Lab Assignment: Interprocess Communication with pipe() and execve()

Start

Log on to Odin and do the following...

```
$ cd 3600
$ ./lab-start.sh
$ cd e
$ vi vrlab14.c
```

Using a Makefile might be helpful.
Remember to check for compile warnings.

Files to be collected: **vrlab14.c**

Objective

In this lab, you will implement a program that demonstrates interprocess communication (IPC) using:

- fork() to create child processes
- pipe() to communicate between processes
- execve() to re-execute the same program in different roles

You will simulate a **producer-consumer system**, where:

- The **producer** generates a random two-digit number and writes it to a pipe
  - Producer exits using first digit of the random number
- The **consumer** reads the number from the pipe
  - Consumer exits using second digit of the random number
- The **parent** waits for both to finish and exits using the full two-digit number

- The **parent** must also accept a command-line argument that will be used as a random number seed value. The value will be passed to the producer, as the producer actually generates the random number. Seed the random number generator: **srand(seed_number);**

Purpose of Lab

The purpose of this lab is to use IPC to communicate information between three separate processes. First, the producer communicates with the consumer via the information sent through the pipe. Then, both the producer and consumer communicate with the original process using the program's exit code. By default, file descriptors are shared among processes even after replacing the process image.

Code Skelton

The provided code defines different roles based on argv[1]:

```
if (argc == 1) {
    // parent mode
    // create pipes and open log file
    // launch producer and consumer processes with execve()
    // *hint* do you need to tell the new process any information about
the pipe?


        // wait and log the exit status of the producer and consumer
        waitpid(cpid_producer, &wstatus1, 0);    // wait for producer
        printf("Producer exited with status %d\n",
WEXITSTATUS(wstatus1));
        dprintf(logfd, "Producer exited %d\n", WEXITSTATUS(wstatus1));

        waitpid(cpid_consumer, &wstatus2, 0);    // wait for consumer
        printf("Consumer exited with status %d\n",
WEXITSTATUS(wstatus2));
        dprintf(logfd, "Consumer exited %d\n", WEXITSTATUS(wstatus2));
       short exit_status = WEXITSTATUS(wstatus1)*10 +
WEXITSTATUS(wstatus2);
        printf("Main process exit status: %d\n", exit_status);
        dprintf(logfd, "Main process exit status: %d\n", exit_status);
        close(logfd);

        return exit_status;

} else if (strcmp(argv[1], "PRODUCER") == 0) {
        // producer mode
        // generate random number
        // write it to the pipe
```

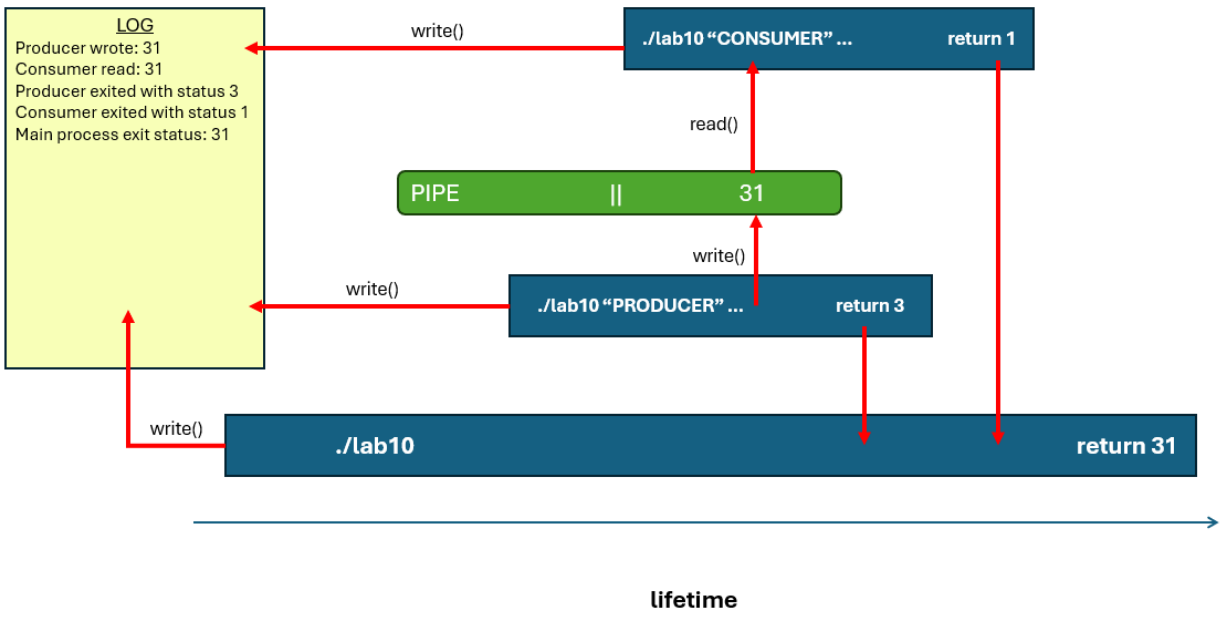
```
        // log
        // return first digit as exit code

    } else if (strcmp(argv[1], "CONSUMER") == 0) {
        // consumer mode
        // read random number from pipe
        // log
        // return last digit as exit code
    }
```

Example Execution

```
(base) mkausch@mkausch:/mnt/c/my_docs/Mike_College/CSUB/Odin_Arch/arch/3600/ta$ ./rvlab14
Producer wrote: 29
Consumer read: 29
Producer exited with status 2
Consumer exited with status 9
Main process exit status: 29
(base) mkausch@mkausch:/mnt/c/my_docs/Mike_College/CSUB/Odin_Arch/arch/3600/ta$ cat log.txt
Consumer read: 29
Producer wrote: 29
Producer exited 2
Consumer exited 9
Main process exit status: 29
```

Execution Timeline


LOG
Producer wrote: 31
Consumer read: 31
Producer exited with status 3
Consumer exited with status 1
Main process exit status: 31
write()
./lab10 “CONSUMER” ...
return 1
read()
PIPE
||
31
write()
write()
./lab10 “PRODUCER” ...
return 3
write()
./lab10
return 31
lifetime