1. Finalización de un proceso, exit()

La llamada al sistema exit es la primera en sys_call_table y es la función que termina con la ejecución de un proceso. Esta función se encarga de retirar los recursos que está utilizando el proceso, así como dejarlo preparado para su posterior eliminación.

Cuando un proceso termina debe de comunicar a su padre su finalización por medio de la señal SIGCHLD, de forma que una vez el padre haya sido informado y realice un wait, el hijo sea totalmente eliminado y quitado del planificador. Para reflejar este hecho se denomina al estado transitorio entre la comunicación de finalización y la eliminación total como estado "zombie".

Además de notificar al padre, el exit se encarga de liberar recursos como la memoria tomada por el proceso, así como el sistema de ficheros y las entradas en el mismo y los registros de estado del procesador. Si dicho proceso no tuviera padre, ya que este acabó antes que él, se eliminaría directamente del planificador en la llamada exit y no habría que esperar a comunicar su estado de terminación a nadie.

Muestra de uso

```
/*Programa en C*/

void main ()
{
   ...
   exit (código de error);
   ...
}
```

La llamada *exit* provoca que el proceso termine y le mande *(código de error)* al programa que invocó a "Programa en C".

2. Ejecución de la llamada al sistema exit

¿Qué secuencia de eventos tiene lugar desde que se encuentra una llamada al sistema exit en el código hasta que se empieza a ejecutar do_exit, la función principal del archivo exit.c?. Veámoslo paso a paso como una interrupción software que es:

1. La función exit de la librería libc coloca los parámetros de la llamada en los registros del procesador y se ejecuta la instrucción INT 0x80.
2. Se conmuta a modo núcleo y, mediante las tablas IDT y GDT, se llama a la función sys_call.
3. La función sys_call busca en la sys_call_table la dirección de la llamada al sistema sys_ exit.
4. La función sys_ exit llama a la función do_ exit, que es la función principal del archivo exit.c.

Descripción:

Las principales tareas que lleva a cabo esta llamada el sistema son:

- Especifica que el proceso está finalizando.
  - Escribe el "exit_code"
- Liberación de recursos:
  - Elimina el temporizador del proceso

- ❖ Elimina la memoria asociada al proceso
- ❖ Elimina el proceso de las colas de los semáforos
- ❖ Cierra todos los ficheros que el proceso tenga abiertos
- ❖ Elimina la información del sistema de ficheros
- ❖ Elimina los registros de estado guardados
- ❖ Abre todos los cerrojos que haya cerrado el proceso
- ❖ Libera el enlace entre el proceso y el terminal
- ❖ Elimina las dependencias y entradas del sistema de ficheros
- ❖ Saca al proceso del grupo o subgrupo al que pertenece

- ➢ Notifica al padre de su terminación y establece el exit_state del proceso como "Zombie"; o si no está el padre, lo finaliza del todo (exit_state se establece a DEAD).
- ➢ Ejecuta el planificador

La función principal donde se realizan todas estas acciones es **do_exit**, ésta llama a **exit_notify** para notificar al padre de la finalización de uno de sus hijos. Y en caso de que el padre ya haya finalizado su ejecución antes que el hijo, do_exit llama a **release_task** para eliminar por completo el proceso hijo de la lista de procesos.

3. Espera de un proceso función wait()

La función **wait** suspende la ejecución del proceso actual hasta que un proceso

hijo haya terminado, o hasta que se produce una señal cuya acción es terminar el proceso actual o llamar a la función manejadora de dicha señal. Si un hijo ya había terminado (está en estado Zombie) la función wait vuelve inmediatamente y se encarga de eliminar de la tabla de procesos al hijo e informar al padre acerca de dicha eliminación.

En el núcleo de Linux, un hijo planificado por el núcleo no es una construcción distinta a un proceso. En su lugar, un hilo es simplemente un proceso que es creado usando la llamada al sistema única en Linux clone(2) ; otras rutinas como la llamada portable pthread_create(3) son implementadas usando clone(2). Desde la versión 2.4. de Linux un hilo puede, y por defecto lo hará, esperar a hijos de otros hilos en el mismo grupo de hilos.

Se puede esperar por un hijo mediante una familia de funciones que tienen como función principal esperar hasta que el estado de los hijos lanzados cambie y retornar información de dicho proceso. El cambio de estado e hijos por los que esperar viene determinado por las opciones de la llamada al sistema.

Las llamadas tiene esta sintaxis:

pid_t wait(int *_status_);
pid_t waitpid(pid_t _pid_, int *_status_, int _options_);
int waitid(idtype_t _idtype_, id_t _id_ , siginfo_t * _infop_ , int _options_);

4. Espera de un proceso función waitpid()

Formato

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

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

Descripción general

Suspende el proceso de llamada hasta que un proceso secundario finalice o se detenga. Más precisamente, waitpid () suspende el proceso de llamada hasta que el sistema obtiene información del estado del hijo. Si el sistema ya tiene información de estado sobre un elemento secundario apropiado cuando se llama a waitpid (), waitpid () regresa de inmediato. waitpid () también finaliza si el proceso de llamada recibe una señal cuya acción es ejecutar un controlador de señal o finalizar el proceso.

pid_t *pid*

Especifica los procesos secundarios que la persona que llama desea esperar:

- Si *pid* es mayor que 0, waitpid () espera la terminación del hijo específico cuyo ID de proceso es igual a *pid* .
- Si *pid* es igual a cero, waitpid () espera la terminación de cualquier hijo cuyo ID de grupo de proceso sea igual al de la persona que llama.
- Si *pid* es -1, waitpid () espera a que finalice cualquier proceso secundario.
- Si *pid* es menor que -1, waitpid () espera la terminación de cualquier hijo cuyo ID de grupo de proceso sea igual al valor absoluto de *pid* .

int ** status_ptr*

Apunta a una ubicación donde waitpid () puede almacenar un valor de estado. Este valor de estado es cero si el proceso secundario devuelve explícitamente el estado cero. De lo contrario, es un valor que puede analizarse con las macros de análisis de estado descritas en "Macros de análisis de estado", a continuación.

El puntero *status_ptr* también puede ser NULL, en cuyo caso waitpid () ignora el estado de retorno del hijo.

***opciones* int**

Especifica información adicional para waitpid (). El valor de las *opciones* se construye a partir del OR inclusivo a nivel de bit de cero o más de los siguientes indicadores definidos en el archivo de encabezado sys / wait.h:

WCONTINUED

Comportamiento especial para XPG4.2 : Informa el estado de los procesos secundarios continuos, así como los finalizados. La macro WIFCONTINUED permite que un proceso distinga entre un proceso continuo y uno terminado.

WNOHANG

Exige información de estado de inmediato. Si la información de estado está disponible de inmediato en un proceso secundario apropiado, waitpid () devuelve esta información. De lo contrario, waitpid () regresa inmediatamente con un código de error que indica que la información no estaba disponible. En otras palabras, WNOHANG verifica los procesos secundarios sin hacer que se suspenda la llamada.

WUNTRACED

Informes sobre procesos secundarios detenidos y procesos terminados. La macro WIFSTOPPED permite que un proceso distinga entre un proceso detenido y uno terminado.

Macros de análisis de estado: si el argumento *status_ptr* no es NULL, waitpid () coloca el estado de retorno del hijo en ** status_ptr* . Puede analizar este estado de retorno con las siguientes macros, definidas en el archivo de encabezado sys / wait.h:

WEXITSTATUS (** status_ptr*)

Cuando WIFEXITED () no es cero, WEXITSTATUS () evalúa los 8 bits de orden inferior del argumento de estado que el hijo pasó a la función exit () o _exit (), o el valor que el proceso hijo devolvió de main ().

WIFCONTINUED (** status_ptr*)

Comportamiento especial para XPG4.2 : esta macro se evalúa a un valor distinto de cero (verdadero) si el proceso secundario ha continuado desde una parada de control de trabajo. Esto solo debe usarse después de un waitpid () con la opción WCONTINUED.

WIFEXITED (** status_ptr*)

Esta macro se evalúa en un valor distinto de cero (verdadero) si el proceso secundario finalizó normalmente (es decir, si regresó de main (), o de lo contrario se llamó la función exit () o _exit ()).

WIFSIGNALED (** status_ptr*)

Esta macro se evalúa en un valor distinto de cero (verdadero) si el proceso secundario finalizó debido a una señal que no se capturó.

WIFSTOPPED (** status_ptr*)

Esta macro se evalúa a un valor distinto de cero (verdadero) si el proceso secundario está actualmente detenido. Esto solo debe usarse después de waitpid () con la opción WUNTRACED.

WSTOPSIG (** status_ptr*)

Cuando WIFSTOPPED () no es cero, WSTOPSIG () evalúa el número de la señal que detuvo al hijo.

WTERMSIG (** status_ptr*)

Cuando WIFSIGNALED () no es cero, WTERMSIG () evalúa el número de la señal que finalizó el proceso secundario.

Valor devuelto

Si tiene éxito, waitpid () devuelve un valor del proceso (generalmente un hijo) cuya información de estado se ha obtenido.

Si se proporcionó WNOHANG, y si hay al menos un proceso (generalmente un hijo) cuya información de estado no está disponible, waitpid () devuelve 0.

Si no tiene éxito, waitpid () devuelve -1 y establece errno en uno de los siguientes valores:
Código de error

Descripción

ECHILD

El proceso especificado por *pid* no existe o no es un elemento secundario del proceso de llamada, o el grupo de procesos especificado por *pid* no existe o no tiene ningún proceso miembro que sea un elemento secundario del proceso de llamada.

EINTR

waitpid () fue interrumpido por una señal. El valor de ** status_ptr* no está definido.

EINVAL

El valor de las *opciones* es incorrecto.

5. RESUMEN

En resumen, **wait** solo permite esperar por hijos que hayan terminado, **waitpid** por hijos que hayan terminado/parado/continuado y **waitid** permite mayor control sobre por qué hijo esperar y más información de retorno.

Cuando un proceso muere, durante el tiempo que su padre este esperando por él, se convierte en un proceso "zombie", ya que se sigue guardando su entrada en la tabla de procesos en espera de que su padre lea su estado de salida. En caso que un proceso hijo sea huérfano, su padre ha muerto antes que él, todos los procesos "zombies" son adoptados por el proceso ***init*** que automáticamente hace **waits** para ir eliminando a los "zombies" de la tabla de procesos.