ITP 30002-01 Operating System, Spring 2020

Programming Exam 10 AM—11:20 AM, June 5 2020

Download `src.zip` from Piazza. Answer each question by completing the indicated source code file in `src.zip`. Note that, while you are taking this exam, you can reference all source code files given in `src.zip`.

Problem 1 (40 points). Complete `char * read_exec(char * exe)` at `problem1/problem1.c`, which runs a given executable `exe` and then returns a string that captures all texts that the `exe` process printed on the standard output. For example, suppose that there exists an executable `hello` which prints out `Hello` to the standard output, and you built a completed version of `problem1.c` as `a.out`. Then, the following scenario is possible:

```
$ ls
hello     a.out       problem1.c
$ ./hello
Hello
$ ./a.out
“Hello
”
```

You must not use `popen` in constructing `problem1.c`. You must not modify the `main` function as given in `problem1.c`. You can assume that the output of `exe` does not exceed 2048 bytes. Note that two example programs `problem1/ex1.c` and `problem1/ex2.c` are given for your reference.

Problem 2 (30 points). Write `chat.c` as a multithreaded chatting program by which a user can send and receive messages simultaneously with another user. Hereafter let us call the executable of a completed version of `chat.c` as `chat`.

`chat` receives two command-line arguments. First argument is to indicate a FIFO for reading message and, and second one is to indicate another FIFO for writing messages. Two processes of `chat` should be able to send and receive messages with each other if the same pair of two FIFOs are given to each process, but in an opposite order, as shown in the following command example:

```
$ ./chat channel1 channel2
```

```
$./chat channel2 channel1
```

For two processes sharing two FIFOs, if a user can give a text line to one process, then it should be printed on the other process's standard output. You can assume that the maximum length of a line given by a user is 256, and two FIFOs given to the program always exist. In addition, once a user presses Ctrl+C, the program must print out `Goodbye` on the standard output and terminates.

In using FIFO, make sure on the following two points:

- Set `O_NONBLOCK` option in addition at opening these two FIFOs in `chat.c`. Otherwise, an opening of a FIFO for reading/writing will be blocked if there is no corresponding writer/receiver.
- Note that `read()` returns -1 when no process has opened the same FIFO for writing.

A pair of example programs, `problem2/sender.c` and `problem2/writier.c`, are given for your reference. You can re-use these source code files,

Problem 3 (30 points). Complete `problem3/mysem.c` by implementing a counting semaphore using Pthread mutex and condition variable (it can be used as a library).

ITP 30002-02 Operating System, Spring 2020

Programming Exam 11:30 AM—12:50 PM, June 5 2020

Download `src.zip` from Piazza. Answer each question by completing the indicated source code file in `src.zip`. Note that, while you are taking this exam, you can reference all source code files given in `src.zip`.

Problem 1 (40 points). Complete `char * read_exec(char * exe)` at `problem1/problem1.c`, which runs a given executable `exe` and then returns a string that captures all texts that the `exe` process printed on the standard output. For example, suppose that there exists an executable `hello` which prints out `Hello` to the standard output, and you built a completed version of `problem1.c` as `a.out`. Then, the following scenario is possible:

```
$ ls
hello     a.out       problem1.c
$ ./hello
Hello
$ ./a.out hello
“Hello
”
```

Here's some constraints and assumptions: (1) You must not use `popen` in `problem1.c`, (2) You must not modify the `main` function as given in `problem1.c`, and (3) You can assume that the output of `exe` does not exceed 2048 bytes. Note that two example programs `problem1/ex1.c` and `problem1/ex2.c` are given for your reference.

Problem 2 (30 points). Write `chat.c` as a multithreaded chatting program by which a user can send and receive messages simultaneously with another user. Hereafter let us call the executable of a completed version of `chat.c` as `chat`.

`chat` receives two command-line arguments. First argument is to indicate a FIFO for reading message and, and second one is to indicate another FIFO for writing messages. Two processes of `chat` should be able to send and receive messages with each other if the same pair of two FIFOs are given to each process, but in an opposite order, as shown in the following command example:

```
$ ./chat channel1 channel2
```

```
$./chat channel2 channel1
```

For two processes sharing two FIFOs, if a user can give a text line to one process, then it should be printed on the other process's standard output. You can assume that the maximum length of a line given by a user is 256, and two FIFOs given to the program always exist. In addition, once a user presses Ctrl+C, the program must print out `Goodbye` on the standard output and terminates.

In using FIFO, make sure on the following two points:

- Set `O_NONBLOCK` option in addition at opening these two FIFOs in `chat.c`. Otherwise, an opening of a FIFO for reading/writing will be blocked if there is no corresponding writer/receiver.
- Note that `read()` returns -1 when no process has opened the same FIFO for writing.

A pair of example programs, `problem2/sender.c` and `problem2/writier.c`, are given for your reference. You can re-use these source code files.

Problem 3 (30 points). Complete `problem3/mysem.c` by implementing a counting semaphore using Pthread mutex and condition variable.