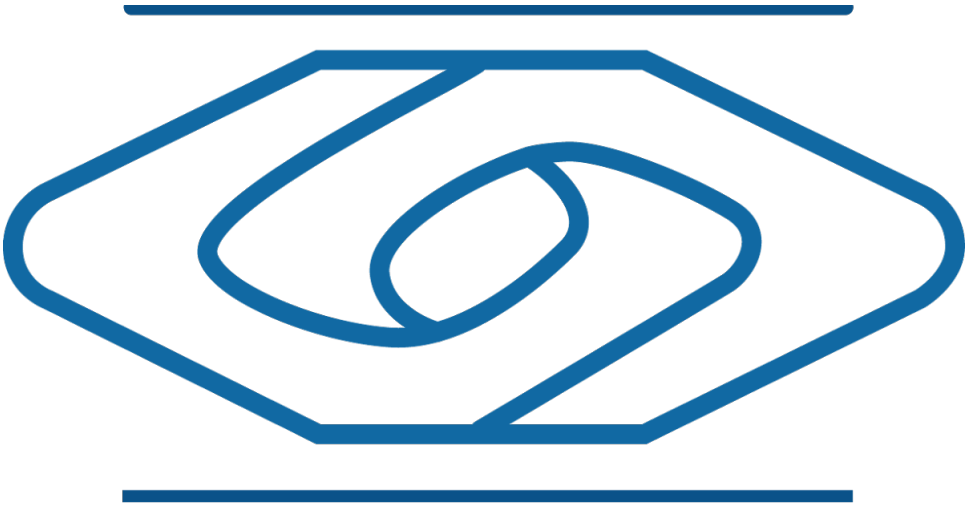


Instituto Politécnico Nacional

Escuela Superior de Computo

Práctica 3: Procesos e señales

Reporte

Profesor: Ulises Velez Saldaña

Alumno: Meza Madrid Raúl Damián

Clase: Sistemas operativos

Grupo: 2CM7

Contents

Introducción

En esta practica utilizaremos elementos como el pid, señales y el time stamp de Linux (función time) para crear 3 programas que terminaran de diferentes maneras.

1.1 Procesos

Una de las principales tareas de un sistema operativo es la de manejar los procesos que son ejecutados para así asignar a cada uno de ellos los recursos necesarios para su respectivo funcionamiento. A la hora de ejecutar un programa en un sistema Unix, el sistema crea un ambiente especial para ese programa. Este ambiente contiene todo lo necesario para que el sistema corra el programa como si ningún otro programa corriera dentro de ese sistema. Cada que el usuario corre un comando, Unix comienza un nuevo proceso. De manera simple, un proceso es una instancia de un programa en ejecución. El sistema operativo rastrea cada proceso a través de un numero ID de 5 dígitos, también conocido como pid (Process ID); este numero es único.[4]

1.2 Señales

Las señales son interrupciones del software enviadas a un programa para indicar que un evento importante ha ocurrido. Este evento puede variar de peticiones del usuario hasta accesos ilegales a memoria. Algunas señales, tales como las señales de interrupción, indican que el usuario ha solicitado que el programa haga algo que no esta dentro de su flujo de control normal. [3]El uso de señales permite que el usuario o otros procesos puedan interferir en la ejecución normal de un programa, obtener información.

1.3 Programas y herramientas utilizados

Esta práctica fue desarrollada en el sistema operativo Ubuntu 18.04.1 LTS. Estos son los programas y herramientas utilizados, junto con el comando de instalación, en caso de que no estuvieran instalados ya.

- Doxygen

```
git clone https://github.com/doxygen/doxygen.git
cd doxygen
mkdir build
cd build
cmake -G "Unix Makefiles" ..
make
make install
```

- make
- cmake

Objetivo

Que el alumno aplique la teoría vista en clase, visualizando los procesos las señales y la manera en la que un programa puede terminar; ya sea de manera normal, por una señal o de manera forzada.

Desarrollo

En esta practica se requiere documentar el código a través de el programa Doxygen, para lo cual primero se genero el archivo config usando el comando

```
doxygen -g config
```

Este archivo es muy largo para ser adjuntado, pero solo se procuro cambiar los parametros

- INPUT=src/
- OUTPUT_DIRECTORY=doc/
- RECURSIVE=Yes

En la práctica pasada se utilizo un script de bash para compilar y probar los programas. Esta vez se utilizara un archivo make (Makefile) para compilar, probar, generar la documentación según lo especificado en el archivo config y limpiar los archivos creados, tanto ejecutables como documentación.

```
programa1:
	bin/Prog1

programa2:
	bin/Prog2

programa3:
	bin/Prog3

comp:
	gcc -o bin/Prog1 src/Prog1.c
```

```
        gcc -o bin/Prog2 src/Prog2.c
        gcc -o bin/Prog3 src/Prog3.c

gendoc:
        doxygen config

clean:
        rm doc/* -rf
        rm bin/* -rf
        rm Makefile.* -rf
```

3.1 Primera parte

Hacer un programa en C que imprima en un ciclo infinito ”Aquí estoy” seguido de el ID del proceso.

3.1.1 Desarrollo

Para conseguir el pid del programa es necesario importar la funcion getpid() de la biblioteca unistd.h, después solo se imprime en formato de int, ya que la funcion getpid() regresa un tipo pid_t el cual es un signed integer.[2]

3.1.2 Codigo fuente

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

int main(int argc, char const *argv[]) {
  while (1) {
    printf("Aqui estoy %d \n", getpid());
  }
  return 0;
}

/**
@file Prog1.c
@author Damian Meza Madrid
@date 16 Marzo 2019
@brief  Programa que imprime el mensaje "Aqui estoy" seguido de el
 ↪  id del proceso de un ciclo infinito
*/
```

3.2 Segunda parte

Modificar le programa 1 para que termine después de 30 segundos e imprima ”Terminación normal”

3.2.1 Desarrollo

Para medir el tiempo de ejecución es necesario importar la funcion time() y difftime() Al inicio del programa se guarda el tiempo como startTime y despues de cada iteracion se revisa que difftime() entre time(NULL) (el tiempo actual) y startTime sea diferente a 30. difftime() regresa la diferencia entre el primer y el segundo argumento.

$$\tau Actual - \tau Inicial = \tau transcurrido \tag{1}$$

3.2.2 Código fuente

```c
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include <stdlib.h>
int main(int argc, char const *argv[]) {
  time_t startTime = time(NULL);
  do {
    printf("Aqui estoy %d \n", getpid());
  } while(30!=difftime(time(NULL),startTime));
  printf("Terminacion normal \n");
  return 0;
}
/**
@file Prog2.c
@author Damian Meza Madrid
@date 16 Marzo 2019
@brief  Programa que imprime el mensaje "Aqui estoy" seguido de el
 ↪  id del proceso durante 30 segundos.
*/
```

3.3 Tercera parte

Modificar el programa 1 para que atienda la señal del sistema operativo Ctrl+c. Terminar mostrando el mensaje ”Terminado por interrupción del sistema operativo”

3.3.1 Desarrollo

Esta vez es necesario detectar la señal del sistema. Para eso se usará primero una variable global declarada como static volatile de tipo sig_atomic_t una función sig_hand que se encargara de cambiar la variable global antes mencionada y la funcion signal() es la encargada de ejecutar la funcion sig_hand en caso de que la señal sea detectada. La manera en la que estas funciones funcionan es un poco complicada, ya que el manual de linux dice *The behavior of signal() varies across UNIX versions, and has also varied historically across different versions of Linux. Avoid its use.*

3.3.2 Código fuente

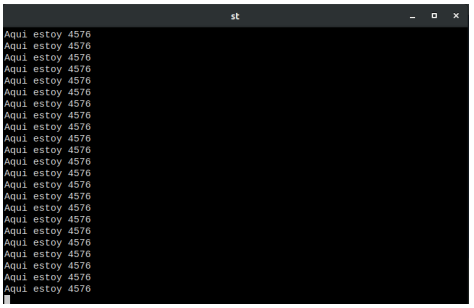
El siguiente código esta basado en la respuesta del usuario Dirk Eddelbuettel en Stack Oveflow [1]

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

static volatile sig_atomic_t sigCatch = 1;
static void sig_hand(int _){
  (void)_;
  sigCatch = 0;
}

int main(int argc, char const *argv[]) {

  signal(SIGINT,sig_hand);

  while(sigCatch) {
    printf("Aqui estoy %d \n", getpid());
  }

  printf("Terminado por interrupcion del sistema operativo\n");
  return 0;
}
/**
@file Prog3.c
@author Damian Meza Madrid
@date 16 Marzo 2019
```

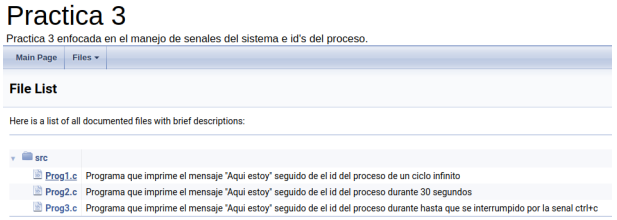
```
26 @brief Programa que imprime el mensaje "Aqui estoy" seguido de el
   ↪  id del proceso durante hasta que se interrumpido por la senal
   ↪  ctrl+c
27 */
```

3.3.3 Resultados

Los programas corrieron de la manera esperada. Se repetían por la duración definida



La generacion de la documentación fue fácil y nos da una pagina donde index.html como se ve a continuacion



3.4 Errores y problemas

Esta vez la practica resulto ser un tanto sencilla, solo fue necesario encontrar los comandos requeridos para la documentancion en Doxygen.

Codigo (Github)

Todo el codigo de esta practica se puede encontrar en :`https://github.com/asdf1234Damian/Operating-Systems/tree/master/Practica03`

References

[1] Dirk Eddelbuettel. re: Catch Ctrl-C in C. `https://stackoverflow.com/a/4217052/8524008`. [Online; consultado en 17 de marzo 2019].

[2] GNU.org. 26.3 Process Identification. http://www.gnu.org/software/libc/manual/html_node/Process-Identification.html. [Online; consultado en 17 de marzo 2019].

[3] tutorialspoint. Unix / Linux - Processes Management. https://www.tutorialspoint.com/unix/unix-processes.htm. [Online; consultado en 17 de marzo 2019].

[4] tutorialspoint. Unix / Linux - Signals and Traps. https://www.tutorialspoint.com/unix/unix-signals-traps.htm. [Online; consultado en 17 de marzo 2019].