P1** pipes. Now run `ballot` and choose the first option. Observe that child processes are no longer terminating. **At the top of your ballot.c file**, add a comment block describing your observations. Then explain why a child process needs to close all **P1** writing ends, so that it can terminate correctly. *(Make sure you revert your changes prior to submitting your work.)*

Sample Output

Here is a sample output from running `ballot`, after all parts have been implemented. Observe that because multiple processes write to the terminal simultaneously, their outputs may interleave in non-obvious ways.

```
$ ./ballot 'Ethel Rosenberg' 'John Anthony Walker' 'Anna Chapman' 'Maria Butina'
I am Ethel Rosenberg at PID 32340.
I am Maria Butina at PID 32343.
I am John Anthony Walker at PID 32341.
I am Anna Chapman at PID 32342.
Main Menu:
  0. End program
  1. Cast ballot
  2. Show vote counts
  3. Set vote count
Choose an option> (User enters 1)
  0. Ethel Rosenberg
  1. John Anthony Walker
  2. Anna Chapman
  3. Maria Butina
Choose a candidate> (User enters 0)
Main Menu:
  0. End program
  1. Cast ballot
  2. Show vote counts
  3. Set vote count
Choose an option> (User enters 1)
  0. Ethel Rosenberg
  1. John Anthony Walker
  2. Anna Chapman
  3. Maria Butina
Choose a candidate> (User enters 2)
Main Menu:
  0. End program
  1. Cast ballot
  2. Show vote counts
  3. Set vote count
Choose an option> (User enters 1)
  0. Ethel Rosenberg
  1. John Anthony Walker
  2. Anna Chapman
  3. Maria Butina
Choose a candidate> (User enters 2)
Main Menu:
  0. End program
  1. Cast ballot
  2. Show vote counts
  3. Set vote count
Choose an option> (User enters 2)
  Ethel Rosenberg has 1 votes
  John Anthony Walker has 0 votes
  Anna Chapman has 2 votes
  Maria Butina has 0 votes
Main Menu:
  0. End program
  1. Cast ballot
  2. Show vote counts
  3. Set vote count
Choose an option> (User enters 3)
  0. Ethel Rosenberg
  1. John Anthony Walker
  2. Anna Chapman
  3. Maria Butina
Choose a candidate> (User enters 2)
Set vote count to? (User enters 31337)
Main Menu:
  0. End program
  1. Cast ballot
  2. Show vote counts
  3. Set vote count
Choose an option> (User enters 2)
  Ethel Rosenberg has 1 votes
  John Anthony Walker has 0 votes
  Anna Chapman has 31337 votes
  Maria Butina has 0 votes
Main Menu:
  0. End program
  1. Cast ballot
  2. Show vote counts
  3. Set vote count
Choose an option> (User enters 0)
PID 32340 terminating.
PID 32341 terminating.
PID 32342 terminating.
PID 32343 terminating.
```

Other Hints and Notes

- Ask plenty of questions on the Blackboard discussion board.

- At the top of your submission, list any help you received as well as web pages you consulted. Please do not use any URL shorteners, such as goo.gl or TinyURL. Also, do not cite shared data services, such as Pastebin or Dropbox.
- Assume the user will always enter a valid menu option, a number zero through three when selecting a candidate, and a non-negative number when setting a candidate's vote count.
- Display PIDs using the use the `%ld` specifier.
- Periodically run the `ps -ef` command, to check if you have any stray processes. Kill them as necessary, using the `killall` command.
- In `candidate.c`, whenever it receives a signal, `read()` returns -1 and `errno` will be set to EINTR. Be sure you handle this situation.

Extra Credit

Sorry, there is no extra credit available for this assignment.