# CHUYÊN ĐỀ HỆ ĐIỀU HÀNH LINUX

## Tuần 7: Tiến trình

GVLT: NGUYỄN Thị Minh Tuyền

# Nội dung

1. Tiến trình trong Linux
2. Các biến môi trường
3. Here document
4. Các lệnh liên quan đến pipeline
5. Tính toán số học

# Nội dung

1. **Tiến trình trong Linux**
2. Các biến môi trường
3. Here document
4. Các lệnh liên quan đến pipeline
5. Tính toán số học

# Tiến trình trong Linux

## Một tiến trình

- chương trình đang thực thi
- cần tài nguyên thiết bị: CPU, bộ nhớ trung tâm, truy cập vào các thiết bị vào/ra.

# Cây phân cấp của tiến trình

- Mỗi tiến trình luôn luôn được ra bởi một tiến trình khác gọi là tiến trình cha.
- Mỗi tiến trình có 0, 1 hoặc + con
- Tiến trình gốc là `init` (PID = 1).
  - `init` được gán tự động, không có cha
- Hai loại tiến trình tồn tại:
  - Tiến trình người sử dụng, luôn luôn tạo ra từ shell đăng nhập;
  - Các tiến trình « kernel »
- Để hiển thị bảng các tiến trình : `ps`, `pstree`, `top`

# Các thuộc tính của một tiến trình

- uid effective user ID
- pid process ID
- ppid parent process ID
- %cpu percentage CPU usage
- %mem percentage memory usage
- vsz virtual size in Kbytes
- started time started
- time accumulated CPU tim
- command command and arguments
- ....

# Tạm dừng và tiếp tục một tiến trình

- Trong Linux, có thể tạm dừng tiến trình chạy ở foreground bằng cách ấn CTRL–Z.
  - Tiến trình tạm dừng có thể tiếp tục sau đó.
- Có hai cách để tiếp tục một tiến trình bị tạm dừng:
  - Chạy ở foreground sử dụng lệnh  fg (foreground),
  - Chạy ở background sử dụng lệnh bg (background).
- Ví dụ:

| | |
|---|---|
| emacs | chạy emacs ở foreground |
| CTRL–Z | tạm dừng emacs |
| bg | tiếp tục emacs ở background |

- Một « job » được định nghĩa như một tiến trình ở background hoặc tạm dừng. Lệnh  `jobs` cho phép liệt kê những tiến trình này.

# Huỷ một tiến trình

- Một tiến trình tự kết thúc sau khi thực hiện lệnh cuối cùng: Được huỷ bởi hệ điều hành.
- Ctrl-C: kết thúc một tiến trình chạy ở foreground.
- Kết thúc một tiến trình với lệnh `kill` : gởi một tín hiệu đến một tiến trình.

```
$ kill PID
```

  - `kill` gởi tín hiệu 15 để kết thúc (`SIGTERM`)

```
$ kill –9 PID
```

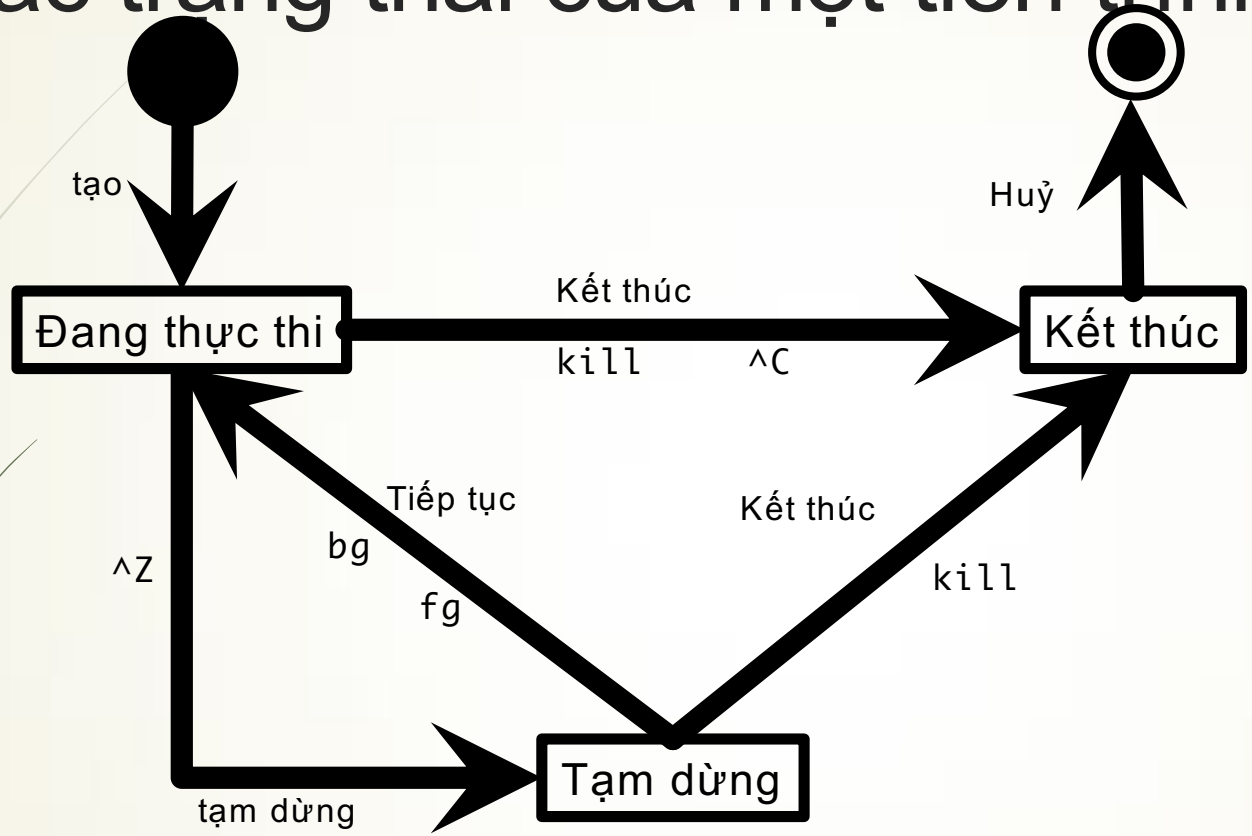
  - `kill` buộc một tiến trình kết thúc bằng cách gởi tín hiệu 9 để huỷ tiến trình (`SIGKILL`).

`killall` gởi một tín hiệu đến tất cả các tiến trình.

# Tạo tiến trình

- Một cách duy nhất để tạo tiến trình: bằng cách nhân đôi (clone).
- Một tiến trình tạo ra tiến trình khác gọi là tiến trình cha, tiến trình mới tạo gọi là tiến trình con.
- Tiến trình con kết thừa một bản copy dữ liệu của cha, và chương trình đang thực thi và tiếp tục.
- Không chia sẻ dữ liệu giữa các tiến trình: mỗi tiến trình làm việc độc lập với nhau, không chia sẻ với tiến trình khác.

# Các trạng thái của một tiến trình



# Tiến trình ở foreground và background

- Mặc định: một tiến trình thực thi ở foreground
  - Các tiến trình cha và con thực thi tuần tự.
  - Mỗi lần chỉ thực hiện một lệnh duy nhất.
- Một lệnh cũng có thể thực thi ở background
  - Sử dụng ký tự "&" ở cuối lệnh
  - Hai tiến trình, cha và con thực thi đồng thời.

# Tiến trình con

- Một  subshell là một tiến trình được chạy bởi shell
- Thực thi các lệnh trong một subshell: (commands)
- Một subshell cho phép cách ly các thao tác. Ví dụ:

```
$ pwd
/home/nguyen/unix
$ (cd .. ; pwd)
/home/nguyen
$ pwd
/home/nguyen/unix
```

- Vòng đời của một subshell không vượt quá ")" :

```
$ (cd ..) ; (pwd)
/home/nguyen/unix
```

# Nội dung

1. Tiến trình trong Linux
2. **Các biến môi trường**
3. Here document
4. Các lệnh liên quan đến pipe
5. Tính toán số học

# Cấu hình shell

- Shell sử dụng các file cấu hình. Những file này được thực thi khi khởi động shell.

`/etc/profile`

  - Một file duy trì bởi admin để cấu hình các mặc định cho tài khoản trên một máy.

`~/.bash_profile`, `~/.bash_login`, `~/.bashrc` et `~/.profile`

  - Các file được duy trì bởi người dùng để tham số hoá tài khoản của họ.

# Các biến môi trường

- Để xem danh sách biến môi trường: lệnh `printenv`

```
$ printenv
TERM_PROGRAM=Apple_Terminal
SHELL=/bin/bash
PATH=/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin ...
HOME=/Users/tuyennguyen1
LOGNAME=tuyennguyen1
USER=tuyennguyen1
PWD=/Users/tuyennguyen1/testUnix
LOGNAME=tuyennguyen1
...
```

# Một số biến được định nghĩa sẵn

- HOME: thư mục đăng nhập
- PATH: danh sách các thư mục tìm kiếm lệnh cách nhau bởi « : »
- PS1: dấu nhắc lệnh  (mặc định : « $ »)
- PS2: dấu nhắc lệnh tiếp tục (mặc định: « > »)
- IFS: các dấu ngăn cách các từ
- PWD: thư mục hiện hành

# Xuất biến

- Các biến môi trường được thao tác như các biến. Ví dụ:

```
echo "$PWD" ; PWD="/etc"
echo "$PWD"
```

- Tạo một biến môi trường :

```
export var_name[=value]
```

# Nội dung

1. Tiến trình trong Linux
2. Các biến môi trường
3. **Here document**
4. Các lệnh liên quan đến pipeline
5. Tính toán số học

# here document

Điều hướng đặc biệt:

```
<< finalword
...
finalword
```

# Các thay thế

- Các thay thế được thực hiện trong here document.

```
$ cat << STOP
> My home directory is $HOME
> STOP

My home directory is /Users/tuyennguyen1
```

# Sinh ra file script

```
$ cat >| myscript.sh << END
#! /bin/bash
# Script généré le $(date)
echo "Il y a \$# paramètres"
exit 0
END

$ cat myscript.sh
#! /bin/bash
# Script généré le Fri May 20 21:57:30 ICT 2016
echo "Il y a $# paramètres"
exit 0

$ chmod +x monscript.sh

$ ./monscript.sh a b
Il y a 2 paramètres
```

# Sinh ra một web page

```
titre="Ma page web"
cat >| mapage.html << FINPAGE
<html>
    <head>
   <title>$titre</title>
    </head>
    <body>
   <h1>$titre</h1>
   <p>Le répertoire $(pwd) contient :</p>
   <ul>
        $(for f in * ; do
       echo "      <li><a href=\"$f\">$f</a></li>"
        done
        )
   </ul>
    </body>
</html>
FINPAGE
```

# Tự động hoá việc nhập liệu

- Giả sử có script naissance.sh :

```
#! /bin/bash
echo -n "Jour ? " ; read jour
echo -n "mois ? " ; read mois
echo –n "An ?"    ; read an
echo "Date naissance : $jour/$mois/$an"
exit 0
```

./naissance.sh << END
25
12
1901
END

# Lệnh <<<

- Cho phép điều hướng một chuỗi trên đầu vào chuẩn.

```
<<< "text line"
```

- tương đương với

```
<< END
 text line
END
```

- Ví dụ :

```
cat <<< "Bonjour"              # echo "Bonjour"
sort -r > tmp <<< "$(ls)"      # ls | sort -r > tmp
```

# Nội dung

1. Tiến trình trong Linux
2. Các biến môi trường
3. Here document
4. **Các lệnh liên quan đến pipeline**
5. Tính toán số học

# Lệnh kết nối với pipe

- Mỗi lệnh kết nối với một pipe được chạy bởi một subshell
- Biến được thay đổi giá trị quanh một pipe ➔ mất dữ liệu cho shell

```
$ a=1 ; a=2 | a=3 ; echo "$a"
1
```

- Đọc một kết quả :

```
$ grep page mapage.html | wc -l | read n ; echo "$n"
$ n=$(grep page mapage.html | wc -l) ; echo "$n"
3
```

- Đọc nhiều giá trị :

```
echo 1 2 3 | read a b c    # that bai
read a b c <<< $(echo 1 2 3)
```

# Xử lý từng dòng

- Để xử lý từng dòng đầu ra của một lệnh :

```
command | while read line
do
    process $line
done
```

Khối lệnh `while .. do .. done` được thực hiện trong một subshell

→ các biến trong `do .. done` không xuất ra giá trị.

- **Giải pháp** (nếu cần những biến này) : trả về một pipe

```
while read line
do
    process    $line
done <<< "$(command)"   # "" important
```

# Ví dụ : Đếm dòng

```
n=0
ls -l | while read line
do
   n=$(expr $n + 1)
done
echo "$n"                  # 0
```

- **Giải pháp :**

```
n=0
while read line
do
   n=$(expr $n + 1)
done <<< "$(ls -l)"
echo "$n"                  # 41 ok
```

# Nội dung

1. Tiến trình trong Linux
2. Các biến môi trường
3. Here document
4. Các lệnh liên quan đến pipeline
5. **Tính toán số học**

# Ước lượng số học

- Các biểu thức số học trong bash :
  - chỉ với số nguyên
  - toán tử và cú pháp của ngôn ngữ C
  - biểu thức được xử lý như thể được đặt trong " ", nghĩa là không cắt dựa trên các khoảng trắng
  - ta có thể bỏ qua $ nếu không nhập nhằng
- Hai hình thức : (( )) và $(( ))

# Thay thế số học

```
$((expression))
```

được thay thế bởi giá trị của biểu thức.

```
$ echo $((20+30))
50
$ x=20 ; y=30 ; echo $(($x+$y))
50
```

- Ta có thể bỏ qua $ :

```
$ echo $((x+y))
50
```

... ngoại trừ trường hợp nhập nhằng :

```
$ set 5 ; echo $((x+$1))
25
```

# Tính toán

- Tôn trọng độ ưu tiên :

```
$ echo $((10-3*(4+5)))
-17
```

- Phép chia nguyên :

```
$ echo $((17/7))
2
$ echo $((17.0/7))
bash: 17.0/7: syntax error: invalid arithmetic
operator (error token is ".0/7")
```

- Đối với số thực, sử dụng lệnh bc bc :

```
$ echo "scale=8; 17/7" | bc
2.42857142
$ bc <<< "scale=4; sqrt(2)"
1.4142
```

# Biểu thức C

- Phép toán logic → 0 hoặc 1

```
$ echo $((10==10))
1
```

- Toán tử bậc 3:

```
$ echo $((3>=5 ? 3:5))
5
```

- Biểu thức với các dấu phẩy: ước lượng từ trái sang phải, giá trị bên phải

```
$ echo $((x=y=4, y++, x+y))
9
```

# Ví dụ với hàm

```
add()    # a b
{
    local a="$1" b="$2"
    local f=$((a+b))
    echo $f
}
```

# Những phần bổ sung so với C

- Tính luỹ thừa :

```
$ echo $((2**5))
32
```

- Hệ đếm :

```
$ echo $((0100))      # hệ 8
64
$ echo $((0x100))     # hệ 16
256
$  echo $((2#100))    # hệ 2       CGNU:0b100
4

$  k=5; echo $(($k#100))

25
```

# Giá trị trả về của một biểu thức

((expression)) thành công nếu biểu thức đúng, nghĩa là ≠0

```
$ ((1)) ; echo $?          # đúng
0                          # thành công
$ ((0)) ; echo $?          # sai
1                          # thất bại
```

- Sử dụng :

```
if ((expression)); then .. ; fi
while ((expression)); do .. ; done
fonction() { .. ; ((expression)) ;}
```

# Ví dụ

```
((x = 3, y=x+2))
((x++)) ; y=$((y+1))
if (($#<2)); then
    echo "2 arguments" > /dev/stderr ; exit 1
fi
is_positive()    # x
{
    local x="$1"
    ((x>=0))
}
if is_positive "$y"; then .. ; fi

i=10
while is_positive "$i"; do echo "$i"; ((i--)); done
```

# Vòng lặp của C

```
for ((expr1; expr2; expr3)); do .. ; done
```

- Giống C, tương đương với

```
((expr1))
while ((expr2)); do
    ..
    ((expr3))
done
```

- Ví dụ :

```
$ for ((i=0,j=5;i<j;i++,j--)); do echo "$i $j"; done
0 5
1 4
2 3
```

# Câu hỏi ?