程序管理

教师: 李军辉

在Linux系统中，

- 触发任何一个事件时，系统都会将他定义成为一个程序，并且给予这个程序一个 ID，称为 PID，同时依据启发这个程序的使用者与相关属性关系，给予这个 PID 一组有效的权限配置。

程序与进程 (program & process)

让我们将程序与程序作个总结：

- 程序 (program)：通常为 binary program，放置在储存媒体中 (如硬盘、光盘、软盘、磁带等)，为实体文件的型态存在；
- 进程 (process)：程序被触发后，运行者的权限与属性、程序的程序码与所需数据等都会被加载内存中，操作系统并给予这个内存内的单元一个识别码 (PID)，可以说，进程就是一个正在运行中的程序。

子进程与父进程

例如:

1. 登陆系统后，会取得一个 bash 的 shell，会产生一个进程并关联一个PID

2. 运行bash命令，并开启一个新的进程，这个新的进程称为子进程, 原来的bash为父进程.

注意: 子程序可以取得父程序的环境变量.

ps -l查看进程.

- PID: 进程编号
- PPID: 父进程编号

&的使用

```
#!/bin/bash
sleep 60
echo "OK"
exit 0
```

运行:

```
bash test.sh
bash test.sh &
```

&的使用

```
#!/bin/bash

for i in `seq 1 10`; do
  for j in `seq 1 5`; do
    echo 'HELLO'
  done
  sleep 10
done
```

运行:

```
bash test.sh &
bash test.sh > /dev/null 2>&1 &
```

假设现在我有6个作业需要运行, 每个作业的执行都可能需要较长时间. 它们之间没有依赖关系？ 怎么来完成这6个作业的运行?

假设现在我有6个作业需要运行, 每个作业的执行都可能需要较长时间. 它们之间没有依赖关系? 怎么来完成这6个作业的运行?

工作管理 / 作业管理

进行作业管理的行为中，其实每个作业都是目前 bash 的子程序，亦即彼此之间是有相关性的。我们无法以 job control 的方式由 tty1 的环境去管理 tty2 的 bash！

由於假设我们只有一个终端介面，因此在可以出现提示字节让你操作的环境就称为前景 (foreground)，至於其他工作就可以让你放入背景 (background) 去暂停或运行。

要注意的是，放入背景的工作想要运行时，它必须不能够与使用者互动。
想想有哪些作业不可以放入背景运行?

要进行 bash 的 job control 必须要注意到的限制是：

- 这些工作所触发的程序必须来自於你 shell 的子程序(只管理自己的 bash)；
- 前景：你可以控制与下达命令的这个环境称为前景的工作(foreground)；
- 背景：可以自行运行的工作，你无法使用 ctrl + c 终止他，可使用 bg/fg 呼叫该工作；
- 背景中运行的程序不能等待 terminal/shell 的输入(input)

直接将命令丢到背景中运行的 &

```
tar -zpcvf ~/etc.tar.gz /etc &
[1] 8400 <== [job number] PID
```

直接将命令丢到背景中运行的 &

```
tar -zpcvf ~/etc.tar.gz /etc &
[1] 8400 <== [job number] PID
```

注意标准输出和标准错误输出.

```
tar -zpcvf ~/etc.tar.gz /etc > /tmp/log.txt 2>&1 &
```

如果我正在使用 vim编写程序，却发现我有个文件不知道放在哪里，需要到 bash 环境下进行搜寻，那此时怎么办?

如果我正在使用 vim编写程序，却发现我有个文件不知道放在哪里，需要到 bash 环境下进行搜寻，那此时怎么办?

- 正常退出vim程序,

如果我正在使用 vim编写程序，却发现我有个文件不知道放在哪里，需要到 bash 环境下进行搜寻，那此时怎么办?

- 正常退出vim程序,
- 将『目前』的工作丢到背景中『暂停』：[ctrl]-z

观察目前的背景工作状态： jobs

jobs [-lrs]
选项与参数：
-l ：除了列出 job number 与命令串之外，同时列出 PID 的号码；
-r ：仅列出正在背景 run 的工作；
-s ：仅列出正在背景当中暂停 (stop) 的工作。

```
jobs -l
[1]- 10314 Stopped vim ~/.bashrc
[2]+ 10833 Stopped find / -print
```

+ 代表默认的取用工作。即输入 fg 时，那么那个 [2] 会被拿到前景当中来处理。

将背景工作拿到前景来处理：fg

fg %jobnumber
选项与参数：
%jobnumber ：jobnumber 为工作号码(数字)。注意，那个 % 是可有可无的！

范例一：先以 jobs 观察工作，再将工作取出：

```
jobs
[1]- 10314 Stopped vim ~/.bashrc
[2]+ 10833 Stopped find / -print
fg <==默认取出那个 + 的工作，亦即 [2]。立即按下[ctrl]-z
fg %1 <==直接规定取出的那个工作号码！再按下[ctrl]-z
jobs
[1]+ Stopped vim ~/.bashrc
[2]- Stopped find / -print
```

让工作在背景下的状态变成运行中： bg

范例一：一运行 find / -perm 755 > /tmp/text.txt 后，立刻丢到背景去暂停！

```
find / -perm 755 > /tmp/text.txt
# 此时，请立刻按下 [ctrl]-z 暂停！
[3]+ Stopped find / -perm 755 > /tmp/text.txt
```

范例二：让该工作在背景下进行，并且观察他！！

```
jobs ; bg %3 ; jobs
[1]- Stopped vim  /.bashrc
[2] Stopped find / -print
[3]+ Stopped find / -perm 755 > /tmp/text.txt
[3]+ find / -perm +7000 > /tmp/text.txt & <==用 bg%3 的情况！
[1]+ Stopped vim ~/.bashrc [2] Stopped find / -print [3]- Running
find / -perm 755 > /tmp/text.txt &
```

管理背景当中的工作： kill

```
kill %2 # 中止背景编号为2的作业
kill 2 # 中止PID为2的作业
```

离线管理问题

假设程序 test.sh需要运行1个小时, 你在某个终端下通过

```
bash test.sh &
```

将该作业于背景运行.

问题: 如果将该终端关闭, 程序test.sh是否还会继续运行?

离线管理问题

假设程序 test.sh需要运行1个小时, 你在某个终端下通过

```
bash test.sh &
```

将该作业于背景运行.

问题: 如果将该终端关闭, 程序test.sh是否还会继续运行?

- 工作管理的背景依旧与终端机有关
- 程序test.sh不会继续进行，而是会被中断掉。

- at 来处理即可！因为 at 是将工作放置到系统背景，而与终端机无关。
- nohup命令

nohup [命令与参数] & <==在终端机背景中工作

1. 先编辑一支会『睡著 500 秒』的程序：

```
vim sleep500.sh
#!/bin/bash
/bin/sleep 500s
/bin/echo "I have slept 500 seconds."
```

2. 丢到背景中去运行，并且立刻注销系统：

```
chmod a+x sleep500.sh
nohup ./sleep500.sh &
[1] 5074
nohup: appending output to 'nohup.out' <==会告知这个信息！
exit
```

进程管理

ps ： 将某个时间点的程序运行情况摘取下来.

ps aux <==观察系统所有的程序数据
ps -lA <==也是能够观察所有系统的数据
ps axjf <==连同部分程序树状态

选项与参数：
-A ： 所有的 process 均显示出来，与 -e 具有同样的效用；
-a ： 不与 terminal 有关的所有 process ；
-u ： 有效使用者 (effective user) 相关的 process ；
x ： 通常与 a 这个参数一起使用，可列出较完整资讯。

输出格式规划：
l ： 较长、较详细的将该 PID 的的资讯列出；
j ： 工作的格式 (jobs format)
-f ： 做一个更为完整的输出。

仅观察自己的 bash 相关程序： ps -l

观察系统所有程序： ps aux

范例五：找出与 cron 与 syslog 这两个服务有关的 PID 号码？
ps aux | egrep '(cron|syslog)'

top：动态观察程序的变化