How Linux actually handles fork and exec

s905060.gitbooks.io/site-reliability-engineer-handbook/content/how_linux_actually_handles_fork_and_exec.html

- Site Reliability Engineer HandBook
- Introduction

- Programming Language

Python

- Time Format
- Subprocess
- Multiprocess
- Rename
- SMTP
- Single instance of program
- Argparse
- Requests
- Pyinstaller
- Readlines
- Raw Input
- With Open
- Configparser
- Gzip
- Listdir
- Basename
- Dirname
- Traversing a Directory Tree
- Startswith
- Endswith
- Virtualenv
- Regular Expressions
- Supervisor
- Socket
- Exception Errors
- Raw_input
- Threading
- Unittest
- Why is it better to use “#!\/usr\/bin\/env NAME” instead of “#!\/path\/to\/NAME” as my shebang?
- OS
- Decorator
- String Formatting
- SimplePrograms
- 'all', 'any' are Python built-ins
- TemporaryFile
- How to capture stdout in real-time with Python
- Python simple techniques and common reference
- python reference fragments
- getpass
- Method overriding in Python
- Multiple levels of 'collection.defaultdict' in Python
- String Format
- Logging
- Convert Unicode Object to Python Dict
- The dir( ) Function
- Python dictionary has_key() Method
- glob – Filename pattern matching
- Lambda, filter, reduce and map
- doctest – Testing through documentation
- Load Python code dynamically
- Map, Reduce, Zip, Filter

- DICTIONARY COMPREHENSION

- DD
- BC
- LDD
- getcap, setcap and file capabilities
- Linux_Basename
- PMAP
- Alternative
- Readlink
- logrotate
- PIDOF
- Dmidecode
- lshw
- printenv
- SS
- w
- Strace
- pstree
- USERMOD
- ltrace
- ethtool
- IP
- Sar
- nethogs
- zip
- FPM
- getent
- ipmitool
- Building RPMs
- Megacli
  - Megacli package version
- RKhunter
- fping
- blkid
- FSCK
- Package Manager
- mktemp
- ls
- Comm
- taskset
- fio
- tree
- ARP
- lsblk

- How-To
  - CentOS: nf_conntrack: table full, dropping packet
  - How To Fix “Error: database disk image is malformed” On CentOS \/ Fedora
  - Finding the PID of the process using a specific port?
  - How-To create hashed SSH password
  - How to display and kill zombie processes
  - Shell command to bulk change file extensions in a directory (Linux)
  - 8 Powerful Awk Built-in Variables – FS, OFS, RS, ORS, NR, NF, FILENAME, FNR
  - Changing the Time Zone
  - HOW DO I DISABLE SSH LOGIN FOR THE ROOT USER?
  - How-To rename the extension for a batch of files?
  - How-To disable IPv6 on RHEL6 \/ CentOS 6 \/ etc
  - How to clear the ARP cache on Linux?
  - How-To crontab running as a specific user
  - Ansible – exclude host from playbook execution
  - HOWTO: Use Wireshark over SSH
  - How-To Change Network Interface Name
  - How-To Creating a Partition Size Larger Than 2TB
  - Hot-To Linux Hard Disk Format Command
  - Hadoop Troubleshooting
  - Hive Troubleshooting
  - HowTo Set up hostbased authentication for passphraseless SSH communication.
  - Difference between a cold and warm reboot
  - ls -l explained
  - df falsely showing 100 per cent disk usage
  - FSCK explained
  - Manually generate password for \/etc\/shadow
  - How To Change Timezone on a CentOS 6 and 7
  - Setting ssh private key forwarding
  - Persist keys in ssh-agent on OS X
  - SSH Essentials: Working with SSH Servers, Clients, and Keys
  - How to Change JVM Heap Setting (-Xms -Xmx) of Tomcat – Configure setenv.sh file – Run catalina.sh
  - SSH ProxyCommand example: Going through one host to reach another server
  - How to get Linux's TCP state statistics
  - Linux TCP retransmission rate calculation
  - How to determine OOM
  - How-to check Java process heapsize
  - Troubleshooting network issues
  - How to check what sudo acces a user has?
  - How to copy your key to a remote server?
  - Linux date and Unix timstamp conversion
  - SSH client personalized configuration
  - How to Error Detection and Correction
  - How To Kerberos
  - How to identify defective DIMM from EDAC error on Linux
  - Howto Install and Configure Cobbler on Centos 6
  - How To Use GPG to Encrypt and Sign Messages on an Ubuntu 12.04 VPS
  - HowTo: Debug Crashed Linux Application Core Files Like A Pro
  - Create init script in CentOS 6
  - Linux Change Disk Label Name on EXT2 \/ EXT3 \/ EXT4 File Systems
  - How to retrieve and change partition's UUID Universally Unique Identifier on linux
  - Using Text-Mode Serial Console Redirection

clone

In the kernel, fork is actually implemented by a clone system call. This clone interfaces effectively provides a level of abstraction in how the Linux kernel can create processes.

clone allows you to explicitly specify which parts of the new process are copied into the new process, and which parts are shared between the two processes. This may seem a bit strange at first, but allows us to easily implement threads with one very simple interface.

Threads

While fork copies all of the attributes we mentioned above, imagine if everything was copied for the new process except for the memory. This means the parent and child share the same memory, which includes program code and data.

This hybrid child is called a thread. Threads have a number of advantages over where you might use fork

- Separate processes can not see each others memory. They can only communicate with each other via other system calls.

  Threads however, share the same memory. So you have the advantage of multiple processes, with the expense of having to use system calls to communicate between them.

  The problem that this raises is that threads can very easily step on each others toes. One thread might increment a variable, and another may decrease it without informing the first thread. These type of problems are called concurrency problems and they are many and varied.

  To help with this, there are userspace libraries that help programmers work with threads properly. The most common one is called POSIX threads or, as it more commonly referred to pthreads

- Switching processes is quite expensive, and one of the major expenses is keeping track of what memory each process is using. By sharing the memory this overhead is avoided and performance can be significantly increased.

There are many different ways to implement threads. On the one hand, a userspace implementation could implement threads within a process without the kernel having any idea about it. The threads all look like they are running in a single process to the kernel.

This is suboptimal mainly because the kernel is being withheld information about what is running in the system. It is the kernels job to make sure that the system resources are utilised in the best way possible, and if what the kernel thinks is a single process is actually running multiple threads it may make suboptimal decisions.

Thus the other method is that the kernel has full knowledge of the thread. Under Linux, this is established by making all processes able to share resources via the clone system call. Each thread still has associated kernel resources, so the kernel can take it into account when doing resource allocations.

Other operating systems have a hybrid method, where some threads can be specified to run in userspace only ("hidden" from the kernel) and others might be a light weight process, a similar indication to the kernel that the processes is part of a thread group.

Copy on write

As we mentioned, copying the entire memory of one process to another when fork is called is an expensive operation.

One optimisation is called copy on write. This means that similar to threads above, the memory is actually shared, rather than copied, between the two processes when fork is called. If the processes are only going to be reading the memory, then actually copying the data is unnecessary.

However, when a process writes to it's memory, it needs to be a private copy that is not shared. As the name suggests, copy on write optimises this by only doing the actual copy of the memory at the point when it is written to.

Copy on write also has a big advantage for exec. Since exec will simply be overwriting all the memory with the new program, actually copying the memory would waste a lot of time. Copy on write saves us actually doing the copy.

The init process

We discussed the overall goal of the init process previously, and we are now in a position to understand how it works.

On boot the kernel starts the init process, which then forks and execs the systems boot scripts. These fork and exec more programs, eventually ending up forking a login process.

The other job of the init process is "reaping". When a process calls exit with a return code, the parent usually wants to check this code to see if the child exited correctly or not.

However, this exit code is part of the process which has just called exit. So the process is "dead" (e.g. not running) but still needs to stay around until the return code is collected. A process in this state is called a zombie (the traits of which you can contrast with a mystical zombie!)

A process stays as a zombie until the parent collects the return code with the wait call. However, if the parent exits before collecting this return code, the zombie process is still around, waiting aimlessly to give it's status to someone.

In this case, the zombie child will be reparented to the init process which has a special handler that reaps the return value. Thus the process is finally free and can the descriptor can be removed from the kernels process table.

Zombie example

```
$ cat zombie.c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        pid_t pid;

        printf("parent : %d\n", getpid());

        pid = fork();

        if (pid == 0) {
                printf("child : %d\n", getpid());
                sleep(2);
                printf("child exit\n");
                exit(1);
        }

        /* in parent */
        while (1)
        {
                sleep(1);
        }
}

ianw@lime:~$ ps ax | grep [z]ombie
16168 pts/9    S      0:00 ./zombie
16169 pts/9    Z      0:00 [zombie] <defunct>
```

Above we create a zombie process. The parent process will sleep forever, whilst the child will exit after a few seconds.

Below the code you can see the results of running the program. The parent process (16168) is in state S for sleep (as we expect) and the child is in state Z for zombie. The ps output also tells us that the process is defunct in the process description.