# Programación servicios y procesos

Sincronización entre procesos

# Sincronización

Utilizaremos señales para llevar la comunicación entre procesos.

Una señal es como un aviso que un proceso manda a otro proceso y le específica función a ejecutar.

Las señales están implementadas mediante interrupciones por software. Cuando un proceso recibe una señal deja la ejecución de su código para atender a la señal. Tras esto, vuelve al lugar del proceso y locontinúaa

# Señales

Toda señal se identifica con un entero:

```
#define SIGKILL         9
#define SIGUSR1         10
#define SIGSEGV         11
#define SIGUSR2         12
...
```

Como norma general, toda señal tiene un significado específico que se traduce en una cierta acción a realizar en caso de ser recibida.

# Respuesta a señal

Ante una señal, un proceso puede reaccionar de tres formas:

- Ignorando la señal recibida.
- Invocando a una función de manejo por defecto (handler).
- Invocando a una función establecida por el programador con un tratamiento específico a realizar.

# Signal

La llamada *signal* nos permite registrar una función manejadora específica para cualquier señal, de manera que podemos establecer el tratamiento a realizar en caso de que una cierta señal sea recibida .

Signal establece que función manejadora (handler) llamaremos al recibir cierta señal.

# Descripción señales

- SIGUSR1

Es la señal definida por usuario número 1. No tiene un significado especial y puede ser empleada con cualquier propósito.

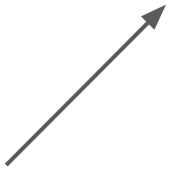
- SIGKILL

Es la señal de terminación de un proceso abrupta

….

# Signal

```
#include <signal.h>
*signal(int Señal,void (*Func)(int));
```



Nº señal que queremos capturar.

Función a la que queremos llamar

Ejemplo:

```
signal(SIGUSR1,gestion_padre);
//cuando el proceso reciba señal SIGUSR1 se realizará
//llamada a gestion_padre()
```

# Enviar señal:kill()

Para enviar señal (signal establece handler no envia) utilizaremos función KILL()

```
#include <stdio.h>
int kill (int Pid, int Señal);
```


Ejemplo

```
kill(pid_padre,SIGUSR1)
//envía señal SIGUSR1 a pid_padre
```

# Función pause()

Si queremos proceso espere a que le llegue señal utilizamos pause()

int pause (void);

Función suspende al proceso que realiza la llamada la cantidad de segundos indicada o hasta que reciba señal.

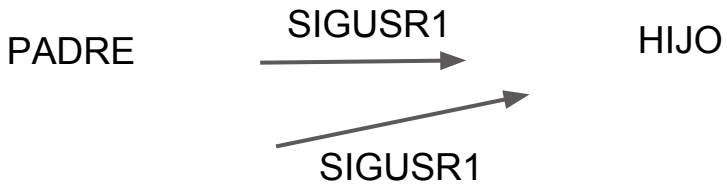
#include <unistd.h>

unsiged int sleep(unsigned int seconds);

# Ejemplo

Queremos un programa que realice lo siguiente

Establecemos handler para SIGUSR1 en proceso hijo
Proceso Padre envía 2 señales SIGUSR1  a hijo



# sincronizar-1.c

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <fcntl.h>

/*-------------------------------------------------------------*/
/*gestion de señales con proceso hijo (handler)*/
void manejador (int segnal)
{
    printf ("Hijo recibe señal...%d\n",segnal);
   }


/*-------------------------------------------------------------*/
```

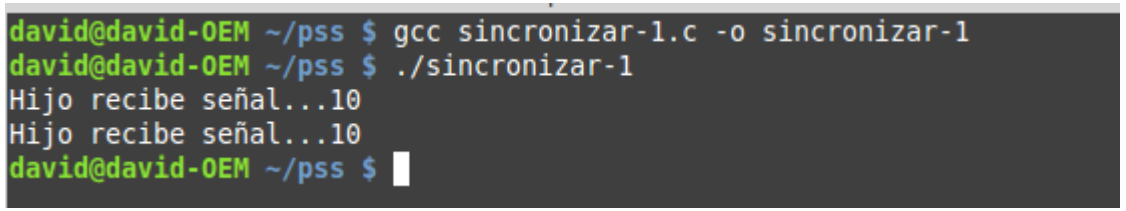
# sincronizar-1.c

```
int main()
{
    int pid_hijo;
    pid_hijo=fork(); //creamos hijo
    switch (pid_hijo)
     {
     case -1:
            printf ("ERROR al crear hijo.. \n");
            exit(-1);
    case 0:  //hijo
            signal (SIGUSR1,manejador); // establezco manejador señal en hijo
            while(1){
                    }; //bucle infinito dejamos hijo vivo para pueda recibir señal
            break;
```

# sincronizar-1.c

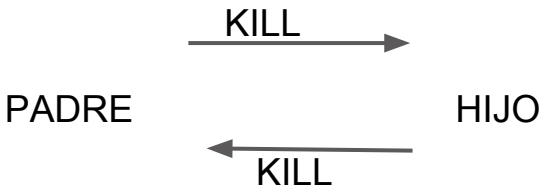
```
    default:
                sleep(1);
                kill(pid_hijo,SIGUSR1); //envia señal a hijo
                sleep(1);
                kill(pid_hijo,SIGUSR1); //envia señal a hijo
                sleep(1);
        break;
        }
    return 0;
}
```

Vemos ejecución



# Ejemplo2

Vamos a realizar un ejemplo en que definiremos un manejador para el padre y otro para el hijo y estableceremos el envío de señales entre ellos de forma continua y por lo tanto su ejecución.



# sincronizar (código texto a posteriori)

```c
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <fcntl.h>

/*-----------------------------------------------------------------*/
/*gestion de señales en proceso padre (handler)*/
void gestion_padre (int segnal)
{
    printf ("Padre recibe señal...%d\n",segnal);
}

/*gestion de señales en proceso hijo (handler)*/
void gestion_hijo (int segnal)
{
    printf ("Hijo recibe señal...%d\n",segnal);
}
/*-----------------------------------------------------------------*/

int main(){
    int pid_padre,pid_hijo;
    pid_padre=getpid();
    pid_hijo=fork(); //se crea al hijo

    switch(pid_hijo){
        case -1: //error
            printf("Error al crear proceso  hijo..\n");
            exit(-1);

        case 0: //hijo
            signal(SIGUSR1,gestion_hijo); //tratamiento señal en proceso hijo
            while (1) { //bucle infinito
                sleep(1);
                kill(pid_padre,SIGUSR1); //envia señal a padre
                }
            break;
```

# sincronizar (código texto a posteriori)

```
        default: //padre
            signal(SIGUSR1,gestion_padre); //tratamiento señal en proceso padre
            while (1) { //bucle infinito
                pause(); //padre espera hasta recibir señal hijo (se ejecutar primero)
                sleep(1);
                kill(pid_hijo,SIGUSR1); //envia señal a hijo
                }
            break;
        }//fin switch

return 0;
}//fin programa
```

# sincronizar.c

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <fcntl.h>
/*------------- Manejadores -----------------------------------------------*/
/*gestion de señales en proceso padre (handler)*/
void gestion_padre (int segnal)
{
    printf ("Padre recibe señal...%d\n",segnal);
}

/*gestion de señales en proceso hijo (handler)*/
void gestion_hijo (int segnal)
{
    printf ("Hijo recibe señal...%d\n",segnal);
}
/*-------------------------------------------------------------*/
```

# sincronizar.c

```
int main(){
 int pid_padre,pid_hijo;
 pid_padre=getpid();
 pid_hijo=fork(); //se crea al hijo

    switch(pid_hijo){
                case -1: //error
                         printf("Error al crear proceso  hijo..\n");
                 exit(-1);

                 case 0: //hijo
                        signal(SIGUSR1,gestion_hijo); //tratamiento señal en proceso hijo
                         while (1) { //bucle infinito
                                  sleep(1);
                                 kill(pid_padre,SIGUSR1); //envia señal a padre
                        }
                         break;
```

# sincronizar.c

```
default: //padre
        signal(SIGUSR1,gestion_padre); //tratamiento señal en proceso padre
        while (1) { //bucle infinito
                pause(); //padre espera hasta recibir señal hijo (se ejecutara primero)
                sleep(1);
                kill(pid_hijo,SIGUSR1); //envia señal a hijo
                }
        break;
 }//fin switch

return 0;

}//fin programa
```

# sincronizar

Al ejecutar procesos no se detienen utilizamos (ctrl+c).

```
david@david-OEM ~/pss $ gcc sincronizar.c -o sincronizar
david@david-OEM ~/pss $ ./sincronizar
Padre recibe señal...10
Padre recibe señal...10
Hijo recibe señal...10
Padre recibe señal...10
Padre recibe señal...10
Hijo recibe señal...10
Padre recibe señal...10
^C
```

# Práctica 5

Realiza un programa en C en donde un hijo envíe 3 señales SIGUSR1 a su padre que provoquen un manejador(handler) que escribirá el mensaje “soy el manejador del padre” y después envíe una señal SIGKILL para que el proceso padre termine.