Austin Brown

CPE 434-01

3/10/2021

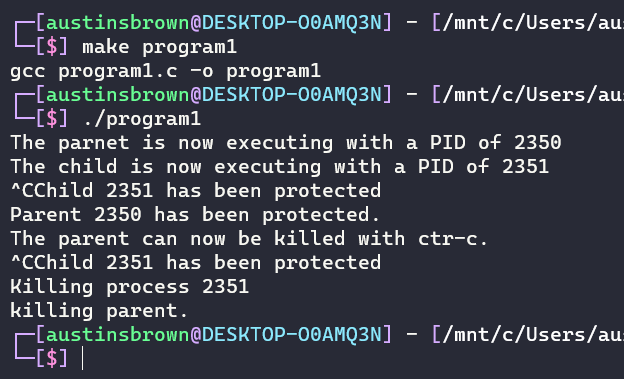
Lab 8

**Theory**

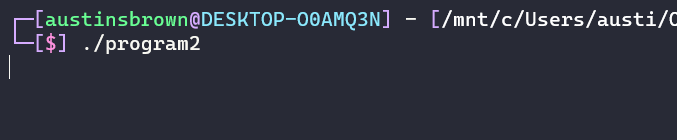
In this lab, we discuss another type of inter process communication. We first looked at shared memory. This is where we have a memory space that multiple processes can access. The problem with this is that the programmer must take great care to avoid synchronization issues. Another way is to use message 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



Results for Program 2





**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);      }      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

|  |
| --- |
| #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);  } |