TP IV – Señales - Signals

Objetivo: Desarrollar una serie de programas que permitan comprobar distintas funcionalidades vinculadas con la gestión señales y su aplicación.

Introducción

Puede acceder on-line al manual de referencia de la libreria GNU C, consultar allí el capítulo 24 Señales (Signals), esta disponible en CD-ROM de programación entregado en esta asignatura, a partir de la carpeta `/Software/Win32/gcc/docs/libc.pdf` .

1. Desarrolle un programa que capture las señales `SIGUSR1` y `SIGUSR2`. Puede usar una misma función para manipular ambas señales, de la siguiente forma:

```
signal(SIGUSR1, sig_usr);
signal(SIGUSR2, sig_usr);
```

en este caso, la función que manipulara ambas señales es `sig_usr()`, cuyo prototipo es:

```
void sig_usr(int signo);
```

donde `signo` corresponde la numero de señal que se ha generado y se ha enviado a este proceso. La captura de la señal puede fallar, en tal caso, la función `signal()` devolverá `SIG_ERR`, el programa deberá controlar eso y emitir un mensaje de error en caso de no poder capturar las señales. Luego de capturar las señales, el programa entrará en un loop infinito y dentro del mismo pondrá un mensaje indicativo de lo que esta haciendo y llamará a la función `pause()`:

```
for ( ; ; ) {
   printf("main():ingreso en pause()\n");
   pause();
}
```

Dentro de la función `sig_usr()` emita un mensaje por cada señal que recibe indicando si se trata de `SIGUSR1` o `SIGUSR2` o cualquier otra señal que llegue; imprima también el numero de la señal capturada.
Compile el programa como `tp41` (`tp41.c`).

2. Ejecute el programa desde el shell en background, de la siguiente forma: `./tp4 &`
Obtendrá una salida como esta:

```
grchere@debian:~/gccwork/src/signals$ ./tp41 &
[1] 2905
main():ingreso en pause()
grchere@debian:~/gccwork/src/signals$
```

donde `2905` (en este caso) se corresponde con el id de proceso asignado por el kernel. Observe que, a pesar de ser un proceso ejecutado en background este shell permite que los datos enviados a stdout por parte de un proceso en background se impriman en la consola, por ello vemos el `printf()` que pusimos dentro del loop infinito.
Ejecute el comando `ps -l` para ver los procesos que su usuario tiene en ejecucion:

```
grchere@debian:~/gccwork/src/signals$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  2709  2707  0  80   0 -  1428 -      pts/0    00:00:01 bash
0 S  1000  2905  2709  0  80   0 -   403 -      pts/0    00:00:00 tp41
0 R  1000  2916  2709  0  80   0 -   872 -      pts/0    00:00:00 ps
grchere@debian:~/gccwork/src/signals$
```

puede observar que `tp41` es un proceso hijo de bash (el shell que lo lanzó), obviamente bash hace un `fork()` de `tp41` y lo pone en background, tal como Ud. debió hacerlo en su propia versión de shell.

3. Utilice el comando `kill` para enviar una señal al proceso `tp41`, de la siguiente forma:

```
grchere@debian:~/gccwork/src/signals$ kill -SIGUSR1 2905
grchere@debian:~/gccwork/src/signals$ sig_usr():recibi SIGUSR1 (10)
main():ingreso en pause()
grchere@debian:~/gccwork/src/signals$
```

Puede comprobar que el kernel generó la señal `10` (`SIGUSR1`), la envió al proceso `2905` (`tp41`), el proceso capturó la señal a través de la función `sig_usr()`, se ejecutó `sig_usr()`, terminó `sig_usr()`, terminó la función `pause()` (provocó el retorno de la misma, puesto que, al momento de recibir la señal, `tp4` estaba ejecutando la función `pause()`). Según la bibliografía que tenemos, en este caso, la función `pause()` debería terminar retornando `-1` y seteando la variable global `errno` al valor `EINTR`.
Pruebe de enviar dos o más veces la señal `SIGUSR1` o `SIGUSR2`, si `tp4` continua respondiendo, esto implica que su sistema operativo no resetea las señales a su valor por defecto. Si `tp4` dejara de responder, implicaría que la captura de la señal sólo aplica a la primer ocurrencia de la misma y si deseamos continuar manipulándola, debemos volver a llamar a la función `signal()` dentro del manipulador para "re-manipularla". Modifique y recompile `tp41` si fuese necesario para que pueda responder a `n` ocurrencias de las señales `SIGUSR1` y `SIGUSR2`.

4. Para terminar un proceso, en realidad hay que enviar la señal de terminación (`SIGKILL`) a dicho proceso; ejecute el comando `kill` correspondiente:

```
grchere@debian:~/gccwork/src/signals$ kill -SIGKILL 2905
grchere@debian:~/gccwork/src/signals$
[1]+  Killed                  ./tp41
grchere@debian:~/gccwork/src/signals$
```

5. Copie `tp41.c` como `tp42.c`, modifique `tp42.c` para adaptarlo según lo dicho en el punto 3 (sólo si fuera necesario). Quite el mensaje que emite el programa en el loop infinito (antes de llamar a la función `pause()`), ahora cuando el programa reciba la señal `SIGUSR2` el programa deberá terminar normalmente (puede hacer esto modificando el loop infinito para que ahora trabaje con una variable global cuyo valor se modifica desde el manipulador de `SIGUSR2` y provoca la terminación normal de `main()`. El programa terminará devolviendo `0` al shell (`return 0`, `exit(0)`).
Compile y ejecute `tp42`.

6. Pruebe el correcto funcionamiento de `tp42` según lo indicado en el punto anterior: envíe la señal `SIGUSR1` varias veces y luego `SIGUSR2` y compruebe con el comando `ps` que el programa

termina su ejecución.

```
grchere@debian:~/gccwork/src/signals$ gcc -Wall -o tp42 tp42.c
grchere@debian:~/gccwork/src/signals$ ./tp42 &
[1] 2955
grchere@debian:~/gccwork/src/signals$ kill -SIGUSR1 2955
grchere@debian:~/gccwork/src/signals$ sig_usr():recibi SIGUSR1 (10)

grchere@debian:~/gccwork/src/signals$ kill -SIGUSR1 2955
grchere@debian:~/gccwork/src/signals$ sig_usr():recibi SIGUSR1 (10)

grchere@debian:~/gccwork/src/signals$ kill -SIGUSR2 2955
grchere@debian:~/gccwork/src/signals$
[1]+  Done                    ./tp42
grchere@debian:~/gccwork/src/signals$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  2709  2707  0  80   0 -  1428 -      pts/0    00:00:02 bash
0 R  1000  2956  2709  0  80   0 -   872 -      pts/0    00:00:00 ps
grchere@debian:~/gccwork/src/signals$
```

7. Copie `tp42.c` como `tp43.c`, modifique `tp43.c` acorde con el siguiente enunciado: La empresa X posee 10 sucursales, este proceso (`tp43`) acumula el total de ventas de estas sucursales (datos provenientes de otros sistemas que serán simulados), cada vez que este proceso recibe la señal `SIGUSR1` se muestran por pantalla los totales de ventas de cada una de las sucursales (los totales pueden ir cambiando, no necesariamente serán siempre los mismos: puede usar la función `random()` para generar números aleatorios).
Podría obtener una salida como esta:

```
grchere@debian:~/gccwork/src/signals$ ./tp43 &
[1] 2735
grchere@debian:~/gccwork/src/signals$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  2679  2677  0  80   0 -  1428 -      pts/0    00:00:00 bash
0 S  1000  2735  2679  0  80   0 -   402 -      pts/0    00:00:00 tp43
0 R  1000  2736  2679  0  80   0 -   872 -      pts/0    00:00:00 ps
grchere@debian:~/gccwork/src/signals$ kill -SIGUSR1 2735
grchere@debian:~/gccwork/src/signals$
suc[0]=282.949175
suc[1]=219.742091
suc[2]=246.506147
suc[3]=281.715697
suc[4]=400.264556
suc[5]=239.175226
suc[6]=208.506593
suc[7]=319.014392
suc[8]=90.060582
suc[9]=249.309716
grchere@debian:~/gccwork/src/signals$ kill -SIGUSR1 2735
suc[0]=286.449696
suc[1]=271.901628
suc[2]=275.976403
suc[3]=454.411340
suc[4]=433.911134
suc[5]=325.277379
suc[6]=236.378879
suc[7]=342.380904
suc[8]=304.577989
```

```
suc[9]=296.180029
grchere@debian:~/gccwork/src/signals$ kill -SIGUSR1 2735
grchere@debian:~/gccwork/src/signals$
suc[0]=396.601089
suc[1]=452.099608
suc[2]=407.539806
suc[3]=517.983646
suc[4]=570.824441
suc[5]=437.867196
suc[6]=342.375018
suc[7]=551.282750
suc[8]=367.395490
suc[9]=461.827833

grchere@debian:~/gccwork/src/signals$ kill -SIGUSR2 2735
grchere@debian:~/gccwork/src/signals$ kill -SIGUSR2 2735
bash: kill: (2735) - No such process
[1]+  Done                    ./tp43
grchere@debian:~/gccwork/src/signals$
grchere@debian:~/gccwork/src/signals$
```

8. Copie `tp43.c` como `tp44.c`. Modifique el programa `tp44.c` acorde con el siguiente enunciado: La Empresa X necesita de un programa que se ejecute una vez por minuto y muestre el total de ventas de cada sucursal. Este programa, al igual que `tp43`, terminará su ejecución cuando se reciba la señal `SIGUSR2`.
Sugerencia: utilizar la función `alarm(nsec)` que permite lanzar la señal `SIGALRM` cada `nsec`, capturar la señal `SIGALRM` y en la función de manipulación de esta señal enviar la señal `SIGUSR2` al proceso `tp43` que debe estar ejecutándose concurrentemente con este proceso. Se puede indicar por línea de comando el número de proceso del proceso `tp43`, ya que este dato se necesitará para enviar la señal desde `tp44` a `tp43`.
Podrá obtener una salida como esta (imprimiendo, una vez por minuto los totales de ventas de las sucursales generados por `tp43`):

```
grchere@debian:~/gccwork/src/signals$ ./tp43 &
[1] 2831
grchere@debian:~/gccwork/src/signals$ ./tp44 2831 &
[2] 2832
grchere@debian:~/gccwork/src/signals$
suc[0]=282.949175
suc[1]=219.742091
suc[2]=246.506147
suc[3]=281.715697
suc[4]=400.264556
suc[5]=239.175226
suc[6]=208.506593
suc[7]=319.014392
suc[8]=90.060582
suc[9]=249.309716
suc[0]=286.449696
suc[1]=271.901628
suc[2]=275.976403
suc[3]=454.411340
suc[4]=433.911134
suc[5]=325.277379
suc[6]=236.378879
suc[7]=342.380904
```

```
suc[8]=304.577989
suc[9]=296.180029
grchere@debian:~/gccwork/src/signals$ kill -SIGUSR2 2832
[2]+  Done                    ./tp44 2831
grchere@debian:~/gccwork/src/signals$ ps
  PID TTY          TIME CMD
 2679 pts/0    00:00:00 bash
 2831 pts/0    00:00:00 tp43
 2833 pts/0    00:00:00 ps
grchere@debian:~/gccwork/src/signals$ kill -SIGUSR2 2831
[1]+  Done                    ./tp43
grchere@debian:~/gccwork/src/signals$ ps
  PID TTY          TIME CMD
 2679 pts/0    00:00:00 bash
 2834 pts/0    00:00:00 ps
grchere@debian:~/gccwork/src/signals$
```

9. El diseño propuesto en el punto anterior debería tener de nuestra parte, algunas mínimas consideraciones:
9.1) ¿Qué sucede si el id de proceso indicado en la linea de comando no existe?
9.2) ¿Qué sucede si `tp43` ha dejado de funcionar mientras se esta ejecutando `tp44`?
9.3) ¿Qué sucede si el id de proceso indicado es `1` (proceso `init`)?

Ejecute estos procesos y utilice el shell para probar el comportamiento de estos programas en estos casos.
Copie `tp44.c` como `tp45.c`. Modifique el programa `tp45.c` acorde con las siguientes respuestas:
9.1) usar la señal nula (null signal), la cual se corresponde con el valor 0, para enviarla al proceso indicado, si `kill()` devuelve `-1` y `errno = ESRCH`, implicaría que el proceso no existe. Si el proceso indicado no existe, mostrar mensaje de error y terminar programa. Ejemplo:

```
if (kill((pid_t) procid, 0)  == -1) {
   fprintf(stderr,"main():Error, %d es un proceso invalido",procid);
   exit(2);
}
```

9.2) controlar el retorno de `kill()` en cada envío, si el envío fue satisfactorio `kill()` devolverá `0`. En caso de detectar un envío erróneo, emitir mensaje por `stderr` y terminar normalmente la ejecución del programa, para ello puede utilizar la función `raise()` para enviarse a sí mismo la señal `SIGUSR2` que le permita salir normalmente del programa:

```
if (kill((pid_t) idproc,SIGUSR1) != 0) {
   fprintf(stderr,"sig_alrm():Error enviando señal SIGUSR1 a %d\n",idproc);
   raise(SIGUSR2); // envio señal para obtener totales de ventas
}
```

9.3) chequear lo recibido por la linea de comando, si el id de proceso es <= 1, mostrar mensaje de error y finalizar programa. Usar valores distintos de retorno de la función `main()` para cada tipo de situación de error que detecte este programa.
Compilar y probar el programa en estas situaciones.

```
grchere@debian:~/gccwork/src/signals$ ./tp45
```

```
tp44 <tp43 proc.id>
Donde:
<tp43 proc.id>=nro. de proceso de tp43 que debe estar previamente en ejecucion
grchere@debian:~/gccwork/src/signals$ ./tp45 -10
main():-10 no es un id de proceso valido
tp44 <tp43 proc.id>
Donde:
<tp43 proc.id>=nro. de proceso de tp43 que debe estar previamente en ejecucion
grchere@debian:~/gccwork/src/signals$ ./tp45 0
main():0 no es un id de proceso valido
tp44 <tp43 proc.id>
Donde:
<tp43 proc.id>=nro. de proceso de tp43 que debe estar previamente en ejecucion
grchere@debian:~/gccwork/src/signals$ ./tp45 1
main():1 no es un id de proceso valido
tp44 <tp43 proc.id>
Donde:
<tp43 proc.id>=nro. de proceso de tp43 que debe estar previamente en ejecucion
grchere@debian:~/gccwork/src/signals$
grchere@debian:~/gccwork/src/signals$ ps
  PID TTY          TIME CMD
 2844 pts/0    00:00:01 bash
 2923 pts/0    00:00:00 tp43
 2933 pts/0    00:00:00 ps
grchere@debian:~/gccwork/src/signals$ ./tp45 2930
main():2930 no es un id de proceso valido
tp44 <tp43 proc.id>
Donde:
<tp43 proc.id>=nro. de proceso de tp43 que debe estar previamente en ejecucion
grchere@debian:~/gccwork/src/signals$
......
grchere@debian:~/gccwork/src/signals$ ./tp43 &
[1] 2951
grchere@debian:~/gccwork/src/signals$ ./tp45 2951 &
[2] 2952
grchere@debian:~/gccwork/src/signals$
suc[0]=282.949175
suc[1]=219.742091
suc[2]=246.506147
suc[3]=281.715697
suc[4]=400.264556
suc[5]=239.175226
suc[6]=208.506593
suc[7]=319.014392
suc[8]=90.060582
suc[9]=249.309716
grchere@debian:~/gccwork/src/signals$ kill -SIGUSR2 2951
grchere@debian:~/gccwork/src/signals$
[1]-  Done                    ./tp43
grchere@debian:~/gccwork/src/signals$
grchere@debian:~/gccwork/src/signals$ ps
  PID TTY          TIME CMD
 2844 pts/0    00:00:01 bash
 2952 pts/0    00:00:00 tp45
 2954 pts/0    00:00:00 ps
grchere@debian:~/gccwork/src/signals$
grchere@debian:~/gccwork/src/signals$ sig_alrm():Error enviando se?al SIGUSR1 a
2951
[2]+  Done                    ./tp45 2951
grchere@debian:~/gccwork/src/signals$ ps
```

```
  PID TTY          TIME CMD
 2844 pts/0    00:00:01 bash
 2955 pts/0    00:00:00 ps
grchere@debian:~/gccwork/src/signals$
```

Observaciones en cuanto al uso de señales

1. Es muy importante determinar si su sistema "resetea" o no los manipuladores de señales, acorde con lo realizado en el punto 3. Si un sistema "resetea" el manipulador a su valor por defecto y nosotros "recapturamos" la señal inmediatamente que entramos en la función de manipulacion (esto es lo habitual), en general, suele ser la opción correcta, pero no siempre es así. Por ejemplo, la señal `SIGCLD` funciona distinto que `SIGCHLD` y su "recapturación" podría provocar un loop infinito: cada vez que usamos la función `signal()` para capturar la señal `SIGCLD`, el kernel chequea si hay procesos hijos por los cuales esperar y en tal caso, ejecuta el manipulador; si dentro del manipulador volvemos a ejecutar la función `signal()` capturándola otra vez, entraremos en loop. Para evitar esto, debemos capturar la señal luego de una llamada a la función `wait()`. El programa 10.3, Pag. 282 de Stevens muestra esta situación.

2. Un programa puede funcionar correctamente en un estado de carga de trabajo normal para el SO, mientras que podría entrar en una condición de carrera (race condition) cuando el SO tenga una mayor carga de trabajo. Esto podría ocurrir en señales que se supone ocurrirán una única vez: por ejemplo, supongamos el uso de la función `alarm()` para ejecutar cierto código dentro de `n` segundos y luego llamamos a la función `pause()` para detener el programa hasta la ocurrencia de la señal `SIGALRM`, ¿Qué sucede si el programa es interrumpido por el SO después de la ejecución de `alarm()` y antes de la ejecución de `pause()`? La lentitud del sistema es tal, que para cuando el SO le vuelve a asginar tiempo de CPU al proceso, resulta que el timer ya ha expirado, con lo cual debe ejecutar el manipulador de `SIGALRM` (antes de entrar en `pause()`), una vez ejecutado el manipulador, el cual hace su trabajo y retorna; se ejecuta `pause()` esperando la ocurrencia de una señal, si no hay ninguna otra señal capturada en el programa, esta función `pause()` nunca retornará, por lo tanto, el programa quedará bloqueado indefinidamente.

3. Se debe ser muy cuidadoso con el código que se hace dentro de un manipulador de una señal, por lo general es un codigo muy pequeño y que activa flags que luego son tratados por el programa principal. Tener en cuenta que se trata de una ejecución asincrónica y que existen funciones no re-entrantes, algunas recomendaciones pueden consultarse en la Pag. 611 del Capitulo 24 "Signal Handling" del manual de referencia de la libreria GNU C (The GNU C Library Reference Manual), que se encuentra en el cd-rom de programación en `/Software/Win32/gcc/docs/libc.pdf`.

4. Hay una cuestión de permisos vinculados con las señales, no cualquier proceso puede enviar una señal a otro, esto está en relación con quien es el dueño (owner) del proceso y del grupo de proceso que envía la señal y del proceso que la recibe.

5. Las señales son mensajes asincrónicos que se encolan para cada proceso, otra particularidad es que, cuando se encola `n` veces una misma señal en un proceso determinado, dicha señal se suele enviar sólo una vez y no tantas veces como se generó.

6. Un mismo programa puede manejar `n` señales, con lo cual, cuando se está ejecutando un manipulador de una señal, éste puede ser interrumpido para pasarle el control a otro manipulador. Supongamos que tenemos que realizar la ejecución de una sección crítica de código, se pueden suspender determinadas señales durante la ejecución de dicha sección y restaurarlas luego de haber salido de la sección crítica (Stevens presenta un ejemplo de esta situación en el capítulo 10 Signals).

7. Tener en cuenta la modificación de variables globales desde `main()` y desde los manipuladores de señales, es conveniente declarar las variables globales con el cualificador `volatile` esto impide que se interrumpa el programa mientras se está actualizando el valor de la variable. También ANSI C define el tipo de dato `sig_atomic_t` para lo mismo, ambos pueden combinarse:

```
static volatile sig_atomic_t bandera;
```

Si desea profundizar el tema, recomendamos la lectura del capitulo 10 "Signals" del libro de Stevens en donde se tratan estos problemas y sus propuestas de solución que exceden el alcance de esta actividad práctica. Existe muchísimo más acerca de señales: un proceso puede bloquear señales, trabajar con un conjunto de señales (signal set), tener asociado una máscara de señales, ver si hay señales pendientes, etc. Muchas de las funciónes que Ud. ya conoce hacen uso de señales, tales como: `abort()`, `system()`, `sleep()`, etc.

8. Existen operaciones bloqueantes, por ejemplo, una operacion `read()` puede dejar el proceso bloqueado indefinidamente esperando una recepción de datos que nunca llegará. En este caso, podemos utilizar la función `select()` o `poll()` para indicar allí un período de time-out para una operación de lectura, a modo de evitar esta situación. Pero, ¿Qué sucede con el resto de funciones bloqueantes que no tienen esta posibilidad? Podemos usar la función `alarm()` antes de la función bloqueante, de la siguiente forma:

```
alarm(5); // por ejemplo, establezco time-out de 5 segundos
llamada_a_funcion_bloqueante();
alarm(0); // desactivo alarma
```

entonces, el manipulador de la señal `SIGALRM` será ejecutado en caso de que la función bloqueante no retorne; desde el manipulador de la señal se puede tomar alguna acción. Pero, ¿Qué sucede si pretendo volver de esta función de manipulación de la señal a otra parte del código? Para ello debo utilizar las funciones `sigsetjmp()` y `siglongjmp()`; estas funciones derivan de las funciones `setjmp()` y `longjmp()`, pero estan adaptadas para trabajar con señales. Veamos una nueva versión de `tp45.c` utilizando esta técnica:

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <string.h>
#include <setjmp.h>

void usage();
void sig_usr(int signo);
void sig_alrm(int signo);
int salir=0;
int idproc=0;
sigjmp_buf jmpbuf;

int main(int argc,char *argv[]) { // inicio main()
   if ( argc != 2 ) {
      usage();
      exit(1);
   }
```

```
   idproc=atoi(argv[1]);
   if ( idproc <= 1 ) {
      fprintf(stderr,"main():%d no es un id de proceso valido\n",idproc);
      usage();
      exit(3);
   }
   if ( kill((pid_t) idproc,0) == -1 ) {
      fprintf(stderr,"main():%d no es un id de proceso valido\n",idproc);
      usage();
      exit(2);
   }
   if (signal(SIGALRM, sig_alrm) == SIG_ERR) {
      fprintf(stderr,"main():imposible capturar SIGALRM\n");
      exit(4);
   }
   if (signal(SIGUSR2, sig_usr) == SIG_ERR) {
      fprintf(stderr,"main():imposible capturar SIGUSR2\n");
      exit(5);
   }
   if (sigsetjmp(jmpbuf,1)) {
      printf("main():dentro de 60 segundos vuelvo a pedir totales!\n");
   }
   alarm(60);
   for ( ;!salir; ) {
      pause();
   }
   return 0;
} // fin main()

void sig_usr(int signo) {     /* signo=nro de señal */
   if (signo == SIGUSR2)  salir=1;
   else printf("sig_usr():recibi señal %d\n", signo);
   return;
}
void sig_alrm(int signo)      {     /* signo=nro de señal */
   if (kill((pid_t) idproc,SIGUSR1) != 0) {
      fprintf(stderr,"sig_alrm():Error enviando señal SIGUSR1 a %d\n",idproc);
      raise(SIGUSR2); // envio señal de salida
   }
   siglongjmp(jmpbuf,1);
}
void usage() {
   printf("tp44 <tp43 proc.id>\n");
   printf("Donde:\n");
   printf("<tp43 proc.id>=nro. de proceso de tp43 que debe estar previamente en
ejecucion\n");
}
```

la salida a obtener por este programa sería:

```
grchere@debian:~/gccwork/src/signals$ ./tp46 3079
main():dentro de 60 segundos vuelvo a pedir totales!
suc[0]=1498.857343
suc[1]=1204.173758
suc[2]=1590.140845
suc[3]=2070.052891
suc[4]=1668.643526
suc[5]=1735.123461
suc[6]=1237.069390
```

```
suc[7]=1876.513227
suc[8]=1316.073510
suc[9]=1736.023419
main():dentro de 60 segundos vuelvo a pedir totales!
suc[0]=1514.862496
suc[1]=1408.207045
suc[2]=1601.421418
suc[3]=2182.057773
suc[4]=1706.484476
suc[5]=1786.676463
suc[6]=1408.395217
suc[7]=2033.849564
suc[8]=1457.069481
suc[9]=1943.772091
....
```

analizando la salida podrá deducir el funcionamiento de `sigsetjmp()`, `siglongjmp()`.

Bibliografía

- Stevens, Richard, "Advanced Programming in the UNIX Environment", Addison-Wesley Professional Computing Series, 1993, ISBN 0-201-56317-7
- LooseSandra Loosemore, Richard M. Stallman, Roland McGrath, Andrew Oram, Ulrich Drepper, "The GNU C Library Reference Manual", Free Software Foundation, 2007, Boston, USA.

Espero que esta práctica haya sido de vuestro agrado y le permita comprender mejor los mecanismos básicos de captura y manipulacion de señales como una forma de comunicacion entre procesos provista por el sistema operativo.