Andrew Ma

Lab 8

CPE 435

3/9/21

Theory

We have studied 5 interprocess communication methods so far: pipes, shared memory, semaphores, message queues, and signals.

3 Interprocess Communication Methods

1) Pipes

Pipes are a one to one mechanism, where only one process talks to another process. There is automatic synchronization in pipes, but it requires intermediate copying of data in pipes by writing to one end and reading from the other end.

2) Shared Memory

Shared memory are segments of memory that are shared between processes. There can be more than just one to one processes, and several processes can attach themselves to the shared memory segment with local pointers to the shared data structure. There is no automatic synchronization in shared memory unlike pipes, so there usually is a flag in the shared data structure and each process can take turns based on the flag.

3) Message Queues

Message queues can be created and different processes can add messages and read messages from the message queues. Message queues have synchronization provided so there doesn't need to be a flag variable like shared memory, and we can configure a message queue to be blocking and let a process wait until there is a valid message. We can also configure processes to use message queues asynchronously and not block until there is a valid message but return immediately.

void (*signal(int sig, void (*func)(int)))(int)

This is part of signal.h. The signal() function chooses one of 3 ways to handle signals with the argument signal number. If the func argument is SIG_DFL, the default handler for the signal runs. If the func argument is SIG_IGN, the signal is ignored. If the func argument points to a signal handler function, then that signal handler function is called.

unsigned int alarm(unsigned int seconds)

This function is part of unistd.h. The alarm() function causes a SIGALRM signal to be sent to the calling process after that many seconds. If the argument is 0, then any pending alarm is cancelled. It returns the number of seconds remaining until any previously scheduled alarm was due to be delivered, and 0 if there was no previously scheduled alarm.

int kill(pid_t pid, int sig)

This function is part of signal.h. The kill() system call can be used to send any signal to any process. If the pid is positive, then the signal is sent to the process with that pid. If the pid is 0, then the signal is sent to every process in the process group of the calling process. If the pid is -1 then the signal is sent to every process that the calling process has permission to send signals except for the process with PID 1. If the pid is less than -1, the signal is sent to every process in the process group whose ID is -pid. The kill() function returns 0 on success and -1 if there is an error.

int pause(void)

This function is part of unistd.h. It suspends the calling thread until a signal is received. The signal received can execute a signal handler function or terminate the process. If the signal received is to terminate the process, then pause() function will not return. If the signal received is to execute a signal handler function, pause() will return after the signal handler function returns.

Observations

Assignment 1

Before 10 seconds has passed and pressing Ctrl+C

```
x@x:~/Desktop/lab08$ gcc assignment1.c -o assignment1 && ./assignment1
^CSystem is protected. I am the parent [37583] and I can't be killed yet
I am the child [37584], and I can't be killed yet
^CSystem is protected. I am the parent [37583] and I can't be killed yet
I am the child [37584], and I can't be killed yet
^CSystem is protected. I am the parent [37583] and I can't be killed yet
I am the child [37584], and I can't be killed yet
^CSystem is protected. I am the parent [37583] and I can't be killed yet
I am the child [37584], and I can't be killed yet
```

After 10 seconds has passed

```
x@x:~/Desktop/lab08$ gcc assignment1.c -o assignment1 && ./assignment1
^CSystem is protected. I am the parent [37583] and I can't be killed yet
I am the child [37584], and I can't be killed yet
^CSystem is protected. I am the parent [37583] and I can't be killed yet
I am the child [37584], and I can't be killed yet
^CSystem is protected. I am the parent [37583] and I can't be killed yet
I am the child [37584], and I can't be killed yet
^CSystem is protected. I am the parent [37583] and I can't be killed yet
I am the child [37584], and I can't be killed yet
Passed 10 seconds
```

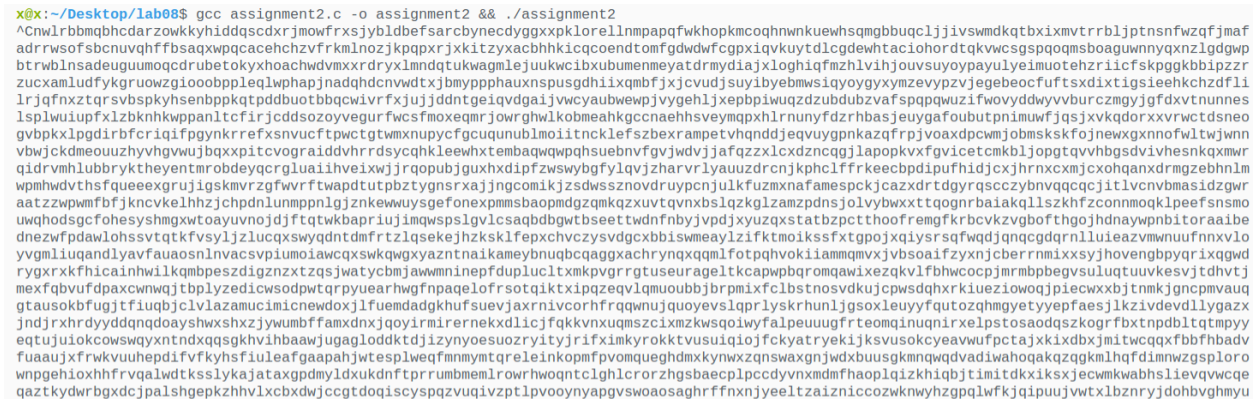
After 10 seconds has passed and Ctrl+C

```
x@x:~/Desktop/lab08$ gcc assignment1.c -o assignment1 && ./assignment1
^CSystem is protected. I am the parent [37583] and I can't be killed yet
I am the child [37584], and I can't be killed yet
^CSystem is protected. I am the parent [37583] and I can't be killed yet
I am the child [37584], and I can't be killed yet
^CSystem is protected. I am the parent [37583] and I can't be killed yet
I am the child [37584], and I can't be killed yet
^CSystem is protected. I am the parent [37583] and I can't be killed yet
I am the child [37584], and I can't be killed yet
Passed 10 seconds
^CParent trying to terminate the child process and has sent signal SIGUSR1
Child [37584] has received signal to terminate from parent
Goodbye

x@x:~/Desktop/lab08$
```

Assignment 2

Before 10 seconds and Ctrl+C



After 10 seconds and Ctrl+C

Conclusion

Yes the program worked as expected. From this lab I learned how to use custom signal handlers to handle default signals like SIGINT. This allows for custom behavior when I press Ctrl+C to not terminate but run custom code. Also, I learned how to use custom signals like SIGUSR1 and SIGUSR2, and I learned how

to send a custom signal to another process with the kill(pid, signal) function. I learned about various types of signals like SIGCHLD that is sent when a child terminates, and about SIGALRM that can be used in conjunction with the alarm(seconds) function to run code after X many seconds.

Source Code

```
# assignment1.c

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

pid_t child_pid = -1;
int const NUM_SECONDS_TIL_RESTORE = 10;

void child_sigint_msg_handler(int signum) {
    printf("I am the child [%d], and I can't be killed yet\n",
getpid());
}

void child_sigusr1_handler(int signum) {
    printf("Child [%d] has received signal to terminate from parent\n",
getpid());
    signal(SIGINT, SIG_DFL);
    kill(getpid(), SIGINT); // child kills itself
}

void child_ignore_sigint(int signum) {
    signal(SIGINT, SIG_IGN);
}
```

```c
void parent_sigint_msg_handler(int signum) {
    printf("System is protected. I am the parent [%d] and I can't be killed yet\n", getpid());
}


void parent_sigchld_handler(int signum) {
    // when child terminates, it sends SIGCHLD
    // so now parent can print goodbye message and terminate

    printf("%s", "Goodbye\n");
    signal(SIGINT, SIG_DFL);
    kill(getpid(), SIGINT); // parent kills itself
}


void parent_sigint_goodbye_handler(int signum) {
    signal(SIGCHLD, parent_sigchld_handler); // when child terminates, it sends SIGCHLD, so associate that with custom sigchld handler

    // send signal to child
    kill(child_pid, SIGUSR1);

    printf("%s", "Parent trying to terminate the child process and has sent signal SIGUSR1\n");

}

void restore_default_sigint(int signum) {
    printf("Passed %d seconds\n", NUM_SECONDS_TIL_RESTORE);

    // change parent's SIGINT handler to goodbye function
    signal(SIGINT, parent_sigint_goodbye_handler);
}
```

```
int main() {
    // create child process
    child_pid = fork();
    if (child_pid == 0) {
        // child process

        signal(SIGINT, child_sigint_msg_handler); // child catches
SIGINT signal and prints message
        signal(SIGUSR1, child_sigusr1_handler); // child catches SIGUSR1
signal and terminates

        // change Alarm signal handler to call function that will reset
the SIGINT signal handler to Ignore
        signal(SIGALRM, child_ignore_sigint);
        // Alarm after 10 seconds to change SIGINT handler to default
        alarm(NUM_SECONDS_TIL_RESTORE);

    } else {
        // parent process

        signal(SIGINT, parent_sigint_msg_handler); // parent catches
SIGINT signal (Ctrl+C) and prints message that system is protected

        // change Alarm signal handler to call function that will reset
the SIGINT signal handler back to default
        signal(SIGALRM, restore_default_sigint);
        // Alarm after 10 seconds to change SIGINT handler to default
        alarm(NUM_SECONDS_TIL_RESTORE);
    }


    while(1);
}
```

```
# assignment2.c
```

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

int const LENGTH = 26;

void sigint_handler(int signum) {
    // print page full of gibberish message
    char alphabet[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g',
                       'h', 'i', 'j', 'k', 'l', 'm', 'n',
                       'o', 'p', 'q', 'r', 's', 't', 'u',
                       'v', 'w', 'x', 'y', 'z'};
    for (int i=0; i < 10000; ++i) {
        printf("%c", alphabet[rand() % LENGTH]);
    }
    printf("%s", "\n");
}

void restore_sigint_handler(int signum) {
    signal(SIGINT, SIG_DFL);
}

int main() {
    signal(SIGINT, sigint_handler);

    signal(SIGALRM, restore_sigint_handler);
    alarm(10);

    while (1);
}
```