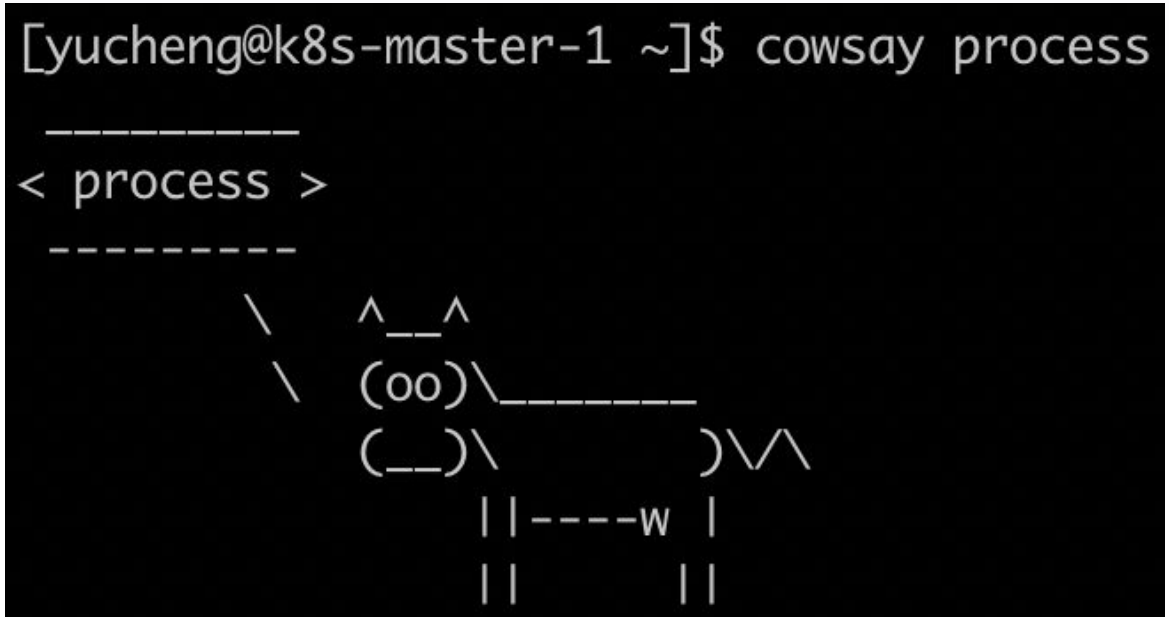
案例篇：系统中出現大量不可中斷程序和殭屍程序怎麼辦？

91APP - Tim

2020/05

再說不可中斷程序和殭屍程序之前我們來說說什麼是程序。



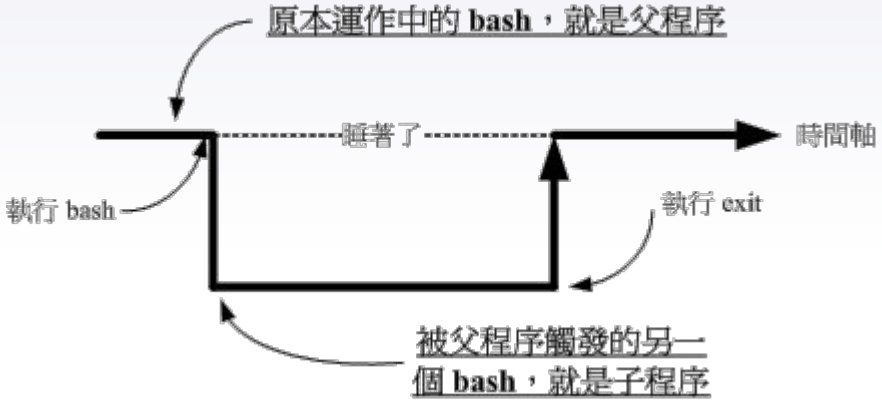
什麼是程序

- 執行一個程式或指令時，系統都會將他定義成為一個程序，並且給予這個程序一個ID，稱為 PID，同時依據啟發這個程序的使用者與相關屬性關係，給予這個PID 一組有效的權限設定。

- 作業系統為了可管理這個程序，因此程序有給予執行者的權限屬性等參數，並包括程式所需要的指令碼與資料或檔案資料等，最後再給予一個 PID 。系統就是透過這個 PID 來判斷該 process 是否具有權限進行工作的！

Ref: 鳥哥私房菜 - 程序管理與 SELinux 初探

子程序與父程序



- 正常情況下：子程序由父程序建立，子程序再建立新的程序。父程序無法預測子程序的結束，所以，當子程序結束後，它的父程序會使用wait()或 waitpid()取得子程序的終止狀態，回收掉子程序的資源。

Ref: 鳥哥私房菜 - 程序管理與 SELinux 初探

子程序與父程序

- 連續執行兩個 bash 後，第二個 bash 的父程序就是前一個bash。因為每個程序都有一個 PID，那某個程序的父程序該如何判斷？就透過Parent PID (PPID) 來判斷即可。

```
[yucheng@k8s-master-1 ~]$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 22172 22171  0  80   0 - 28895 do_wai pts/0    00:00:00 bash
0 R  1000 22194 22172  0  80   0 - 38312 -      pts/0    00:00:00 ps
```

- 第一個 bash 的 PID 與第二個 bash 的 PPID 都是 22172，因為第二個 bash 是來自於第一個所產生的

Ref: 鳥哥私房菜 - 程序管理與 SELinux 初探

程序状态 - top

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3110	root	20	0	483368	229220	36832	S	3.7	0.7	8299:32	kube-apiserver
1799	root	20	0	1450140	77212	34736	S	2.3	0.2	5831:46	kubelet
3100	root	20	0	10.1g	37264	11092	S	1.7	0.1	3101:12	etcd
1033	root	20	0	68180	21612	21232	S	0.7	0.1	1632:29	systemd-journal
2071	root	20	0	949176	86460	25188	S	0.7	0.3	2017:05	dockerd
2069	root	20	0	693708	16932	14260	S	0.3	0.1	758:17.05	rsyslogd
3194	root	20	0	107336	8604	2860	S	0.3	0.0	25:41.13	containerd-shim
3215	root	20	0	147592	29280	14348	S	0.3	0.1	1533:13	kube-scheduler
31923	**yucheng**	**20**	**0**	**162028**	**2340**	**1600**	**R**	**0.3**	**0.0**	**0:00.17**	**top**
1	root	20	0	46400	6900	4172	S	0.0	0.0	4:27.29	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:04.86	kthreadd
4	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
6	root	20	0	0	0	0	S	0.0	0.0	5:50.82	ksoftirqd/0

程序幾個主要狀態碼 (S 列 STAT)

- R (Running): 該程序目前正在運作，或者是可被運作。
- D: 不可被喚醒的睡眠狀態 (Uninterruptible Sleep)，通常這支程式可能在等待 I/O 的情況。
- Z (Zombie): 僵屍狀態，程序已經終止但卻無法被移除至記憶體外。
- S (Sleep): 該程序目前正在睡眠當中 (Idle)，但可被某些訊號 (Signal) 喚醒。
- I (Idle) : 也就是空閒狀態，用在不可中斷睡眠的核心執行緒上。前面說了，等待 I/O 時不可中斷程序用 D 表示，但對某些核心執行緒來說，它們有可能實際上並沒有任何負載，用 Idle 正是為了區分這種情況。要注意，D 狀態的程序會導致平均負載升高，I 狀態的程序卻不會。

程序幾個主要狀態碼 (S 列 STAT)

- T 或 t，也就是 Stopped 或 Traced 的縮寫，表示程序處於暫停或者跟踪狀態。
- X(dead)：程序已死亡，不會在top 或 ps 命令中看到它。

程序幾個主要狀態碼 (S 列 STAT)

- <: 高優先權的工作
- N: 低優先權的工作
- L: 已分頁鎖定於記憶體(for real-time and custom IO)
- s : 一個 session leader
- +: 在前景的程序群組中

```
[root@xuegod63 ~]# ps -axu | grep tar
root      1911  0.0  0.3 1279032 7184 ?        S<l  20:23   0:02 /usr/bin/pulseaudio
--start
root      4746 14.0  0.0 123660  1480 pts/1    S+   21:11   0:01 tar -zcvf usr.tar.gz
 /usr
root      4758  0.0  0.0 112680   984 pts/0    S+   21:11   0:00 grep --color=auto ta
r
[root@xuegod63 ~]# ps -axu | grep tar
root      1911  0.0  0.3 1279032 7184 ?        S<l  20:23   0:02 /usr/bin/pulseaudio
--start
root      4746 14.8  0.0 123660  1480 pts/1    R+   21:11   0:02 tar -zcvf usr.tar.gz
 /usr
root      4766  0.0  0.0 112680   984 pts/0    S+   21:11   0:00 grep --color=auto ta
r
[root@xuegod63 ~]# ps -axu | grep tar
root      1911  0.0  0.3 1279032 7184 ?        S<l  20:23   0:02 /usr/bin/pulseaudio
--start
root      4746 15.7  0.0 123660  1480 pts/1    D+   21:11   0:02 tar -zcvf usr.tar.gz
 /usr
root      4774  0.0  0.0 112680   984 pts/0    S+   21:11   0:00 grep --color=auto ta
```

殭屍程序

- 通常，造成僵屍程序的成因是因為該程序應該已經執行完畢，或者是因故應該要終止了，但是該程序的父程序卻無法完整的將該程序結束掉，而造成那個程序一直存在記憶體當中。如果你發現在某個程序的 CMD 後面還接上 <defunct> 時，就代表該程序是僵屍程序

```
top - 15:51:49 up 175 days, 11:07,  1 user,  load average: 0.00, 0.01, 0.05
Tasks: 10972 total,   1 running, 143 sleeping,   0 stopped, 10828 zombie
%Cpu(s):  1.3 us,  1.5 sy,  0.0 ni, 97.1 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 32664880 total, 29416588 free,   686052 used,  2562240 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 29917972 avail Mem
```

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
1 Z     0   306 28254  0  80   0 -     0 do_exi pts/0    00:00:00 app <defunct>
1 Z     0   307 28254  0  80   0 -     0 do_exi pts/0    00:00:00 app <defunct>
1 Z     0   308 28254  0  80   0 -     0 do_exi pts/0    00:00:00 app <defunct>
1 Z     0   309 28254  0  80   0 -     0 do_exi pts/0    00:00:00 app <defunct>
```

孤兒程序

- 一個父程序退出，而它的一個或多個子程序還在執行，那麼那些子程序將成為孤兒程序。孤兒程序將被init程序(程序號為1)所收養，並由 init 程序對它們完成狀態收集工作。

Ref: 鳥哥私房菜 - 程序管理與 SELinux 初探

案例分析

```
# 先删除上次启动的案例
$ docker rm -f app
# 重新运行案例
$ docker run --privileged --name=app -itd feisky/app:iowait
```

- 使用教學提供的Docker image 進行分析

案例分析

```
Tasks: 165 total,   1 running, 146 sleeping,   0 stopped,  18 zombie
%Cpu(s):  0.1 us,  0.1 sy,  0.0 ni, 99.8 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 32664880 total, 29590988 free,   410728 used,  2663164 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 30216648 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  959 root      20   0       0      0      0 S   0.3  0.0  73:24.18 xfsaild/dm-0
17142 yucheng   20   0  162020   2296   1588 R   0.3  0.0   0:00.08 top
    1 root      20   0   46084   6668   4156 S   0.0  0.0 113:35.86 systemd
```

- 查看 Top 發現有 18 隻殭屍程序

案例分析

```
root     17075  0.0  0.0   4368   428 pts/0    Ss+  19:03   0:00 /app
root     17108  0.0  0.0      0     0 pts/0    Z+   19:03   0:00 [app] <defunct>
root     17109  0.0  0.0      0     0 pts/0    Z+   19:03   0:00 [app] <defunct>
root     17126  0.0  0.0      0     0 pts/0    Z+   19:03   0:00 [app] <defunct>
root     17127  0.0  0.0      0     0 pts/0    Z+   19:03   0:00 [app] <defunct>
root     17128  0.0  0.0      0     0 pts/0    Z+   19:03   0:00 [app] <defunct>
```

- 使用 ps -aux 後發現 app 這隻父程序一直在產出殭屍程序

案例分析

```
[yucheng@k8s-master-3 ~]$ pstree -aps 17075
systemd,1 --switched-root --system --deserialize 22
  └─containerd,2059
      └─containerd-shim,17056 -namespace moby -workdir...
          └─app,17075
              ├─{app,17108}
              ├─{app,17109}
              ├─{app,17126}
              ├─{app,17127}
              ├─{app,17128}
              ├─{app,17129}
              ├─{app,17140}
              ├─{app,17141}
              ├─{app,17143}
              ├─{app,17144}
              ├─{app,17155}
```

- 使用 pstree 查詢其中一隻殭屍程序後發現app 的父程序 PID 為 17075

殭屍程序處理

- 殭屍程序並不是問題的根源，罪魁禍首是產生大量殭屍程序的父程序。因此，我們可以直接除掉元凶，通過 kill 發送 SIGTERM 或者 SIGKILL 信號。元凶死後，殭屍程序變成孤兒程序，由 init 充當父程序，並回收資源。

思考一下

- 你碰過的不可中斷狀態程序和殭屍程序問體。你是怎麼分析他們的來源？又是怎麼解決的？