INGENIERÍA UNIVERSIDAD DE LOS ANDES MÉRIDA VENEZUELA | DEPARTAMENTO DE COMPUTACIÓN ESCUELA DE SISTEMAS

# Laboratorio de Sistemas Operativos
# Semestre A-2016
# Práctica Dos
# Laboratorio

Prof. Rodolfo Sumoza
Prep. Alvaro Araujo

## 1. Señales.

El concepto de señal en sistemas operativos se refiere a una forma asíncrona (no se sabe cuándo se van a producir) de comunicarse con un proceso, este medio de comunicación es limitado, no se pueden enviar datos arbitrariamente. Una señal representa una notificación que es enviada a un proceso para notificarle la ocurrencia de un evento importante, el comportamiento que genera una señal en un proceso por lo general está predefinido, en algunos casos se puede asignar un nuevo comportamiento para manejar la señal en cuestión.

Cada señal tiene puede representarse con números enteros( `1,2,3, ...`), también se les puede asignar un nombre que las representa ( `SIGINT, SIGKILL, SIGSEGV, ...`). Con cualquiera de las dos representaciones se puede referir a una señal específica. Para listar las señales disponibles en un sistema LINUX, se ejecuta:

```
root@pc:/# kill -l
```

## 2. Funcionamiento de las señales.

Las señales poseen un manejador de señales (*signal handler*) el cual es una función que se invoca cuando el proceso recibe la señal. La llamada ocurre de manera asíncrona; es decir, en ningún fragmento del código fuente de la aplicación se realiza una llamada directamente a esta función. Cuando una señal

se envía a un proceso el sistema operativo detiene la ejecución del proceso y se da paso a la ejecución a la función manejadora de la señal, luego que finaliza la ejecución de esta función se reanuda la ejecución del proceso en el punto exacto donde se realizo la pausa (cambio de contexto).

## 3. Métodos para el envío de señales

Para enviar una señal a un proceso existen 3 métodos; los cuales son:

- Envió de una señal a través del teclado: Este es el método más común para los usuarios, éste consiste en presionar una combinación de teclas que el sistema operativo tiene predefinidas para generar el envío de una señal a un proceso con el cual esté interactuando. Ejemplo(**`Ctrl-c, Ctrl-z, Ctrl-\`**)

- Envío de una señal a través de la línea de comandos: En este método se realiza el envío usando un comando de la consola, el comando utilizado comúnmente es **`kill`**; este comando permite enviar una señal a un proceso determinado especificando su **PID** (identificador de proceso) y especificando el identificador de la señal (número o nombre). La sintaxis del comando es la siguiente:

```
user@pc:/$ kill -señal PID
```

- Envío de una señal a través de una llamada al sistema: Esta es la forma más común para enviar una señal de un proceso a otro, al igual que el comando **`kill`** de la consola se debe especificar el **PID** del proceso al cual va dirigida la señal y el identificador de la señal. Se puede utilizar la siguiente manera:

```
1   #include<unistd.h>
2   #include<sys/types.h>
3   #include<signal.h>
4   #include<stdio.h>
5
6   int main()
7   {
8     //Conozco mi PID
9     pid_t my_pid = getpid();
10
11    //Envio senial de parada
12    kill(my_pid, SIGSTOP);
13
14    while(1)
15      printf(".");
```

```
16
17              return 0;
18          }
```

Luego de ejecutar el código del ejemplo anterior pueden surgir algunas preguntas: ¿realmente terminó la ejecución del programa gracias a la señal **SIGSTOP**?, ¿cuál es la función por defecto de la señal **SIGSTOP**?, ¿qué funcionalidad tiene el comando de la consola `fg` (foreground)? y ¿qué diferencia hay entre la señal **SIGSTOP** y la señal **SIGINT**?

## 4. Captura de señales

Por lo general la mayoría de las señales pueden ser capturadas por un proceso, sin embargo existen señales las cuales no pueden ser capturadas. Un ejemplo de éstas es la señal SIGKILL(a la que se le asigna el número 9 en la mayoría de los sistemas LINUX). El comportamiento de esta señal es terminar inmediatamente el proceso.

Con el siguiente código se busca explorar cuáles señales del sistema operativo se pueden manejar.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <signal.h>
6
7  // -- Funcion manejadora de senales
8  void captura_signal(int);
9
10 int main()
11 {
12   int i=0;
13
14   /* Asignando manejador a c/u de las senales */
15   for(i=1;i<65;i++)
16     signal(i, captura_signal);
17
18   /* Auto envio de senales */
19   for(i=1;i<65;i++)
20   {
21     printf("- Senal Enviada Nro: %d\n", i);
22     fflush(stdout);
23     kill(mi_pid, i);
24   }
```

```
25
26    /* Espera por una senal */
27    pause();
28    printf(" ...Finalizado!!! \n");
29    return 0;
30  }
31
32  // ++ Funcion manejadora de senales
33  void captura_signal(int num_signal)
34  {
35    printf("   + Senal recibida numero: %d\n",
          num_signal);
36    fflush(stdout);
37  }
```

Cabe destacar que el hecho de que se pueda cambiar el comportamiento de una señal a través de una función manejadora no implica que sea recomendable hacerlo, por ejemplo a la señal **SIGSEGV** (enviada cuando ocurre una violación de segmento) se le puede asignar otro comportamiento; sin embargo, en caso de que no esté justificado cambiar el comportamiento, no se debería modificar. Las señales **SIGUSR1**, **SIGUSR2** no las utiliza el sistema operativo, las mismas son para uso del usuario.

Una señal puede ser capturada si se define una función manejadora y se le asigna a dicha señal. Esto se puede ilustrar con el siguiente ejemplo:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <signal.h>
6
7  // -- Funciones manejadoras de senales
8  void captura_c(int);
9  void captura_z(int);
10 void captura_bar(int);
11
12 int main()
13 {
14   int i=0;
15
16   /* Conociendo el numero de proceso asignado
17      por el SO */
18
19   pid_t mi_pid = getpid();
20   printf("\n\t Mi PID es: %d\n\n", mi_pid);
21
```

```
22    /* Asignando manejador a c/u de las senales */
23    signal(2, captura_c);   // SIGINT
24    signal(3, captura_bar); // SIGQUIT
25    signal(20, captura_z);  // SIGTSTOP
26
27
28
29    while(1)
30    {
31      printf(".");
32      fflush(stdout);
33      sleep(1);
34    }
35
36    printf(" ...Finalizado!!! \n");
37    return 0;
38  }
39
40  // ++ Funciones manejadoras de senales
41  void captura_c(int num_signal)
42  {
43    printf(" No terminare la ejecucion \n");
44    fflush(stdout);
45  }
46
47  void captura_z(int num_signal)
48  {
49    printf(" No pausare la ejecucion \n");
50    fflush(stdout);
51  }
52
53  void captura_bar(int num_signal)
54  {
55    printf(" Esto tampoco funcionara \n");
56    fflush(stdout);
57  }
```

## 5. Temporizadores

Un temporizador es esencialmente un mecanismo que permite medir el tiempo. Los temporizadores en los sistemas operativos permiten gestionar los tiempos de espera, esto es importante tanto para la interacción de un programa con un usuario (ejemplo, esperar 10 segundos para el ingreso de un dato), como también es importante para la interacción entre componentes del sistema operativo (ejemplo, cerrar una conexión transcurrido un tiempo).

El sistema operativo provee una manera sencilla para configurar temporizadores a través del uso de señales de alarmas,sin embargo, sólo se puede usar un contador de tiempo a la vez; aunque debería ser suficiente un solo temporizador en el caso de un programa no tan complejo.

Se puede implementar un temporizador para esperar un máximo de 10 segundos por un dato del usuario, de la siguiente manera:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <signal.h>
5
6  void alarma_accion(int sig_num)
7  {
8          printf("Tiempo expirado. Saliendo...\n\n");
9          exit(0);
10 }
11
12 int main()
13 {
14         char user[20];
15
16         signal(SIGALRM, alarma_accion);
17
18         printf("\n\t Nombre de Usuario: ");
19         fflush(stdout);
20
21   // Inicio de alarma
22         alarm(10);
23         fgets(user,10,stdin);
24
25   // Fin de alarma
26         alarm(0);
27
28         printf("\n\t Usuario: %s\n\n", user);
29
30   return 0;
31 }
```

¿Qué sucede si se quieren utilizar dos temporizadores simultáneamente o un temporizador un poco más preciso (del orden de los nanosegundos)?

Si se quiere implementar algo como lo planteado en la pregunta anterior se puede usar el tipo de temporizador que ofrece la **API POSIX**. Las principales cualidades que posee este temporizador son: su alta precisión y la posibilidad de tener varios temporizadores corriendo a la vez.

Para estudiar el uso de estos temporizadores observemos el siguiente ejemplo:

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  #include <signal.h>
6  #include <sys/time.h>
7  #include <time.h>
8
9  void manejador_alarma(int);
10 void texto_letra(char *);
11
12 int main() {
13
14   // temporizador 1
15   struct itimerspec tempo1;
16
17   // temporizador 2
18   struct itimerspec tempo2;
19
20   timer_t reloj1;
21   timer_t reloj2;
22
23   // Inicializacion relojes
24   timer_create(CLOCK_REALTIME, 0, &reloj1);
25   timer_create(CLOCK_REALTIME, 0, &reloj2);
26
27   signal(SIGALRM, manejador_alarma);
28
29   // Duracion de la alarma Tempo 1
30   tempo1.it_value.tv_sec = 60;
31   tempo1.it_value.tv_nsec = 000000000;
32
33   // Repeticion de la alarma Tempo 1
34   tempo1.it_interval.tv_sec = 0;
35   tempo1.it_interval.tv_nsec = 000000000;
36
37   // Activacion Tempo 1
38   timer_settime(reloj1, 0, &tempo1, NULL);
39
40   system("clear");
41   char buffer[1000];
42   char opcion = 'n';
43
44   strcpy(buffer, "\n\t Este es un mensaje de la
```

```
        Agencia Nacional U571-A. \n\t Este Mensaje se
        autodestruira en 60 segundos a partir del inicio
        de su ejecucion. \n\t Si elige conocer la mision
        tendra solo 5 segundos para aceptarla. \n\t
        Conocer la mision puede comprometer su vida. \n\t
         Suerte...! \n\n\n\t    Desea conocer la Mision?
        [s/n]: \0");
45
46    texto_letra(buffer);
47    scanf("%c",&opcion);
48
49    if(opcion == 's')
50    {
51      strcpy(buffer, "\n\t      Su mision consiste en
           encontrar las dos unicas soluciones para la
           ecuacion:\n\n\t        a^n + b^n = z^n  \n\n\t
                La combinacion de los numeros resultantes
           conforman las coordenadas donde \n\t      se
           encuentra escondido un objeto que compromete la
            seguridad nacional. \n\n\n\t      Acepta la
           Mision? [s/n]: \0");
52
53      texto_letra(buffer);
54      getchar();
55
56      // Duracion de la alarma Tempo 2
57      tempo2.it_value.tv_sec = 5;
58      tempo2.it_value.tv_nsec = 000000000;
59
60      // Repeticion de la alarma Tempo 2
61      tempo2.it_interval.tv_sec = 0;
62      tempo2.it_interval.tv_nsec = 000000000;
63
64      // Activacion Tempo 2
65      timer_settime(reloj2, 0, &tempo2, NULL);
66
67      scanf("%c",&opcion);
68
69      if(opcion == 's')
70      {
71        strcpy(buffer, "\n\t Gracias! Tu nacion lo
             agradecera ... autodestruccion! \n\n");
72        texto_letra(buffer);
73        system("clear");
74      }
75      else
```

```
76      {
77        strcpy(buffer, "\n\t Le has fallado a tu nacion
              ... autodestruccion! \n\n");
78        texto_letra(buffer);
79        system("clear");
80      }
81    }
82    else
83    {
84      strcpy(buffer, "\n\t Le has fallado a tu nacion
            ... autodestruccion! \n\n");
85      texto_letra(buffer);
86      system("clear");
87    }
88
89    timer_delete(reloj1);
90    timer_delete(reloj2);
91
92    return(0);
93  }
94
95  void texto_letra(char *buffer)
96  {
97    int i=0;
98    while(buffer[i] != '\0')
99    {
100     printf("%c", buffer[i]);
101     fflush(stdout);
102     i++;
103     usleep(50000);
104   }
105 }
106
107 void manejador_alarma(int sig_num)
108 {
109   char buffer[1000];
110   strcpy(buffer, "\n\t Tiempo expirado. Le has fallado
          a tu nacion ... autodestruccion! \n\n");
111   texto_letra(buffer);
112   system("clear");
113   exit(0);
114 }
```

```
user@pc:/$ gcc -lrt tempo_poxis.c -o tempo_poxis.out
```

## 6. Recomendaciones y Observaciones

- Las señales no se acumulan para los procesos. Es decir, si se recibe una señal y, sin haberse atendido llega otra señal del mismo tipo, cuenta como si hubiese llegado una sola señal.
- No se debe pensar que la llamada al sistema ***alarm*** hace que un proceso se detenga durante el tiempo especificado. Esta llamada lo que hace es notificar al sistema operativo para que se envíe una señal **SIGALRM** al proceso en cuestión luego de transcurridos los segundos especificados.
- Las funciones manejadoras de señales deben ser cortas y concretas, no se deben realizar grandes cálculos en estas.
- No se debe cambiar el comportamiento que genera una señal en un programa a menos que se tenga conciencia de lo que esto implica y que la señal involucrada no sea de vital importancia para el buen funcionamiento del programa.

## 7. Ejercicio

Realice un programa que al ser ejecutado 2 veces emule un juego de ping pong, luego de recibir la pelota uno de los procesos deberá esperar 2 segundos para devolver la pelota. Cabe destacar que alguno de los dos procesos debe realizar el saque, es decir comenzar la partida. El partido debe culminar luego de 7 toques de cada proceso.