ITF22519: Introduction to Operating Systems

Fall Semester, 2021

Lab8: InterProcess Communication(IPC) 1

October 15, 2021

In some applications, different proccesses may have to coordinate with each other, especially when they have to solve a common problem. In this lab, we will look at several approaches that two (or more) proccesses in `one machine` or in `the same system` can communicate.

IPC is nothing but a way for one proccess to exchange a piece of information with another process. They can communicate by using **Pipe, Signal, Socket, Shared Memory, Message Queue and Semaphores**. In this lab, we will go through the first three approaches.

Before you start, remember to `commit` and `push` your previous lab to your git repository. Then, try to `pull` the new lab:

```
$ cd Introduction2OS/labs
$ git pull upstream main
$ cd lab8
```

1 Pipe

The simplest form of IPC is a `pipe`. In the textbook, pipe is a pseduo file which is considered as a special file. There are unnamed pipe and named pipe. The later is also called FIFO.

1.1 Unnamed Pipe

You have a bit experience with Pipe in lab1 assignement. When you pipe the output of one program into the input of another, you are creating an unnamed pipe. For example, this command in Bash would take output from `ps` and `pipe` it to `less` so that you can see it.

```
$ ./ps -el | less
```

Another way to create an unnamed pipe is to use the `pipe(2)` system call in C. This function creates a unidirectional pipe and returns a two element array where the first element is the read and the second element is to write end of the pipe. The `pipe()` function can be called before the `fork()` system call. Then, the child and parent processes can use it to communicate. For more information about pipes, use `pipe(2)` and `pipe(7)`.

Task 1

The program `pipe_test.c` uses unnamed pipe. Run the code example and explain the output.

1.2 Named Pipe

Another approach to create a pipe is a named pipe, known as a FIFO due to its behavior, by using `mkfifo` command in your terminal or `mkfifo(3)` library call in C. A FIFO behaves exactly the same way as a pipe except that it has a name so that any process can reference to it. Now, open two terminals in the same directory and create a new FIFO using `mkfifo`.

```
$ mkfifo fifo_test
```

In one terminal, run the following command:

```
$ cat fifo_test
```

This command is now watching FIFO and will show anything written to the FIFO. Now, in another terminal, type the following command:

```
$ echo "Hi FIFO" > fifo_test
```

Task 2

- What happens when you run the `echo` command?
- What happens if you run `echo` command first and then `cat` command?
- Use `man fifo(7)`, where is the data is sent through FIFO store?

Task 3

Write two programs `read_fifo.c` and `write_fifo.c` which will be run on two terminals:

- `write_fifo.c` (while true): reads one line that user enters into one terminal and writes it to a named fifo.
- `read_fifo.c` (while true): reads any input from the same named fifo and print them out to the other terminal.
- Do we need to synchronize the two programs to ensure that the read operation will always happen after the write operation?

2 Signals

You have been using signal so far but you may not aware of it. Signal is a special command that a program receives from kernel mode in operating system. Here is one example. When you press Ctrl-C in your terminal to close a program, you send SIGNIT (or kill) signal to that program. When you press Ctrl - to quit a program, you send SIGQUIT to the program. For more information about signal, use `man page signal`. They are well defined in Linux. However, you can define your own sginals and signal handlers and use them for IPC. Lets do simple practice with Ctrl-C.

Task 4

Compile and run the program `sig_test.c` and answer the following questions.

- What happens when you try to quit the program by using `CTRL + C`?. Explain the result.
- What is the signal number that `CTRL + C` send?
- To exit the program?

Task 5

Compile and run the program `sig_fork.c` and answer the following questions:

- What is the output of the program?
- Explain the output, specify how variable `count` is increased in child and parent processes.

Task 6

Write a program `sig_com.c` that exploits signal for the communication between child and parents processes as follow:

- Child process:

```
while(true)
read input from user
if input is "quit"
break the loop
else
send SIGUSR1 to the parent using kill function
```

- Parent process waits for the child to exit. On receiving `SIGUSR1` signal, print a message: "Received SIGUSR1 from my child".

3 Socket

Socket is a special of pipe in UNIX systems. They allow two-way communication between two processes which may be running on the same computer or two different computers in a network.

Task 7

Use man pages `socket(7), socket(2)` to answer the following questions:

- What are the six types of sockets?
- What are the two domains that can be used for local communications?

4 What To Submit

Upload your related files and reports to GitHub repository.