LLamadas al Sistema Unix

Sistemas Operativos

Escuela de Ingeniería Civil Informática

Esperar un proceso
wait() y waitpid()

UNIVERSIDAD DEL BÍO-BÍO

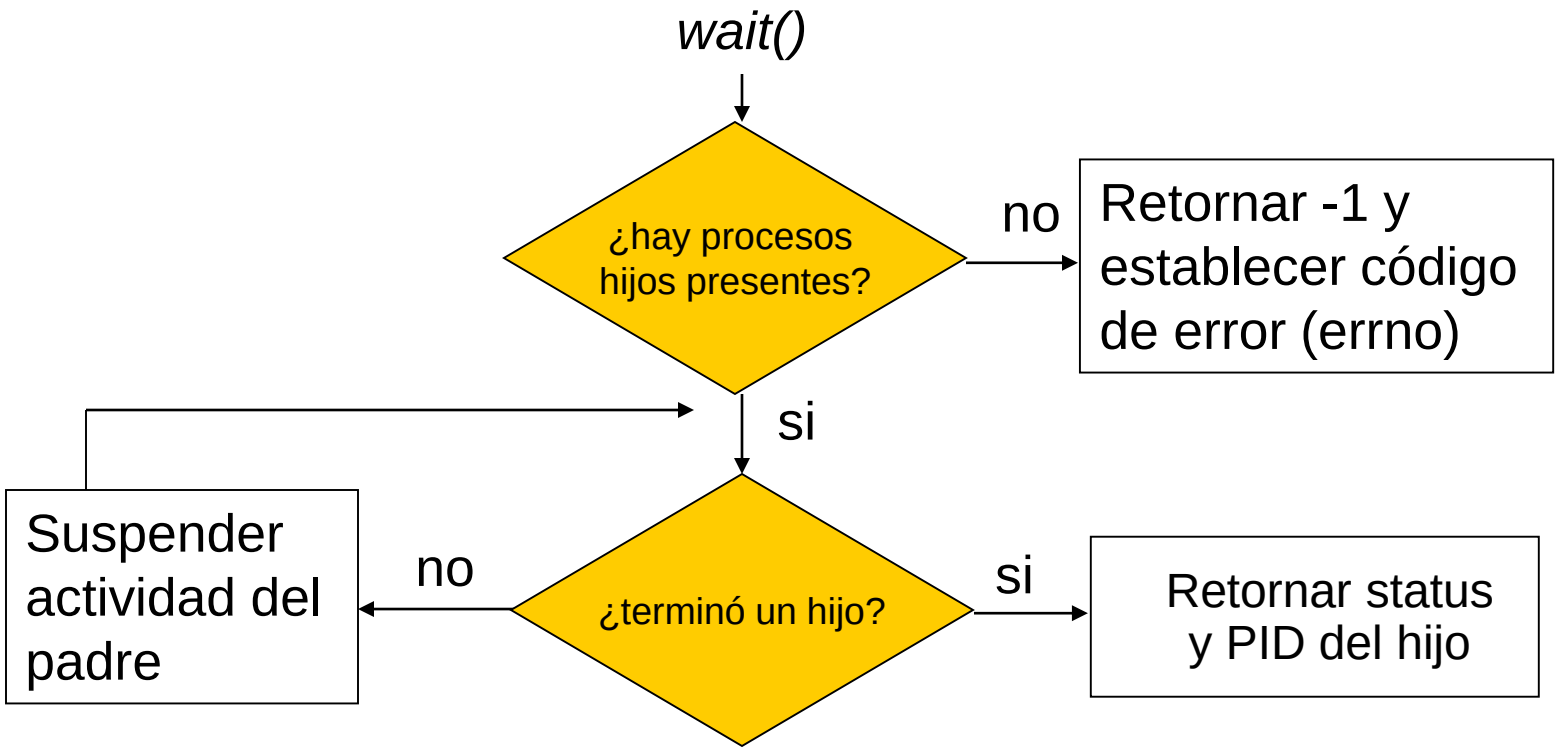
Esperar un Proceso – wait()

- La llamada al sistema *wait()* permite esperar otro proceso:

- Monitorea los cambios de estado y devuelve el PID de:
  - Un proceso hijo que termina
  - Un proceso hijo detenido o reanudado por una señal

Archivo cabecera	`#include <sys/types.h>` `#include <wait.h>`		
Formato	`pid_t wait(int *stat_loc);`		
Salida	Exito	Fallo	Valor en errno
	PID del hijo	-1	Si

Funcionamiento de wait()



Funcionamiento de wait()

- Ejecutar `wait()` sobre un proceso que no tiene hijos
  → retorna un -1 y código de error 10 (`ECHILD`).
- La variable de status `stat_loc` es de 32 bits y contiene:
  - El código de salida (de 0 a 255) en el byte más significativo
    → Si el proceso hijo termina normalmente.

byte 3	byte 2	byte 1	byte 0
		Código de salida	0

  - El número de la señal en el byte menos significativo
    → Si el proceso hijo termina debido a una señal

byte 3	byte 2	byte 1	byte 0
		0	Número de la señal

Funcionamiento de wait()

- El argumento *stat_loc* es un puntero, por lo cual una vez que *wait()* retorna el PID de un proceso hijo esperado se puede consultar el status de salida de ese proceso utilizando para ello un conjunto de macros predefinidas:

```
#include <sys/wait.h>
```

`WIFEXITED(stat_loc)`	¿el proceso terminó normalmente? (0=false sino true)
`WEXITSTATUS(stat_loc)`	retorna el código de salida del proceso hijo (exit(?))
`WIFSIGNALED(stat_loc)`	¿el proceso terminó debido a una señal?
`WTERMSIG(stat_loc)`	retorna el código de la señal
`WIFSTOPPED(stat_loc)`	¿el proceso está actualmente detenido?
`WSTOPSIG(stat_loc)`	retorna la señal que causó que el hijo se detuviera
`WIFCONTINUED(stat_loc)`	¿el proceso hijo continúa en ejecución?
`WCOREDUMP(stat_loc)`	¿se ha generado un archivo CORE? Cuando hay error

Ejemplo de wait()

```
...
int stat;

if (fork() == 0)
    exit(1);
else
    wait(&stat);

if (WIFEXITED(stat))
   printf("Exit status: %d\n", WEXITSTATUS(stat));
else if (WIFSIGNALED(stat))
        signal(WTERMSIG(stat), "Exit signal");
...
```

- Salida del programa:

```
$ ./a.out
Exit status: 1
```

Esperar un Proceso – wait()

- Si un hijo termina, permanece en estado **zombie** hasta que:
  - El proceso padre invoque `wait()` y recupere su estado (y libere al descriptor de proceso asociado)
- Los procesos **zombie** que no tiene hijos: son adoptados por el proceso init (1).
- El proceso init siempre espera a sus hijos
- Por lo tanto, un **zombie** se elimina cuando su padre termina

Limitaciones de wait()

- *wait()* siempre retorna el status del primer hijo que termina o que se detiene.

  → podría no ser el status del hijo que queremos!

- *wait()* siempre se bloquea si la información de status no está disponible

  → esto significa que el padre no puede seguir ejecutándose a la espera del término de un hijo.

- **Solución:** usar otra llamada al sistema → *waitpid()*

Funcionamiento de waitpid()

- La llamada al sistema `waitpid()` permite esperar un proceso especificando su `PID`.
- El prototipo es el siguiente:

Archivo cabecera	`#include <sys/types.h>` `#include <wait.h>`		
Formato	`pid_t waitpid(pid_t pid, int *status, int options);`		
Salida	Exito	Fallo	Valor en errno
	PID del hijo o cero	-1	Si

Funcionamiento de waitpid()

- `waitpid()` funciona de manera diferente según el valor que se le de a *`pid`*:

```
pid_t waitpid(pid_t pid, int *status, int options);
```

Valor de PID	Esperar por
`< -1`	cualquier proceso hijo en el cual el **PGID** sea igual al valor absoluto de *`pid`*.
`-1`	cualquier proceso hijo (funciona igual a wait())
`0`	cualquier proceso hijo cuyo **PGID** sea similar al **PGID** del proceso que invoca waitpid().
`> 0`	El proceso hijo con este *`pid`*

Funcionamiento de waitpid()

- *waitpid()* puede esperar un proceso de diferentes maneras según el valor del parámetro *options*:

```
pid_t waitpid(pid_t pid, int *status, int options);
```

Valor de options	Significado
WNOHANG	No bloquear el proceso si no hay cambio de estado en los procesos hijos. Retorna cero en este caso
WUNTRACED	Reporta término del proceso hijo (debido a las señales SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU)
WCONTINUED	Reporta reanudación de un hijo (debido a la señal SIGCONT)

Ejemplo de waitpid()

```
...
int status;
pid_t cpid = fork();

if (cpid == -1) {
   perror("fork");
   exit(1);
}
if (cpid == 0) { // Código del Hijo
   printf("Child PID is %ld\n", (long)getpid());
   pause(); // Esperar por señales
} else // Código del padre
   do {
      pid_t w = waitpid(cpid, &status, WUNTRACED | WCONTINUED);
      if (w == -1) {
         perror("waitpid");
         exit(1);
      }
      if (WIFEXITED(status))
          printf("exited, status=%d\n", WEXITSTATUS(status));
      else if (WIFSIGNALED(status))
              printf("killed by signal %d\n", WTERMSIG(status));
           else if (WIFSTOPPED(status))
                   printf("stopped by signal %d\n", WSTOPSIG(status));
                else if (WIFCONTINUED(status))
                        printf("continued\n");
   } while (!WIFEXITED(status) && !WIFSIGNALED(status));
...
```

Ejemplo de waitpid()

- Posible salida del programa anterior:

```
$ ./a.out &
Child PID is 32360
[1] 32359
$
$ kill -STOP 32360
stopped by signal 19
$
$ kill -CONT 32360
continued
$
$ kill -TERM 32360
killed by signal 15
[1]+ Done    ./a.out
$
```