Linux Zombie Process

Linux kernel version：v4.15.10

What’s a Zombie Process?

> 當一個 process 在 linux 上結束後，不會立刻從內存中全部刪除，它的 descriptor 會留在，memory 當中，這時候 process 的 status 變成 `EXIT_ZOMBI`，然後會發送 `SIGCHLD` signal 給 process 的 parent process，通知其 child process 已經結束。
>
> 然後，parent process 應該執行 `wait()`、`wait4()`、`waitpid()` 等等 `wait()` system call，來讀取 child process 的 exit status 和一些資訊，做完之後 child process才會被完全從 memory中清除。
>
> 但是，若 parent process 沒有呼叫 `wait()` system call，那這個 zombie child process 就會被一直留在 memory 當中，直到被清除。

Dangers of Zombie Process

> zombie process 本身殘留的 descriptor 並不會佔據太大的資源，但是每個 zombie process 會保留它的 PID，且系統的 PID 是有限的 32 位元的系統最大值為 32768。
>
> 如果 zombie process 增長到佔據所有 PID 時就會導致新的 process 無法啟動，因此少量的 zombie process 存在是可以的，雖然有 zombie 依舊表示其 parent process 可能有 bug。

Getting Rid of Zombie Processes

> zombie process 不能用 `SIGKILL` signal kill，因為該 process 已經 dead。
>
> 清理 zombie process 的方法其一：
>
> 向 parent process 傳送 `SIGCHILD` signal，parent process 就會去執行 `wait()` system call 清理 zombie child process
>
> 下方 code 的 pid 使用 parent process 的 pid
>
> ```
> kill -s SIGCHILD pid
> ```
>
> 其二：如果是因為 parent process 的設計問題，導致忽略 SIGCHILD signal，可能要中止 parent process，如果child process 的 parent process 結束，init 就會成為新的 parent process （init 是Linux 啟動時的第一個 process pid 1），init 定期執行 wait() system call 清理 zombie process
>
> 其三：直接重開機，也會清除 zombie process

有哪些資訊被回收、回收的資訊用途

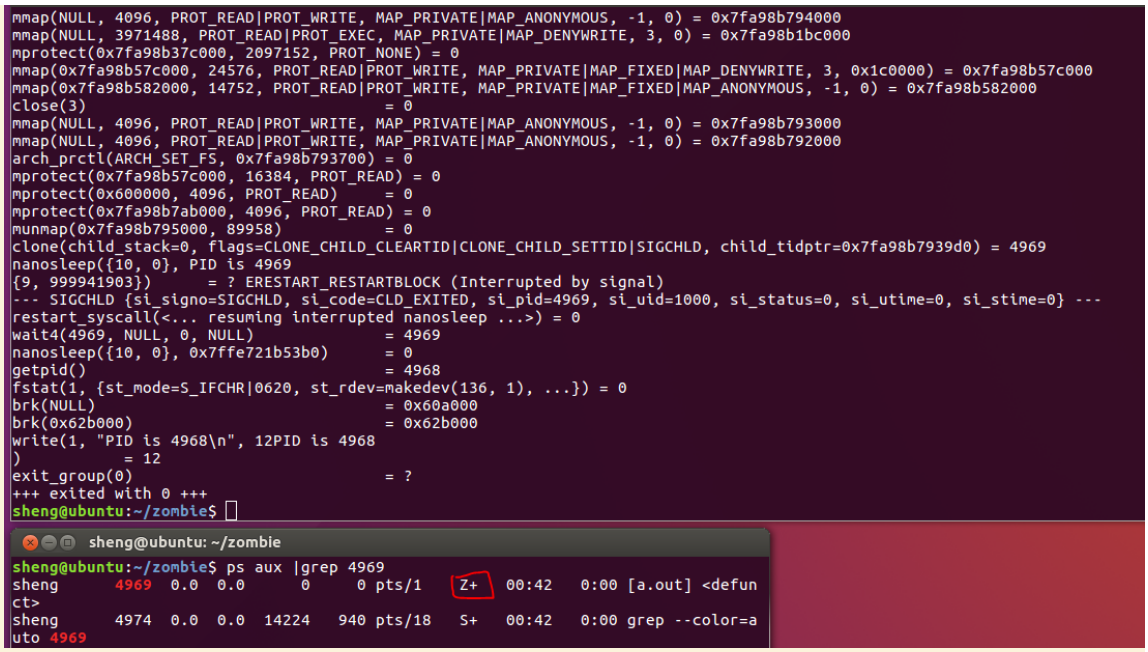
- 回收後 PID 可以再被其他 process 使用

- 允許 parent process 得到 child process 的 PCB （Process Control Block），其中包含 child process 的 exit status（包含了Success, failure （of various sorts） or error （signal））、CPU time, memory usage, IO cycles 等

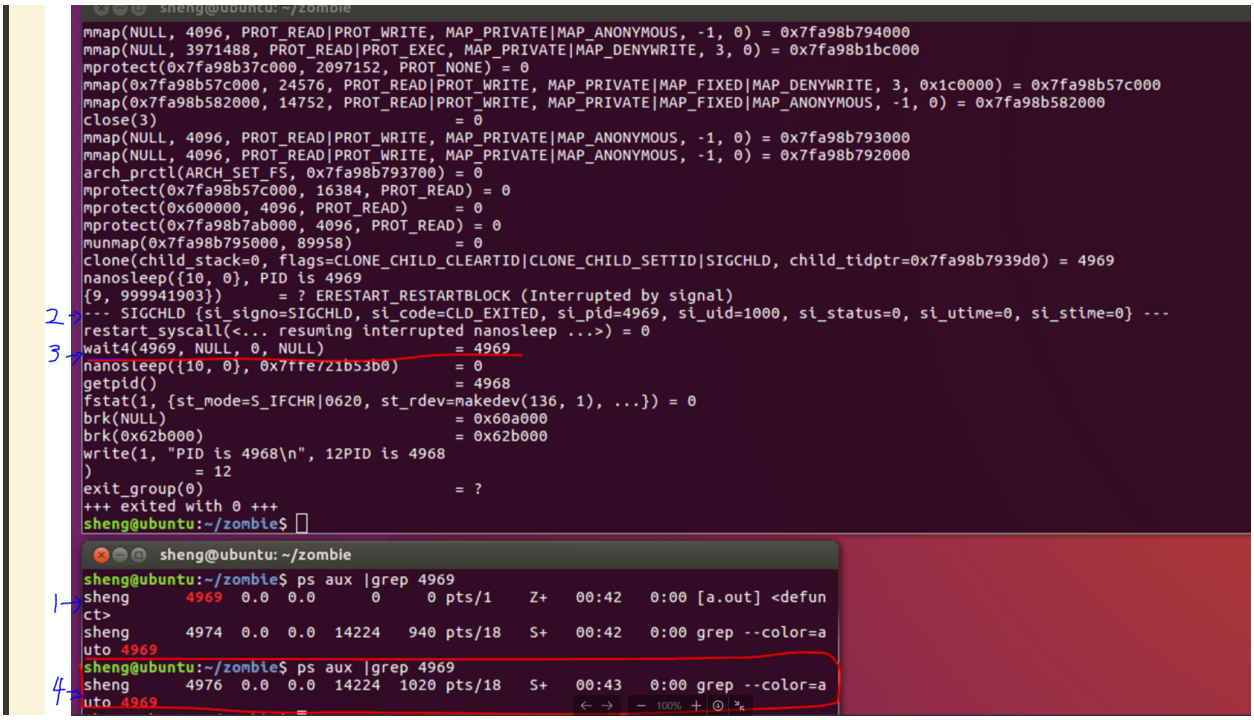
Trace Route

```
//zombie.c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(){

    pid_t PID = fork();

    switch(PID){
        //PID == -1 代表 fork 出錯
        case -1:
            perror("fork()");
            exit(-1);
        // PID == 0 代表是子程序
        case 0:
            //exit(0);
            break;
        // PID > 0 代表是父程序
        default:
            //先 sleep 觀察 zombie process
            sleep(10);
            //呼叫 wait system call 回收 child process
            waitpid(PID,NULL,0);
            //再 sleep 是觀察是否是在 parent process 執行期間回收的
            //如果 parent process 先結束 將是由 init process回收
            sleep(10);


    }
    printf( "PID is %d\n", getpid());

    return 0;
}
```

> 利用 `strace` 觀察 `zombie.c` 的 system call 呼叫狀況
>
> 編譯執行 `zombie.c` 後看到 child process 的 pid **4969**
>
> 1. 觀察 child process 已執行完， status 轉換為 zombie

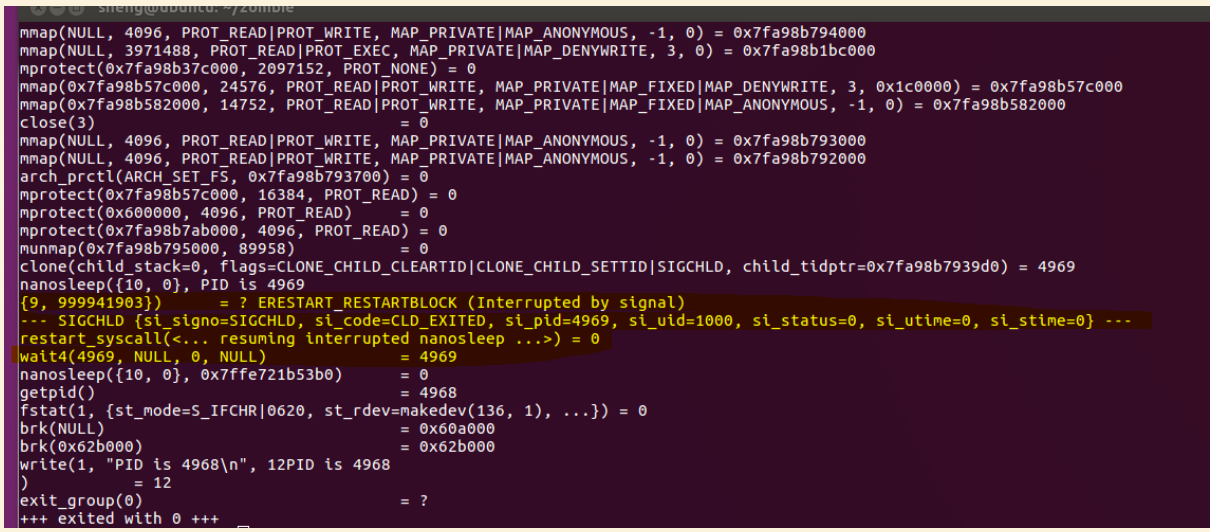


2. 傳送 `SIGCHLD` signal 給 parent process 但 parent 還在 sleep 所以還沒回收
3. 等 parent process 已脫離 sleep 再觀察，下一個執行為 `waitpid()` system call
4. 清理 zombie process 完畢



呼叫流程

- 首先 parent process 先接到由 child process 傳來的 `SIGCHLD` signal



- 收到 `SIGHILD` signal 後呼叫 `waitpid()` 會捕獲到 signal，去清理 zombie process 然後 return 被回收的 child process 的 pid

- 如果沒有 signal 進來但還是調用了 `waitpid()` 時

```
wait4(5202, NULL, 0, NULL)              = -1 ECHILD (No child processes)
```

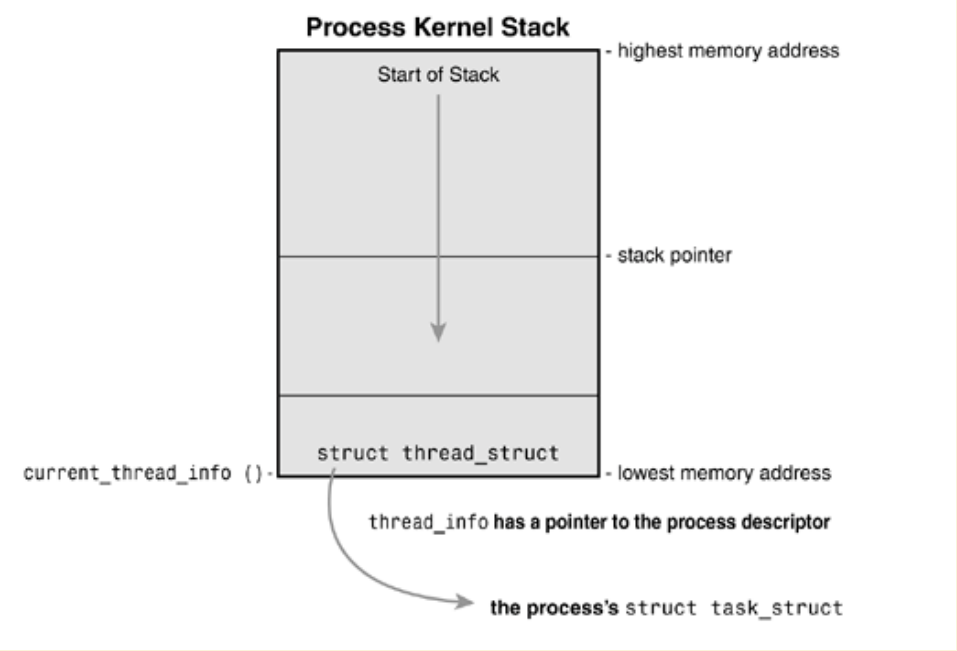
函式用途、目的

`waitpid()` 目的：回收 zombie process pid、process descriptor

waitpid()運作流程

- call `wait4()`
  - 它的主要作用是等待指定的進程終止，並且透過 rusage 獲取 child process 的資源使用訊息。它主要用於 parent process 中，當 parent process 調用 `wait4()` 時，它會等待，直到指定的 child process 終止為止。
- `wait4()` call `do_wait()`
  - `do_wait()` 函數是 `wait4()` 的輔助函數，負責實際上等待進程改變狀態，將當前的工作加入 wait queue，並返回 process的信息給調用者。
- `do_wait()` call `do_wait_thread()` or `ptrace_do_wait()`
  - `ptrace_do_wait()` 檢查目前正在被 `ptrace` debug 的 process 是否滿足等待條件。
  - `do_wait_thread()` 檢查目前 process 中的所有 thread 是否滿足等待條件。
    - 以上兩者如果都返回 0 代表沒找到符合等待條件的 process，返回其他值則表示有找到，具體判斷返回值取決於 `wait_consider_task()` 的 return value。

- `do_wait_thread()` or `ptrace_do_wait()` call `wait_consider_task()`
  - `wait_consider_task()` 檢查被傳進來的 process 是否符合等待條件。
- `wait_consider_task()` call `wait_task_zombie()`
  - 如果在 `wait_consider_task()` 檢查到 process 已經 exit 但是還沒被處理掉，則會歸類為 zombie process 並調用 `wait_task_zombie()` 來處理
- `wait_task_zombie()` call `release_task()`
  - `wait_task_zombie()` 會查看 process 的 exit_code ，將 process 的 exit_state 修改為 EXIT_DEAD，並返回 process 的 exit status、pid 等等訊息給 `do_wait()`
  - `release_task()` 用來釋放一個 process，因為當 process 進入 zombie process 時，它的系統資源仍然在被占用的狀態。
  - 並 `release_task()` 會通知 parent process 已經將 child process 釋放
- 當最後要 release process descriptor 時的步驟
  1. release_task call `__exit_signal()`， `__exit_signal()` call `__unhash_process()`， `__unhash_process()` call `detach_pid()` 將 process 從 pidhash 中移除並將 process 從 task list 中刪除
  2. `__exit_signal()` 會 releases any remaining resources used by the now dead process
     and finalizes statistics and bookkeeping.
  3. `release_task()` call `put_task_struct()` 釋放 process kernel stack、thread_info structure 和 slab cache containing the task_struct（**kernel stack** 是用來記錄 kernel information 每個 process 都有、`thread_info` 中含有指向 `task_struct` 的 pointer）



1. （參考書籍上有補充一個我們先前的流程中沒有蒐集到的 `release_task()` 功能）If the task was the last member of a thread group, and the leader is a zombie, then `release_task()` notifies the zombie leader's parent.

Reference

- Linux Kernel Development, 3/e, Robert Love
- https://www.howtogeek.com/119815/htg-explains-what-is-a-zombie-process-on-linux/
- https://www.baeldung.com/cs/process-lifecycle-zombie-state
- https://blog.csdn.net/ambitiousssssss/article/details/124373565?spm=1001.2101.3001.6650.4&utm_medium=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~Rate-4-124373565-blog-113338301.pc_relevant_default&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~Rate-4-124373565-blog-113338301.pc_relevant_default&utm_relevant_index=4
- https://www.cnblogs.com/yungyu16/p/13024626.html
- https://mozillazg.com/2017/07/python-how-to-generate-kill-clean-zombie-process.html

- https://en.wikipedia.org/wiki/Process_control_block
- https://stackoverflow.com/questions/16416793/why-is-a-zombie-process-necessary
- https://www.quora.com/Why-are-zombie-processes-necessary
- https://elixir.bootlin.com/linux/v4.15.10/source/kernel/exit.c#L1673
- https://web.stanford.edu/class/archive/cs/cs110/cs110.1196/static/lectures/07-Signals/lecture-07-signals.pdf
- https://www.mkssoftware.com/docs/man5/siginfo_t.5.asp#Signal_Codes
- http://books.gigatux.nl/mirror/kerneldevelopment/0672327201/ch03lev1sec1.html
- https://blog.csdn.net/gatieme/article/details/51577479
- https://www.qnx.com/developers/docs/7.0.0/#com.qnx.doc.neutrino.lib_ref/topic/w/wait4.html
- https://www.cnblogs.com/zengyiwen/p/5755182.html