Austin Brown

CPE 434-01

3/10/2021

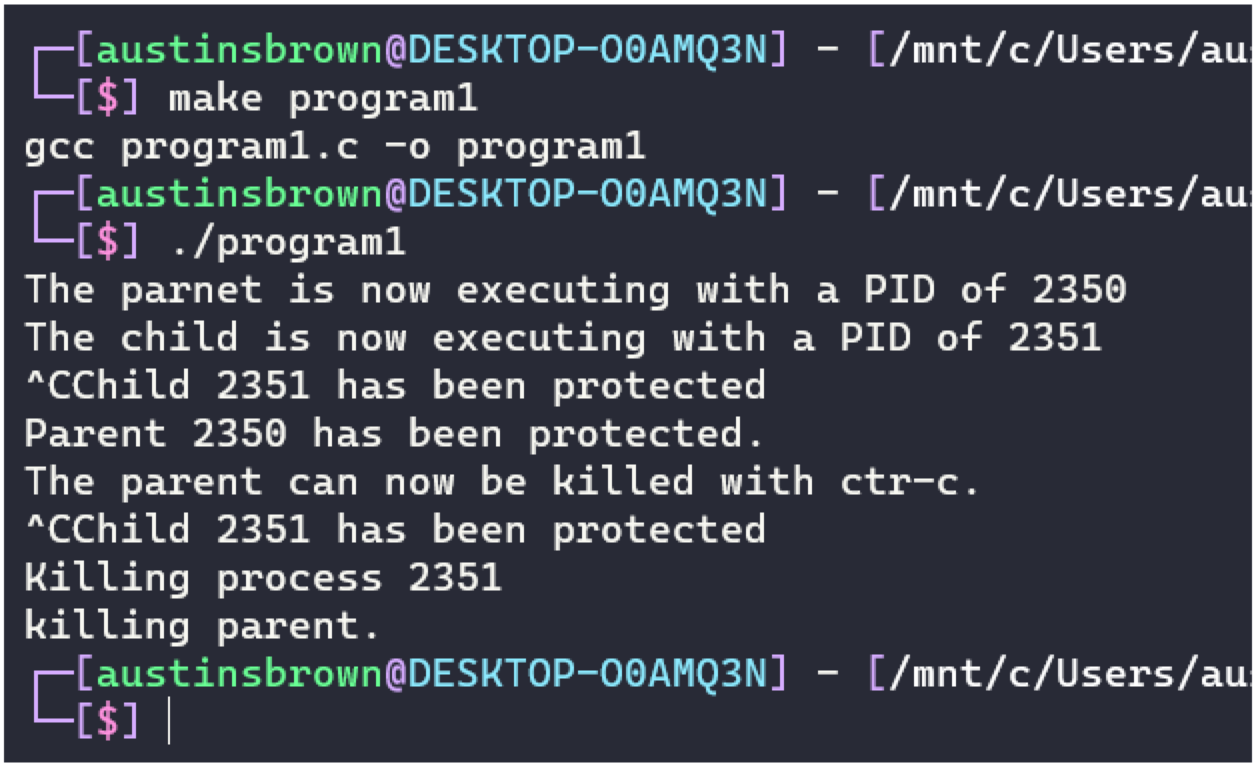
Lab 8

Theory

In this lab, we discuss another type of inter process communication. We first looked at shared memory. This is where we have a memory space that multiple processes can access. The problem with this is that the programmer must take great care to avoid synchronization issues. Another way is to use message queues. These have fewer problems than shared memory but require more overhead. There are also semaphores which are used to signal to other processes. In this lab, we use POSIX signals. They serve as notifications to processes. One example is the SIGINT signal. It means that an interrupt has occurred. SIGALRM means that an alarm has occurred. SIGKILL means that a kill command has been received. The signal function is used to modify a signals behavior. It takes the signum, and a handler as arguments.

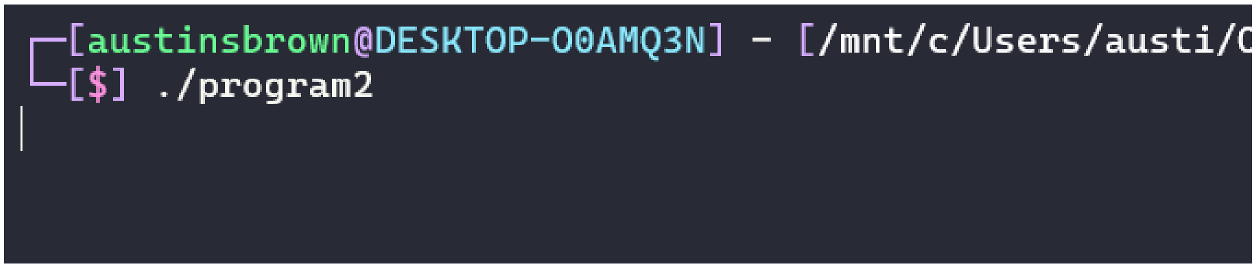
Observations

Results for Program 1

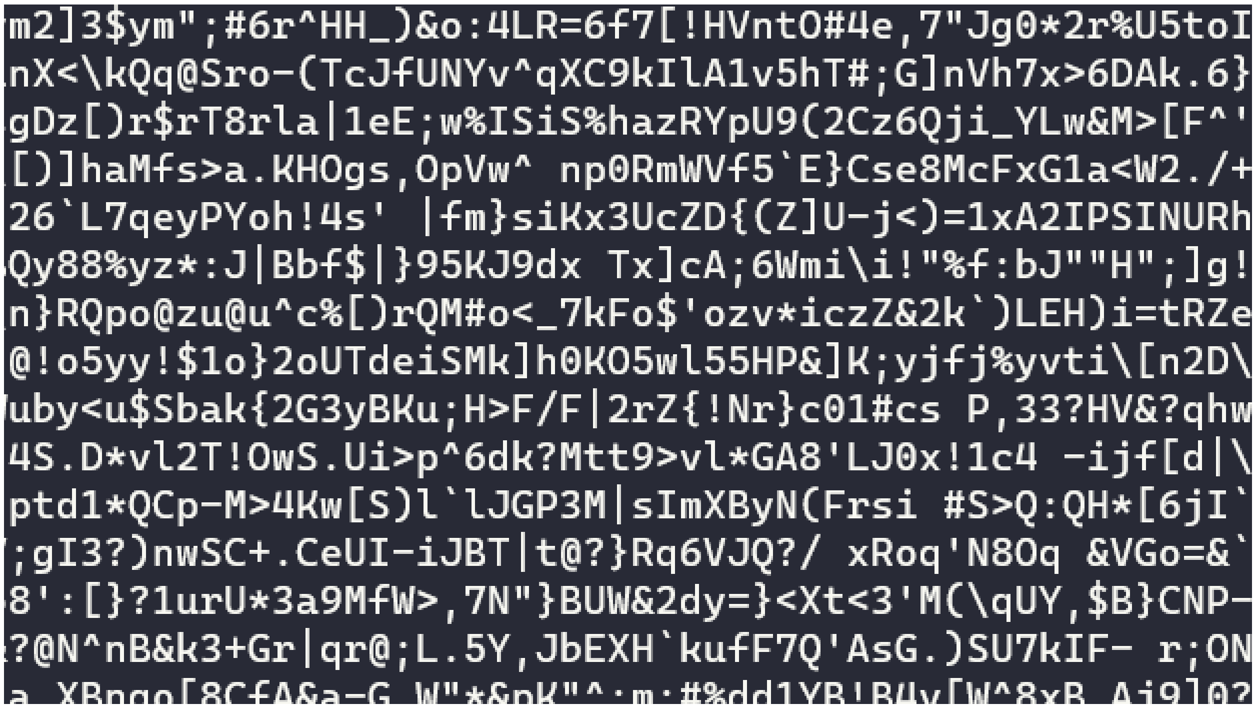


```
┌─[austinsbrown@DESKTOP-O0AMQ3N] - [/mnt/c/Users/au
└─[$] make program1
gcc program1.c -o program1
┌─[austinsbrown@DESKTOP-O0AMQ3N] - [/mnt/c/Users/au
└─[$] ./program1
The parnet is now executing with a PID of 2350
The child is now executing with a PID of 2351
^CChild 2351 has been protected
Parent 2350 has been protected.
The parent can now be killed with ctr-c.
^CChild 2351 has been protected
Killing process 2351
killing parent.
┌─[austinsbrown@DESKTOP-O0AMQ3N] - [/mnt/c/Users/au
└─[$] |
```

Results for Program 2



```
┌─[austinsbrown@DESKTOP-O0AMQ3N] - [/mnt/c/Users/austi/C
└─[$] ./program2
|
```



```
m2]3$ym";#6r^HH_)&o:4LR=6f7[!HVntO#4e,7"Jg0*2r%U5toI
nX<\kQq@Sro-(TcJfUNYv^qXC9kIlA1v5hT#;G]nVh7x>6DAk.6}
gDz[)r$rT8rla|1eE;w%ISiS%hazRYpU9(2Cz6Qji_YLw&M>[F^'
[)]haMfs>a.KHOgs,OpVw^ np0RmWVf5`E}Cse8McFxG1a<W2./+
26`L7qeyPYoh!4s' |fm}siKx3UcZD{(Z]U-j<)=1xA2IPSINURh
Qy88%yz*:J|Bbf$|}95KJ9dx Tx]cA;6Wmi\i!"%f:bJ""H";]g!
n}RQpo@zu@u^c%[)rQM#o<_7kFo$'ozv*iczZ&2k`)LEH)i=tRZe
@!o5yy!$1o}2oUTdeiSMk]h0KO5wl55HP&]K;yjfj%yvti\[n2D\
uby<u$Sbak{2G3yBKu;H>F/F|2rZ{!Nr}c01#cs P,33?HV&?qhw
4S.D*vl2T!OwS.Ui>p^6dk?Mtt9>vl*GA8'LJ0x!1c4 -ijf[d|\
ptd1*QCp-M>4Kw[S)l`lJGP3M|sImXByN(Frsi #S>Q:QH*[6jI`
;gI3?)nwSC+.CeUI-iJBT|t@?}Rq6VJQ?/ xRoq'N8Oq &VGo=&`
8':[}?1urU*3a9MfW>,7N"}BUW&2dy=}<Xt<3'M(\qUY,$B}CNP-
?@N^nB&k3+Gr|qr@;L.5Y,JbEXH`kufF7Q'AsG.)SU7kIF- r;ON
a XBnqo[8CfA&a-G W"*&pK"^:m:#%dd1YB!B4y[W^8xB Aj9]0?
```

Conclusion

Signals are yet another way to allow processes to communicate with each other. The signal function can be used to modify the way that certain signals behave. The only exception to this is certain signals such as SIGKILL cannot be modified. This implementation is asynchronous which allows for programs to be more robust. Overall, the code performed exactly as expected.

Appendix

Program 1

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

void protect(int);
void killChild(int);
void sendAlarm(int);
void killParent(int);

pid_t pid;

int main()
{
    pid = fork(); // create a child process

    if(pid == 0) // the child executes this
    {
        printf("The child is now executing with a PID of %d\n", getpid());
        signal(SIGINT, protect); // keeps ctr-c from working
        signal(SIGUSR1, killChild); // SIGUSR1 triggers kill child
    }
    else // the parent executes this
    {
        printf("The parnet is now executing with a PID of %d\n", getpid());
        signal(SIGINT, protect); // keeps ctr-c from working
        signal(SIGALRM, sendAlarm); // SIGALRM triggers the send alarm function
        alarm(10);
    }
```

```c
    for(;;);
    return 0;
}

void protect(int signal)
{
    if(pid == 0)
        printf("Child %d has been protected\n", getpid());
    else
        printf("Parent %d has been protected.\n", getpid());
}


void killChild(int signal)
{
    printf("Killing process %d\n", getpid());
    exit(0);
}


void sendAlarm(int sig)
{
    printf("The parent can now be killed with ctr-c.\n");
    signal(SIGINT, killParent);
}


void killParent(int sig)
{
    kill(pid, SIGUSR1);
    wait(0);
    printf("killing parent.\n");
    exit(0);
}
```

Program 2

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

void explode();
void escape();

int main()
{
    signal(SIGINT, explode);
    signal(SIGALRM, escape);
    for(;;);
}

void explode()
{
    srand(time(NULL)); // set up seed
    alarm(10);
    for(;;)
        printf("%c", (rand()%(126-
32))+32); // print random ascii stuff
}

void escape()
{
    printf("\n");
    exit(0);
}
```