阿男的 Linux 内核世界

阿男

Dec 22, 2016

Contents

1 **进程** 5
Parent Process 与 Child Process 5
小结 8
僵尸进程 8
为什么要有 zombie process? 11

Chapter 1

进 程

Parent Process 与 Child Process

这篇文章我们接着学习 Process。我们学习了 fork() 函数，并且写了 sample code 来使用 fork() 函数创建一个 child process：

```
/* exit() */
#include <stdlib.h>

#include <sys/types.h>

/*
 * fork()
 */
#include <unistd.h>

/*
 * stderr
 * stdout
 * fprintf
 */
#include <stdio.h>
```

```c
void err_sys(const char* fmt, ...);

int main () {
    pid_t pid;
    if ((pid = fork()) < 0) { /* 创建 child process */
        err_sys("fork error");
    } else if (pid == 0) { /* 这里是 child process */
        puts("child process here.");
        puts("child process will sleep for 10 seconds...");
        sleep(10);
    } else { /* 这里是 parent process */
        puts("parent process here.");
        printf("parent get the child pid: %d\n", pid);
    }

    /* 父亲孩子共用的代码部分 */
    sleep(3);

    if (pid == 0) { /* child process */
        puts("child process exit.");
    } else { /* parent process */
        puts("parent process exit.");
    }

    return 0;
}

void err_sys(const char* fmt, ...) {
    va_list ap;
    fprintf(stderr, fmt, ap);
    exit(1);
}
```

这篇文章里面，我们就仔细地分析一下这段代码的执行过程。首先把它编译成可执行文件：

```
[weli@fedora process]$ cc process.c -o process
```

编译完成后，我们把代码执行起来：

```
[weli@fedora process]$ ./process
parent process here.
parent get the child pid: 22418
child process here.
child process will sleep for 10 seconds...
```

然后马上用 ps 命令来查看 process 的运行情况：

```
[weli@fedora process]$ ps -ef | grep process | grep -v grep
weli     22417 18153  0 21:00 pts/2    00:00:00 ./process
weli     22418 22417  0 21:00 pts/2    00:00:00 ./process
```

如上所示，我们可以用 ps 看到 parent 和 child 两个 process。并且从日志和 ps 的输出可以看出来，编号为 22418 的是 child process，它的 parent 是 id 为 22417 的 process。因为我们写的代码是让 parent process 先退出，然后 child 要多等 10 秒再推出，因此我们可以看看 parent process 退出后，child process 的状态是什么样子的：

```
[weli@fedora process]$ ps -ef | grep process | grep -v grep
weli     22418     1  0 21:00 pts/2    00:00:00 ./process
```

从上面的输出可以看到，如果我们等到 parent process 退出，而 child process 没有退出的时候，此时 ps 用查看，就可以正在运行的 process 只有一个了。而且我们看此时这个在运行的 child process，它的 ppid 变成了 1。也就是说，因为 parent process 已经退出不存在了，所以 1 号进程称为了 child process 的 parent，这是内核对 process 继承关系的管理办法。最后我们看一下，等到两个进程都执行完后的状态：

```
[weli@fedora process]$ ./process
parent process here.
parent get the child pid: 22618
child process here.
child process will sleep for 10 seconds...
parent process exit.
[weli@fedora process]$ child process exit.

[weli@fedora process]$ ps -ef | grep process | grep -v grep
```

可以看到，此时 ps 已经看不到 parent 和 child 两个 process 了，因为它们都执行完成了。

小结

我们在这篇文章里看到了 parent 和 child process 的执行过程。并且我们知道了，如果 parent process 先退出，那么还在执行的 child process 的 parent process 就变成了 1 号 process。下篇文章中，阿男给大家讲反过来的情况，就是 child process 先结束的情况。

僵尸进程

这篇文章里面，阿男仍然是接续上一篇的内容，带大家继续学习 Process。上一篇文章里我们使用了一段代码来展示了 parent process 和 child process 的关系：我们看到了 parent process 退出后，还在运行的 child process 的 parent 变成了 pid 为 1 的 process。那如果是 child process 先退出，会是什么情况呢？我们这篇文章就讨论这种情况。这次我们要用到的代码是这样的：

```
/* exit() */
#include <stdlib.h>

#include <sys/types.h>

/*
 * fork()
 */
#include <unistd.h>

/*
 * stderr
 * stdout
 * fprintf
 */
#include <stdio.h>
```

```c
void err_sys(const char* fmt, ...);

int main () {
    pid_t pid;
    if ((pid = fork()) < 0) { /* create a process */
        err_sys("fork error");
    } else if (pid == 0) { /* child process */
        /* Child process 直接退出 */
        exit (0);
    } else {
        /* Parent process 睡眠 60 秒，保证 child process 先退出 */
        fputs("parent process goes to sleep...", stderr);
        sleep (60);
    }
    return 0;
}

// 错误处理函数
void err_sys(const char* fmt, ...) {
    /* va_list 可变长参数，使用 `man stdarg` 命令学习相关文档。*/
    va_list ap;
    fprintf(stderr, fmt, ap);
    exit(1);
}
```

可以看到，我们这次所使用的代码其实和之前使用的差不多，而最大的区别就是，这次的代码是 child process 先退出，然后 parent process 要 sleep 60 秒才退出。因此我们可以看看再这样的情况下，和 child process 先退出会有什么不同。首先我们编译这段代码：

```
[weli@fedora process]$ cc zombie.c -o zombie
```

然后我们运行编译后的代码：

```
[weli@fedora process]$ ./zombie
parent process goes to sleep...
```

此时我们使用 `ps` 命令查看这个程序的两个 process 的运行状况：

```
[weli@fedora process]$ ps -ef | grep zombie
weli      9203  5502  0 19:53 pts/0    00:00:00 ./zombie
weli      9204  9203  0 19:53 pts/0    00:00:00 [zombie] <defunct>
```

此时通过 pid 和 ppid 的关系可以看到，parent process 的 id 是 9203，child process 的 id 是 9204。此时 parent process 在正常运行的状态，而 child process 的状态是 `<defunct>`。也就是说，child process 虽然已经退出了，但是它并没有完全消失，而是还留下了一个 defunct 状态的 “壳”，我们管这种状态的进程叫做 zombie process，也就是僵尸进程。

为什么叫它 zombie process? 因为这个 process 的资源已经已经被释放干净了，只是个没有生命力的空壳而已。我们可以使用 lsof 命令来验证这点：

```
[weli@fedora process]$ lsof -p 9204
[weli@fedora process]$
```

如上所示，我们使用 lsof 命令来查看这个 zombie process 所使用的文件资源，因为只要是在运行的 process，都会打开一些文件的，至少这个 process 自身所对应的程序文件是一定要打开的。我们可以对比一下使用 `lsof` 命令查看正在运行的 parent process 的情况：

```
[weli@fedora process]$ lsof -p 9203
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
zombie  9203 weli  cwd    DIR  253,2     4096 523330
/home/weli/projs/kernel-learning/process
zombie  9203 weli  rtd    DIR  253,0     4096      2 /
zombie  9203 weli  txt    REG  253,2     8912 523335
/home/weli/projs/kernel-learning/process/zombie
zombie  9203 weli  mem    REG  253,0  2093616 795514 /usr/lib64/libc-2.23.so
zombie  9203 weli  mem    REG  253,0   172080 795403 /usr/lib64/ld-2.23.so
zombie  9203 weli    0u   CHR  136,0      0t0      3 /dev/pts/0
zombie  9203 weli    1u   CHR  136,0      0t0      3 /dev/pts/0
zombie  9203 weli    2u   CHR  136,0      0t0      3 /dev/pts/0
```

从上面的输出可以看到，正在运行的 parent process 至少在使用着自己所对应的程序文件`/home/weli/projs/kernel-learning/process /zombie`，而实际上已经退出的 child prorcess 就是一个空壳，实实在在就是一个 zombie！那么为什么 child process 执行完成后，内核还要保留这样一个 zombie process 呢? 这个问题作为本篇文章的读后思考问题，阿男下篇文章为大家进行讲解。

为什么要有 zombie process?

阿男在上篇文章里给大家留了一个问题：child process 退出后，此时 parent process 还没退出，为什么 child process 此时要保留一个 zombie process?

答案是：这是为了进程间同步而考虑。

Linux 提供了一个函数叫做 `waitpid`:

```
WAIT(2)                                                    Linux
Programmer's Manual
WAIT(2)

NAME
       wait, waitpid, waitid - wait for process to change state

SYNOPSIS
       #include <sys/types.h>
       #include <sys/wait.h>

       pid_t wait(int *status);
```