040613479 UNIX Tools

Lecture 6: Process

Topic

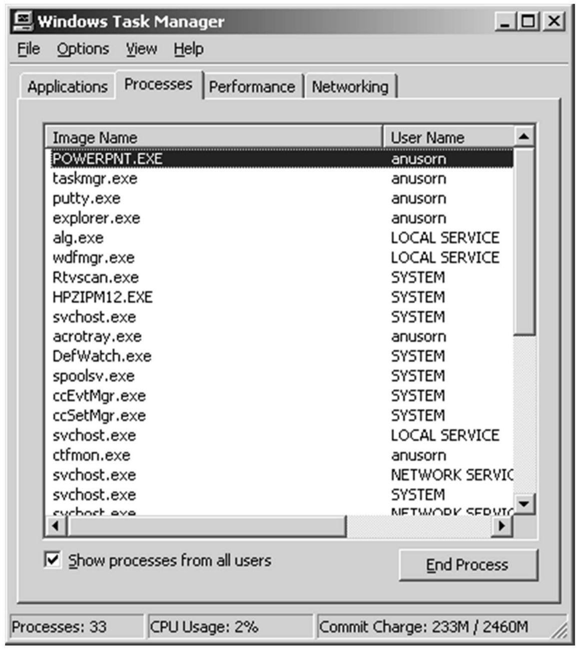
- Overview
  - คืออะไร? เกิดขึ้นตอนไหน?
- วงจรการทำงานของโปรเซส
- Parent-child Relationship
  - Process Attributes
  - ps demo
- Background vs. Foreground Process
- Job control
  - State Diagram
- crontab

ตัวอย่างสถานการณ์

- มีหลายคำสั่ง เช่น
  - "เขียนสคริปต์ไป พร้อมกับค้นหาไฟล์..."
  - "เก็บสถิติว่ามีการเข้าใช้ช่วงเวลาใดมากที่สุด..."
  - "ต้องคอมไพล์โปรแกรม แต่ติดธุระไม่สามารถอยู่ตรวจสอบได้..."
- แนวทาง
  - ทำทีละคำสั่ง
  - เชื่อมด้วย ; ()
  - script
  - fg bg
  - crontab

Overview

- ป้อนคำสั่ง-> เชลล์ตีความ->ประมวลผล-> แสดงผลลัพธ์-> ปรากฏ Prompt
- โปรเซส (Process) ที่ผ่านมา
  - การป้อนคำสั่ง
  - Shell Prompt - bash
- โปรเซส vs. โปรแกรม
  - ผู้ใช้ 2 คนเรียกใช้คำสั่งเดียวกัน -> หนึ่งโปรแกรมแต่มีสองโปรเซส
  - เรียกใช้ shell script
  - คำสั่งที่ป้อนอาจเชื่อมโดย piping
- Parent-child relationship
  - Process Status- ps Command
- Foreground or background Job/process
  - Job Control



Windows Task Manager

File Options View Help

Applications | Processes | Performance | Networking

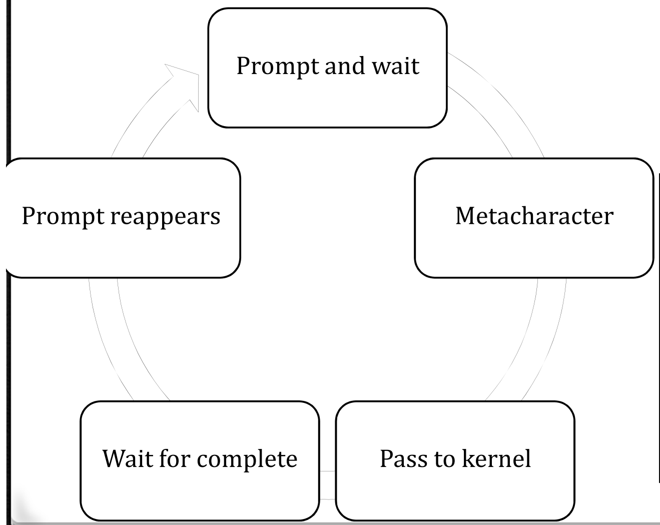
Image Name	User Name
POWERPNT.EXE	anusorn
taskmgr.exe	anusorn
putty.exe	anusorn
explorer.exe	anusorn
alg.exe	LOCAL SERVICE
wdfmgr.exe	LOCAL SERVICE
Rtvscan.exe	SYSTEM
HPZIPM12.EXE	SYSTEM
svchost.exe	SYSTEM
acrotray.exe	anusorn
DefWatch.exe	SYSTEM
spoolsv.exe	SYSTEM
ccEvtMgr.exe	SYSTEM
ccSetMgr.exe	SYSTEM
svchost.exe	LOCAL SERVICE
ctfmon.exe	anusorn
svchost.exe	NETWORK SERVIC
svchost.exe	SYSTEM
svchost.exe	NETWORK SERVIC

☑ Show processes from all users

End Process

Processes: 33 | CPU Usage: 2% | Commit Charge: 233M / 2460M

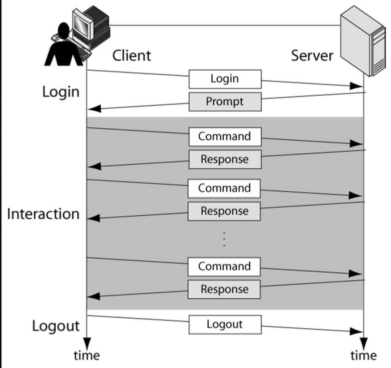
Shell interpretive cycle



Execution Steps

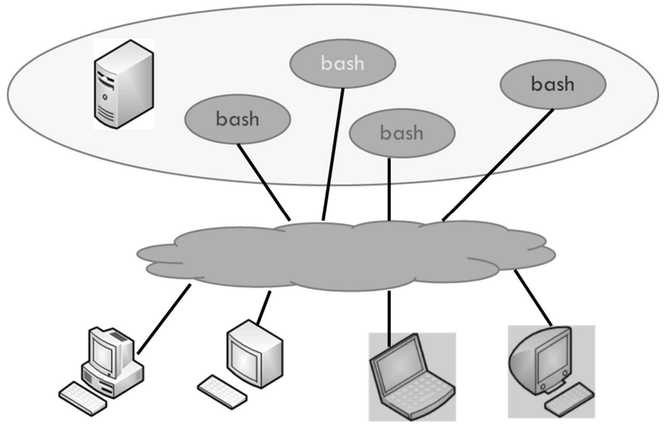
1. Command parsing
2. Variable evaluation
3. Command substitution
4. Redirection
5. Wildcard expansion
6. Path determination

วงจรการทำงานของโปรเซส



1. ป้อนคำสั่ง
2. แม่ออกลูก (fork) แม่รอ (wait)
3. ลูกทำงาน (exec) สร้างสภาพแวดล้อมเพื่อทำคำสั่งที่ผู้ใช้ป้อน
4. ลูกทำงานเสร็จ (exit) ->exit status ($?)
5. แม่ตื่น (wake up) มาทำงานต่อ

Multitasking



What is a Process?

- Program + x permission
  - Machine readable code (binary) that is stored on disk
- Process
  - A program that is loaded into memory and executed
- The kernel (OS) controls and manages processes
  - It allows multiple processes to share the CPU (multi-tasking)
  - Manages resources (e.g. memory, I/O)
  - Assigns priorities to competing processes
  - Facilitates communication between processes
  - Can terminate (kill) processes

The Process Hierarchy

- Processes are associated in parent-child relationships
  - A process can create another process and therefore become the "parent" of the created process. The created process becomes the "child".
  - A process can have multiple children, but every child can only have one parent.
  - The "family tree" of processes on the system constitute the process hierarchy
  - A child process inherits various characteristics from its parent at creation time
    - Real UID and real GID
    - Effective UID and effective GID
    - Current working directory
    - File descriptor of parent process
    - Environments variables
  - PID, PPID

The real UID is the UID of the user who started the process

The effective UID is the UID that is used when checking user privileges of the process

Process Status Command: ps

```
$ ps -x
  PID  TT  STAT     TIME COMMAND
54616  ??  S     0:00.38 sshd: aws@ttyp0 (sshd)
54889  ??  I     0:00.02 sshd: aws@ttyp2 (sshd)
54617  p0  Ss    0:00.18 -bash (bash)
54897  p0  R+    0:00.01 ps -x
54890  p2  Is+   0:00.04 -bash (bash)
54892  p2  T     0:00.01 ping 202.44.40.1
54911  p2  D     0:03.11 find / -name *.c

$ ps
$ ps -l
```

ps demo

- Process Hierarchy/ Family Tree
  - Parent-child Relationship - pid, ppid
  - pstree command
- Process Attributes
- Basic Output Fields

Fields	Meaning
PID	process identification number
TT	controlling terminal of the process
STAT	state of the job
TIME	amount of CPU time the process has acquired so far
COMMAND	name of the command that issued the process

STATe

- D - a process in disk (or other short term, uninterruptible) wait
- I - an idle process (sleeping for longer than about 20 seconds)
- J - a process which is in jail(2)
- R - a runnable process
- S - a process that is sleeping for less than about 20 seconds
- T - a stopped process
- Z - a dead (zombie) process
- "+" symbol indicates are foreground processes,
- "s" indicates are session leaders

```
[aws@CSUnix ~]$ id
uid=1001(aws) gid=0(wheel) groups=0(wheel)
[aws@CSUnix ~]$ ps -auo uid,ruid
USER       PID %CPU %MEM   VSZ  RSS TT  STAT STARTED    TIME COMMAND           UID RUID
root      1679  0.0  0.1 14420 1912 v0  Is+  18Sep14 0:00.00 /usr/libexec/get    0    0
root      1680  0.0  0.1 14420 1912 v1  Is+  18Sep14 0:00.00 /usr/libexec/get    0    0
root      1                                                               get    0    0
root      1                                                               get    0    0
root      1                                                               get    0    0
root      1                                                               get    0    0
root      1                                                               get    0    0
root      1                                                               get    0    0
cs543023 96002  0.0  0.2 17576 3376  0  Is+   3:56PM 0:00.01 -bash (bash)     1053 1053
cs543023 96098  0.0  0.2 17576 3380  1  Ss+   4:36PM 0:00.04 -bash (bash)     1053 1053
cs543023 96101  0.0  0.2 23368 3892  1  T     4:36PM 0:00.01 vi xFile         1053 1053
cs543023 96107  0.0  0.2 23368 3892  1  T     4:40PM 0:00.01 vi xFile         1053 1053
aws      96221  0.0  0.2 17576 3408  2  Ss    5:01PM 0:00.04 -bash (bash)     1001 1001
aws      96319  0.0  0.1 16588 2088  2  R+    5:14PM 0:00.00 ps -auo uid,ruid 1001 1001
aws      96303  0.0  0.2 17576 3408  3  Is    5:12PM 0:00.01 -bash (bash)     1001 1001
root     96309  0.0  0.1 41388 2420  3  I+    5:12PM 0:00.00 passwd              0 1001
[aws@CSUnix ~]$
```

Sometimes processes need to access resources (like files) that they do not have ownership of. If they require such access, the effective UID changes from their *real* UID to the UID of the user who owns that resource

The real UID is the UID of the user who started the process. The effective UID is the UID that is used when checking user privileges of the process

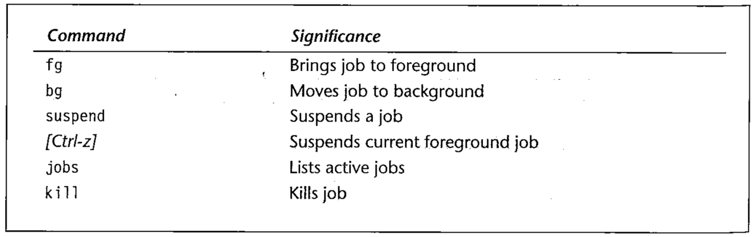
Effective UID is usually equal to the real UID

Setuid binaries are a special case the effective UID can differ from the real UID

Job control commands

- In Unix a <u>group of processes constitutes a job</u>
- Unix allows users to control jobs from the terminal
- Only one job at a time may be in the **foreground**
- The multiple jobs may run in the **background**
- User control any Job by Command and Job id.
  - Job Level or Process Level

TABLE 8.3 *Job Control Commands*



Command	*Significance*
`fg`	Brings job to foreground
`bg`	Moves job to background
`suspend`	Suspends a job
[Ctrl-z]	Suspends current foreground job
`jobs`	Lists active jobs
`kill`	Kills job

Foreground VS. Background Process

- เป็นไปตาม...วงจรการทำงานของโปรเซส...แต่
- *<u>A foreground process</u>* is different from a *<u>background process</u>* in two ways:
  - Some foreground processes show the user an interface, through which the user can interact with the program.
  - The user must wait for one foreground process to complete before running another one.
- To start a foreground process, enter a command at the prompt, e.g.,
  - $ *command1*
- The next prompt will not appear until command1 finishes running.

http://www.tldp.org/LDP/Linux-Dictionary/html/index.html

Foreground Job/Process

- In a multitasking operating system, such as UNIX/Linux, the foreground process is the program that the user is interacting with at the present time (for example, data entry).
- Different programs can be in the foreground at different times, as the user jumps between them. In a tiered windowing environment, it is the topmost window.

Background Job/Process

- A program that is running without user input. A number of background processes can be running on a multitasking operating system, such as UNIX /Linux, while the user is interacting with the foreground process (for example, data entry).
- Some background processes daemons, for example never require user input. Others are merely in the background temporarily while the user is busy with the program presently running in the foreground.

Foreground Job Control

Suspending and Restarting a Foreground Job

```
$ fgLoop.scr
^z                                          #foreground job
  suspended
[2] + Stopped (SIGTSTP)        fgLoop.scr
$ date
Tue Sep 12 12:43:44 PDT 2000
$ fg                                        #resume job
fgLoop.scr
------------------------------------------------------------
```

Terminating a Foreground Job

```
$ fgLoop.scr
^z                                          # Suspend job
[2] + Stopped (SIGTSTP)        fgLoop.scr
$ fg                                        # Restart job
fgLoop.scr
^c                                          # Cancel job
<return>
$
```

Background Job Control

- ใช้ Ampersand (&)
- ระวัง Standard stream
- Default Job

Starting a Background Job

```
$ longJob.scr&
[1]     1728406
------------------------------------------------------------
```

Suspending, Restarting, and Terminating a Background Job

```
$ longJob.scr&
[1]     1795841
$ stop %1
[1] + 1795841 Stopped (SIGSTOP)        longJob.scr&
$ bg %1
[1]     longJob.scr&
$ kill %1
[1] + Terminated               longJob.scr
```

Job Control

Moving a Job Between Foreground and Background

```
$ longJob.scr                     # Start long running job in fg
                                  # Job is running in foreground.
^z                                # Suspend job
[1] + Stopped (SIGTSTP) longJob.scr
$ bg                              # Move job to background.
[1]     longJob.scr&
$ fg %1                           # Bring active job to foreground
longJob.scr                       # Job running in fg
```

Multiple background jobs

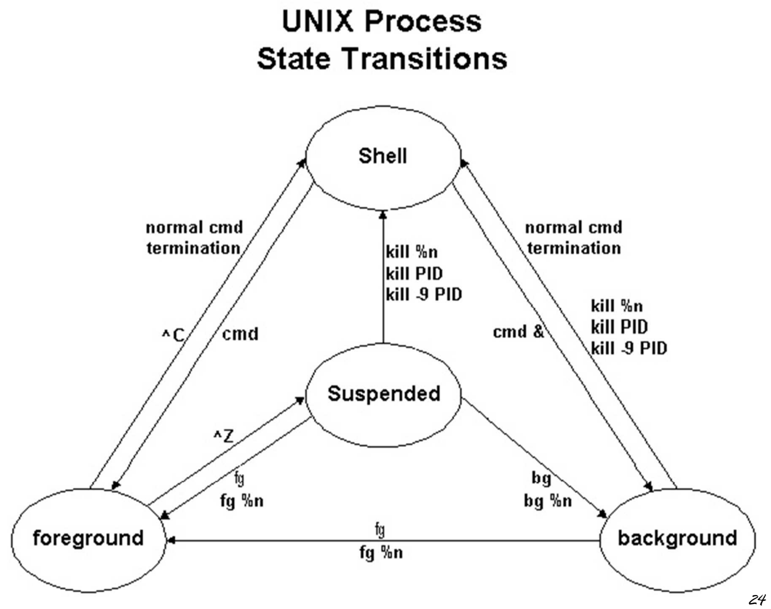
ควบคุม **Standard Stream** ให้ดี...

```
$ jobs
[4] + Stopped (SIGTSTP)         longJob.scr
[3] -  Running                  bgCount200.scr&
[2]    Running                  bgCount200.scr&
[1]    Running                  bgCount200.scr&

$ bgCount200.scr: 2000                  # Message from job [1]
bgCount200.scr: 800                     # Message from job [3]
bgCount200.scr: 1600                    # Message from job [2]
bgCount200.scr: 2200                    # Message from job [1]
bgCount200.scr: 1000                    # Message from job [3]
bgCount200.scr: 1800                    # Message from job [2]
bgCount200.scr: 2400                    # Message from job [1]
```

ps demo Again…

- Foreground and background Process
  - Standard Input
  - Standard Output
  - Standard Error
- From….Family Tree
  - When a process become to Orphan?
  - “nohup” Command



ความต่างระหว่าง Job id vs. pid

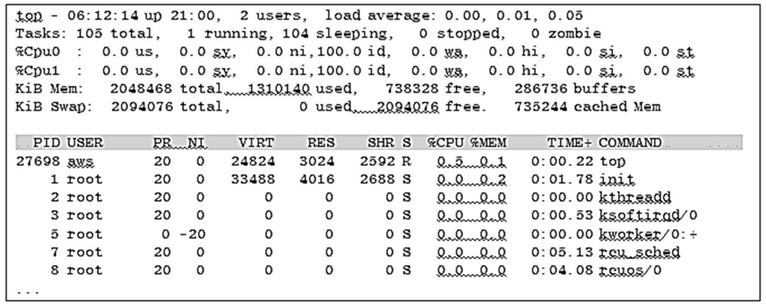
Starting a Background Job

```
$ longJob.scr&
[1]     1728406
```

PID Command Output

```
$ ps
        PID TTY     TIME CMD
    2229478 ttyq0   9:44 bash
    2229618 ttyq0   9:27 bash
    2247678 ttyq0  10:55 bash
    2209680 ttyq0   9:42 sh
```

top Command



```
top - 06:12:14 up 21:00,  2 users,  load average: 0.00, 0.01, 0.05
Tasks: 105 total,   1 running, 104 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   2048468 total,  1310140 used,   738328 free,   286736 buffers
KiB Swap:  2094076 total,        0 used,  2094076 free.   735244 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
27698 aws       20   0   24824   3024   2592 R   0.5  0.1   0:00.22 top
    1 root      20   0   33488   4016   2688 S   0.0  0.2   0:01.78 init
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.53 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:+
    7 root      20   0       0      0      0 S   0.0  0.0   0:05.13 rcu_sched
    8 root      20   0       0      0      0 S   0.0  0.0   0:04.08 rcuos/0
...
```

- คำสั่ง top ทำหน้าที่แสดงรายการสถานะของโปรเซสในหน่วยความจำในปัจจุบันแบบ Real-Time โดยจะเรียกข้อมูลทุกๆ ช่วงเวลาที่ตั้งไว้ ซึ่งต่างจากคำสั่ง ps ที่แสดงข้อมูลในปัจจุบันเท่านั้น กล่าวได้ว่า top ทำงานในลักษณะ Dynamic ส่วน ps ทำงานในลักษณะ Static

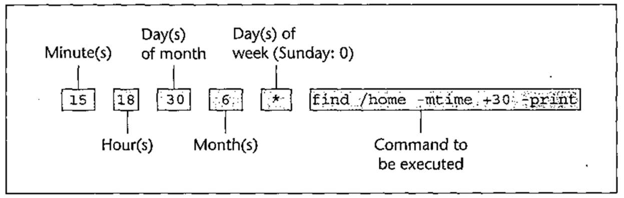
top Command Example

$ vi ^z	*งานที่ 1*
$ ping cloud.google.com >ping.log &	*งานที่ 2*
$ find / -mtime +30 >find.log ^z	*งานที่ 3*

`crontab` command

- ตั้งเวลาการทำงานของคำสั่ง
- cron daemon
- crontab entry
  - 00-10
  - 3, 6
  - Override
- ผลลัพธ์จากการทำงานจะส่ง mail
- ทำไมจึงไม่ส่งออกจอภาพ? ถ้าต้องการให้ออกจอภาพทำอย่างไร?

FIGURE 8.2 *The Components of a crontab Entry*



Sample crontab entry (first 5 fields only)

	First 5 fields only	Match
1.	00-10 17 * * *	
2.	00-10 17 * 3, 6, 9, 12 *	
3.	00-10 17 10,20,30 * *	
4.	00-10 17 * * 1, 3	
5.	00-10 17 * 3, 6, 9, 12 1, 3	
6.	00-10 17 10,20,30 * 1, 3	
7.	0, 30 * * * *	
8.	0 0 * * *	
9.	55 17 * * 4	
10.	30 0 10, 20 * *	
11.	00, 30 09-17 * * 1-5	

ตั้งเวลากัน

- per-user crontab files => /var/cron/tabs/*
- crontab
  - view (-l), remove (-r) or edit (-e)

- ตั้งเวลาให้ echo “my first crontabs at <time>” ....
- ไม่แสดงผลลัพธ์ที่ standard output.....why?
- ให้เก็บลงไฟล์ ทำอย่างไร?
- “ทุก 30 นาที ตั้งแต่เวลา 9-15 ในวันจันทร์ที่ 19 กันยายน”

Quiz 2

- 26 Sep. 2016 : 9.30 AM
- Shell and Process : Lab 5 , 6 ,7