Taller de syscalls y señales

Información útil

- https://www.gnu.org/software/libc/manual/html_node/index.html#Top
  - https://www.gnu.org/software/libc/manual/html_node/Signal-Handling.html#Signal-Handling
  - https://www.gnu.org/software/libc/manual/html_node/Processes.html#Processes
- https://linuxjourney.com/
  - https://linuxjourney.com/lesson/monitor-processes-ps-command
  - https://linuxjourney.com/lesson/kernel-overview

Ejercicio 1

¿Cuántos procesos se lanzan y qué comportamiento se puede observar de cada uno?

Ejecutamos el programa y nos pide un argumento. Pongo cualquier cosa para que funcione.

Vemos que imprime alternadamente `sup!` y `ya va!`. Finalmente salta un error:

```
ERROR child exec(...): No such file or directory
```

Corremos strace para ver que está pasando:

```
strace -q ./hai64_original test > /dev/null
```

Vemos que hace una vez esto:

```
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x1d82b50) = 9554
rt_sigaction(SIGINT, {sa_handler=0x40108e, sa_mask=[INT],
sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x406bf0},
{sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
rt_sigaction(SIGCHLD, NULL, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0},
8) = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7fffbd45acf0) = 0
write(1, "sup!\n", 5)                   = 5
kill(9554, SIGURG)                      = 0
```

Luego repite varias veces:

```
rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
rt_sigaction(SIGCHLD, NULL, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0},
8) = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
nanosleep({tv_sec=1, tv_nsec=0}, 0x7fffbd45acf0) = 0
write(1, "sup!\n", 5)                   = 5
kill(9554, SIGURG)                      = 0
```

y termina con:

```
--- SIGINT {si_signo=SIGINT, si_code=SI_USER, si_pid=9554, si_uid=1000} ---
wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 1}], 0, NULL) = 9554
exit_group(0)                           = ?
+++ exited with 0 +++
```

Veamos que son las syscalls que no conocemos:

- **sigprocmask()** is used to fetch and/or change the signal mask of the calling thread. The signal mask is the set of signals whose delivery is currently blocked for the caller. `int rt_sigprocmask(int how, const kernel_sigset_t *set, kernel_sigset_t *oldset, size_t sigsetsize);`.
- **sigaction()** system call is used to change the action taken by a process on receipt of a specific signal. `int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);`. `signum` specifies the signal and can be any valid signal except SIGKILL and SIGSTOP. If `act` is non-NULL, the new action for signal signum is installed from `act`. If `oldact` is non-NULL, the previous action is saved in `oldact`.

Parece que el proceso padre, crea el proceso hijo y luego redefine las señales `SIGINT` y `SIGCHLD`. Espera 1 segundo, escribe `sup!` y le manda al hijo la señal `SIGURG`. Finalmente recibe del hijo la señal `SIGINT`. Entonces llama a `wait` (recordar que al tener el argumento `pid` = -1, vuelve cuando algún hijo termina) con el *PID* del hijo, indicando que terminó y entonces el padre termina.

Según `signal(7)`:

Signal	Standard	Action	Comment
SIGCHLD	P1990	Ign	Child stopped or terminated
SIGURG	P2001	Ign	Urgent condition on socket (4.2BSD)
SIGINT	P1990	Term	Interrupt from keyboard

Por otro lado, sólo vemos que imprime `sup!` por lo tanto podemos suponer que el `ya va!` lo imprime el hijo.

Ahora sigamos a los hijos:

```
strace -q -f ./hai64_original test > /dev/null
```

Para emepezar vemos un solo clone (el del padre que ya habíamos visto) y sólo dos pid, el del padre y el hijo. Por lo tanto solo hay dos procesos.

Vemos también que es el hijo el que escribe `ya va!`.

¿Utilizan los procesos algúna forma de IPC? ¿Cuál es y para qué se usa en este caso? ¿Qué puede inducir del programa a partir de las syscalls observadas?

Por las *syscalls* que redefinen las señales podemos asumir que estan usando señales.

Cada vez que el padre manda `SIGURG`, el hijo escribe `ya va!`.

```
[pid 11541] kill(11542, SIGURG)         = 0
[pid 11542] --- SIGURG {si_signo=SIGURG, si_code=SI_USER, si_pid=11541,
si_uid=1000} ---
[pid 11541] rt_sigprocmask(SIG_BLOCK, [CHLD],  <unfinished ...>
[pid 11542] write(1, "ya va!\n", 7 <unfinished ...>
```

Depués de que el hijo recibe `SIGURG` 5 veces, consigue el *PPID* , que es el *PID* del padre, le manda `SIGINT` al padre y ejecuta lo que le pasamos por como parámetro original. El padre entra a `wait` luego de recibir `SIGINT` del hijo. Cuando el hijo termina, el padre sale del `wait` y termina.

```
[pid 13043] getppid( <unfinished ...>
[pid 13043] kill(13042, SIGINT <unfinished ...>
[pid 13042] <... nanosleep resumed> {tv_sec=1, tv_nsec=35443}) = ?
ERESTART_RESTARTBLOCK (Interrupted by signal)
[pid 13043] <... kill resumed> )        = 0
[pid 13042] --- SIGINT {si_signo=SIGINT, si_code=SI_USER, si_pid=13043,
si_uid=1000} ---
[pid 13042] wait4(-1,  <unfinished ...>
[pid 13043] execve("/usr/bin/ls", ["ls"], 0x7ffe1e24f4e0 /* 63 vars */) = 0
            .
            .
            .
            .
            .
[pid 13043] exit_group(0)               = ?
[pid 13043] +++ exited with 0 +++
<... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) =
13043
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=13043,
si_uid=1000, si_status=0, si_utime=500, si_stime=0} ---
exit_group(0)                           = ?
+++ exited with 0 +++
```

Ejercicio 2

Ver `src/hai.c`.

Ejercicio 3

Ver `antikill.c`.