Operating Systems Lab

CPE 435-01

Lab 08: Signals

By: David Thornton

Lab Date: 1 March 2021

Lab Due: 9 March 2021

Demonstration Due: 9 March 2021

Introduction

The purpose of this lab is to give students an introduction to signals in Linux.

Theory

Topic 1: Discuss each inter-process communication methods we have used in this class until now.

-

Topic 2: Discuss any 3 methods of inter-process communication. Describe each of them.

-

Observations

All code performs as expected.

```
dht0002@DAVID-PC: ~/CPE435/Lab08/Code
dht0002@DAVID-PC:~/CPE435/Lab08/Code$ gcc question1.c -o q1 -Wall
dht0002@DAVID-PC:~/CPE435/Lab08/Code$ ./q1
^C
Detected SIGINT in process 5692. System is still protected!
^C
Detected SIGINT in process 5692. System is still protected!
^C
Detected SIGINT in process 5692. System is still protected!
Program can now be terminated...
^CReceived kill signal 2
Killing child process...
Requested termination from parent process
        Dying process 5693
Child process killed
        Dying process 5692
```

Figure 1: Regular output of question1.c

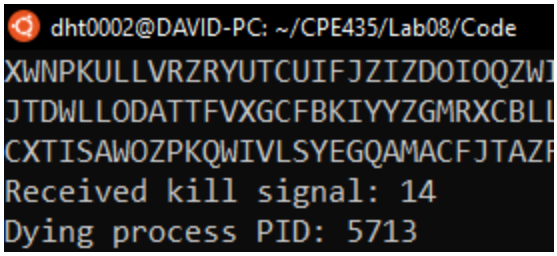

Figure 2: Regular output of question2.c

Conclusion

This lab was successful in expanding my knowledge of signals. [Demo link]

Appendix

Appendix 1:question1.c

```
// ****************************************
// Program Title: Lab 08
// Project File: question1.c
// Name: David Thornton
// Course Section: CPE-435, SP 2021
// Due Date: 03/09/2021
// ****************************************

#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

pid_t pid = 0;
void kill_func(int killSignal);
void child_kill_func(int killSignal);
void myFunction(int sigVal);
void printProtect(int sigVal);

int main()
{
	pid = fork();
	if(pid == 0) // child
	{
		signal(SIGINT, printProtect);
```

```
                signal(SIGTERM, child_kill_func);
                while(1);
        }
        else // parent
        {
                signal(SIGINT, printProtect);
                signal(SIGALRM, myFunction);
                alarm(10);
                while(1);
        }
        while(1);
        return(0);
}


void kill_func(int killSignal)
{
        printf("Received kill signal %d\n", killSignal);
        printf("Killing child process...\n");
        kill(pid, SIGTERM);
        wait(0);
        printf("Child process killed\n");
        printf("\tDying process %d\n", getpid());
        exit(0);
}


void child_kill_func(int killSignal)
{
        printf("Requested termination from parent process\n");
        printf("\tDying process %d\n", getpid());
        exit(0);
}


void myFunction(int sigVal)
{
        printf("Program can now be terminated...\n");
        signal(SIGINT, kill_func);
}


void printProtect(int sigVal)
{
        if(pid == 0) // child
        {
                return;
```

```
        }
        else if (pid > 0)
        {
                printf("\nDetected SIGINT in process %d. System is still protected!\n", getpid());
        }
}
```

Appendix 2:question2.c

```
// ****************************************
// Program Title: Lab 08
// Project File: question2.c
// Name: David Thornton
// Course Section: CPE-435, SP 2021
// Due Date: 03/09/2021
// ****************************************

#define ALARM_LENGTH 10

#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>

void kill_func(int killSignal);
void timeBomb(int sigVal);

int main()
{
        printf("Doing nothing\n");
        signal(SIGINT, timeBomb);     // when SIGINT is recieved, call timeBomb
        while(1);
        return -1;
}


void timeBomb(int sigVal)
{
        printf("\nReceived signal: %d\n", sigVal);
        signal(SIGALRM, kill_func);   // when SIGALRM is recieved, call kill_func
        alarm(ALARM_LENGTH);                          // set alarm
        printf("Setting alarm for %d second(s).\n", ALARM_LENGTH);
        while(1)
        {
                char randomchar = 'A' + (random() % 26);
```

```
                printf("%c", randomchar); // print random char
        }
        printf("\nEnd of alarm, exiting program...\n");
        exit(1);
}


void kill_func(int killSignal)
{
        printf("\nReceived kill signal: %d\n", killSignal);
        printf("Dying process PID: %d\n", getpid());
        exit(0); // kills the process that made the function call
}
```