CprE 308 Laboratory 4: Inter-Process Communication

Department of Electrical and Computer Engineering
Iowa State University

1 Submission

Fill the given `308-lab4-report-template.doc` and submit it through Canvas. To protect the format of the template doc, you can convert it to PDF before submitting. Feel free to adjust the answer boxes if needed.

- 10 pts - A summary of what you learned in the lab session. This should be no more than two paragraphs. Try to get at the main idea of the exercises, and include any particular details you found interesting or any problems you encountered.
- 80 pts - A write-up of each experiment in the lab. Each experiment has some items you need to include. For output, screen-capture or copy-paste results from your terminal and summarize when necessary. For program code, include comments that explain the important steps within the program. Include all relevant details.

Due: One week after your lab session.

2 Introduction

In this lab session, you will learn about Linux signals through exercises 3.1 - 3.5 and the Inter-Process Communication (IPC) mechanisms through exercises 3.6 - 3.8.

Before you proceed to the exercises, read the overview of signals and IPC mechanisms in the man pages:

- `man 7 signal`
- `man svipc`

Execute the C programs given in the following exercises. Observe and interpret the results. You will learn about some of the different ways of communication between Unix processes by performing the suggested experiments. You are encouraged to alter the programs to run your own experiments or to figure out what's going on. Sometimes a well placed printf will tell you a lot about the program.

3 Excercises

3.1 Introduction to Signals

Read the overview of signals (`man 7 signal`) and the description of signal() call (`man signal`). Create a program with code below:

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void my_routine();

int main() {
    signal(SIGINT, my_routine);
    printf("Entering infinite loop\n");
    while(1) {
        sleep(10);
    } /* take an infinite number of naps */
    printf("Can't get here\n");
    return 0;
}

/* will be called asynchronously, even during a sleep */
void my_routine() {
    printf("Running my_routine\n");
}
```

Compile and run the program. Press CTRL-C a few times and then press `CTRL-\`.

(3 pts) After reading through the man pages on signals and studying the code, what happens in this program when you type CTRL-C?

(3 pts) What is the signal handler function for this code?

(3 pts) Remove the `signal(...)` statement in main. Recompile and rerun the program. Type `CTRL-C`. Why did the program terminate? Hint: find out how a program usually reacts to receiving what `CTRL-C` sends (man 7 signal).

(3 pts) Replace the `signal(...)` statement with `signal(SIGINT, SIG_IGN)`. Run the program and type CTRL-C. What's happening now? (HINT: Look up `SIG_IGN` in `man signal`)

(3 pts) The signal sent when `CTRL-\` is pressed is `SIGQUIT`. Replace the `signal(...)` statement with `signal(SIGQUIT, my_routine)` and run the program. Press `CTRL-\`. Why can't you kill the process with `CTRL-\` now?

3.2 Signal Handlers

When a signal is received, the integer value correlated to the signal can also be passed to the signal handler. Create a program with the following code:

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void my_routine(int);

int main() {
    signal(SIGINT, my_routine);
    signal(SIGQUIT, my_routine);

    printf("My pid is: %d\n", getpid());
    printf("Entering infinite loop\n");
    while(1) { sleep(10); }
    printf("Can't get here\n");
}

void my_routine(int signo) {
    printf("The signal number is %d.\n", signo);
}
```

Compile and run the program. Press `CTRL-C` and `CTRL-\` a few times. You can stop the program using the kill command (`kill <pid>`) in another bash instance.

(5 pts) What are the integer values of the two signals? What causes each signal to be sent?

3.3 Signals for Exceptions

Signals can be used to handle exceptions. In this exercise, you will create a program that handles the division-by-zero exception using signals. The signal sent for division by zero is SIGFPE. Write a program that prints out "Caught a SIGFPE" when receiving a SIGFPE signal. The program should also create a division-by-zero scenario like the code below:

```
    int a = 4, b = 0;
    a = a/b;
```

Add an exit statement `exit(0)` at the end of the signal handler so the program will terminate after getting the signal.

In the report, please:

(10 pts) Include your source code

(5 pts) Which of the following statements should come first to trigger your signal handler? Explain why.

- the `signal()` statement
- the division-by-zero statements

3.4 Signals using alarm()

Look up the man pages of function calls `atoi` and `alarm`. Read the following code and find out what it does.

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

char msg[100];
void my_alarm();
int main(int argc, char * argv[]){
    int time;
    if (argc < 3) {
        printf("not enough parameters\n");
        exit(1);
    }
    time = atoi(argv[2]);
    strcpy(msg, argv[1]);
    signal(SIGALRM, my_alarm);
    alarm(time);
    printf("Entering infinite loop\n");
    while (1) { sleep(10); }
    printf("Can't get here\n");
}

void my_alarm() {
    printf("%s\n", msg);
    exit(0);
}
```

(4 pts) What are the input parameters to this program, and how do they affect the program?

(6 pts) What does the statement `alarm(time)` do? Mention how signals are involved.

3.5 Signals and fork

Observe and explain the behavior of the following program.

```
#include <signal.h>
```

```
void my_routine( );
int ret;
int main() {
    ret = fork( );
    signal(SIGINT, my_routine);
    printf("Entering infinite loop\n");
    while(1) { sleep(10); }
    printf("Can't get here\n");
}

void my_routine() {
    printf("Return value from fork = %d\n", ret);
}
```

Run the program and trigger the SIGINT signal.

(2 pts) How many processes are running?

(2 pts) Identify which process prints out which message.

(2 pts) How many processes received signals?

3.6 Pipes

A Unix pipe is a one-way communication channel. A pipe is used to pass a character stream from one process to another. On the command line, you've already seen a pipe:

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

will take output from ps and pipe it to less so you can view it.

Pipes are declared using the pipe function. The pipe function takes an array of two integers and returns two file descriptors: the read end will be `p[0]`, and `p[1]` will be the write end. A file descriptor is a tag that describes a resource to the operating system. You pass the file descriptor to the operating system on read and write calls so the operating system knows which resource you're reading from or writing to. There are limits to how many file descriptors can be open in any process, and because they are system resources, there is also a system limit on the total (i.e., how many pipes can be there one time).

Observe the output of this program and explain.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>


#define MSGSIZE 16
int main() {
    char *msg = "How are you?";
    char inbuff[MSGSIZE];
    int p[2];
    int ret;
    pipe(p);
    ret = fork( );
    if (ret == 0) {
        write(p[1], msg, MSGSIZE);
    } else {
        sleep(1);
        read(p[0], inbuff, MSGSIZE);
        printf("%s\n", inbuff);
    }
    exit(0);
}
```

Include answers to the following questions in your report:

(2 pts) How many processes are running? Which is which (refer to the if/else block)?

(6 pts) Trace the steps the message takes before printing to the screen, from the array `msg` to the array `inbuff`, and identify which process is doing each step.

(2 pts) Why is there a sleep statement? What would be a better statement to use instead of sleep for this small example?

3.7 Shared Memory

Download the example programs `shm_server.c` and `shm_client.c`. Read man pages for the following commands.

- Shared Memory – shmget, shmctl, shmat, shmdt

Compile and run `shm_server.c` and `shm_client.c`. **Be sure to start the server program prior to the client.**

(2 pts) How do the separate processes locate the same memory space?

(3 pts) There is a major flaw in these programs, what is it? (Hint: Think about the concerns we had with threads)

(3 pts) Now run the client without the server. What do you observe? Why?

(3 pts) Now add the following two lines to the server program just before the exit at the end of main:

```
shmdt(shm);
shmctl(shmid, IPC_RMID, 0);
```

Recompile the server. First run the server and client together. Then run the client without the server.

What do you observe? What did the two added lines do? (HINT: look at the man pages of shmdt and shmctl; look for the explanation of IPC_RMID in the shmctl man pages)

3.8 Message Queues and Semaphores

This section of the lab does not require any experiments. This section simply explores some of the other IPC mechanisms available to a programmer. For more in depth exploration of IPC mechanisms please see "Inter-process Communications in UNIX: The Nooks and Crannies" (2nd Edition), by John Gray.

Read man pages for the following commands to answer questions in this section.

- IPC overview - svipc
- Message Queues – msgget, msgctl, msgsnd, msgrcv
- Semaphores – semget, semctl, semop

Include answers to the following questions in your report:

(2 pts) Read the man pages of msgsnd. Message queues allow for programs to synchronize their operations as well as transfer data. How much data can be sent in a single message using this mechanism?

(2 pts) Read the man pages of msgrcv. What will happen to a process that tries to read from a message queue that has no messages (hint: there are more than one cases)?

(2 pts) Both Message Queues and Shared Memory create semi-permanent structures that are owned by the operating system (not owned by an individual process). Although not permanent like a file, they can remain on the system after a process exits. Describe how and when these structures can be destroyed.

(2 pts) Are the semaphores in Linux general or binary? Describe in brief how to acquire and initialize them.