Server Operating Systems

Lecture 10

Processes and Memory

System Processes

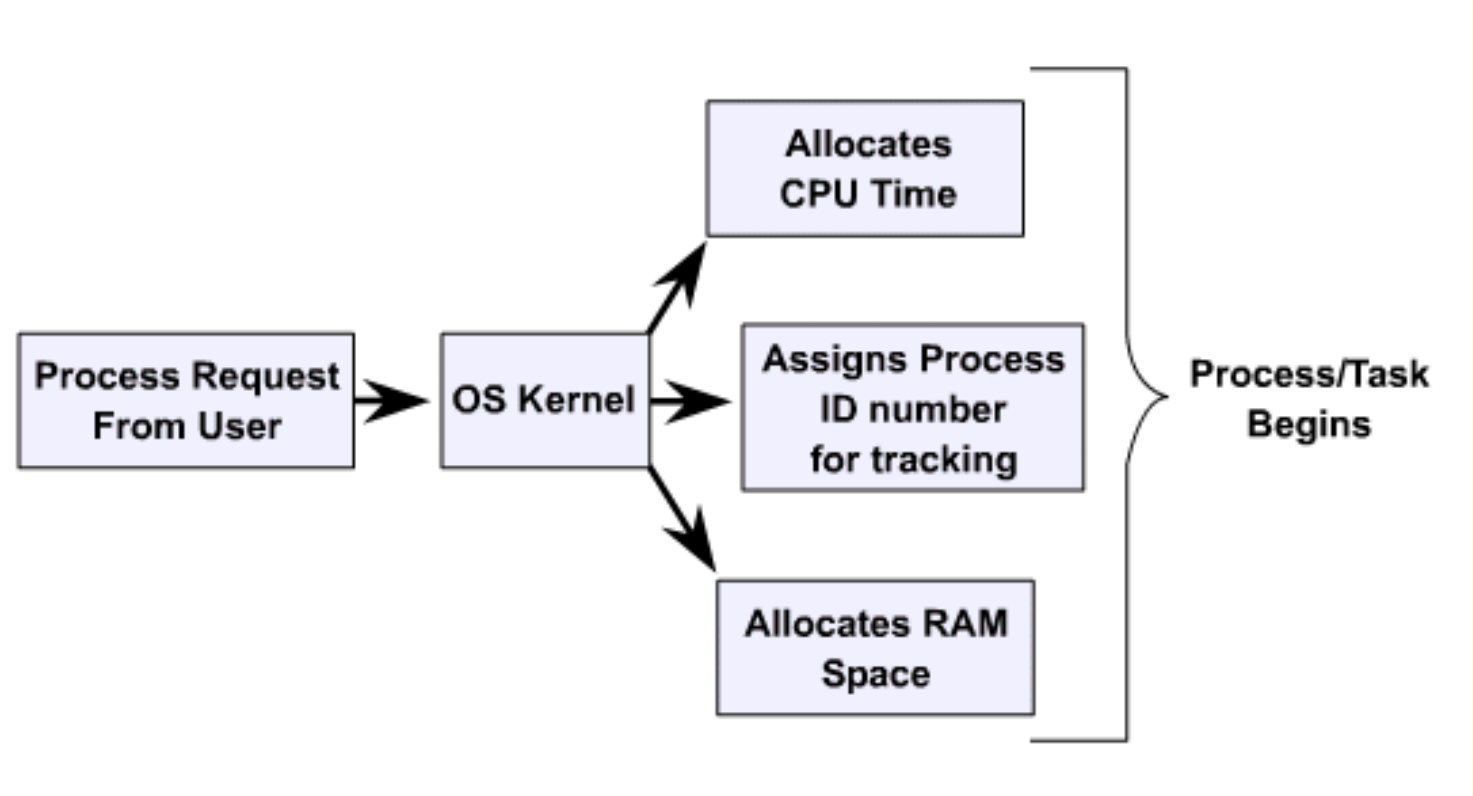
UNIX manages tasks using ***processes***

Each program creates a process which is assigned a unique ***process identification number*** **(PID)**

Process can **spawn** a subprocess, thus creating a process hierarchy with parent / child relationships

Some simple commands, such as **cd**, are executed by the shell itself and do not create a separate process

Kernel Allocation for Processes



Upon boot, first two processes started

sched (scheduler)(pid 0)

init (initialization)(pid 1) which manages other processes

Types of Processes

Daemon

- processes that exist for a specific purpose
- (**lpsched daemon)** exists for the sole purpose of handling print jobs
- like NT services, running inactive in the background until needed

Parent

- process which spawns another process
- Following boot-up, a process called **init** daemon is invoked
- Every process, except init, has a parent process

Child

- process spawned by another process
- When working in a terminal window, that terminal's PID is the *parent process ID* (PPID) of any commands issued in the terminal
- These commands are child processes of the terminal process

Types of (troubled) Processes

Orphan

- child process is running and parent is killed
- system passes the orphan process to init which then becomes the parent process and terminates it

Zombie (or Defunct)

- a child process does not return to the parent process with its output
- process becomes “lost” in the system
- The only resource this process uses is a slot in the process table; it cannot be stopped in a conventional manner
- The only way to kill a zombie is to reboot the system

ps Command

$ ps [-options]

Option	Meaning	Function or Purpose
`ps`	No Options	Display information for current user processes in current shell or terminal window
`ps -e`	Every	Display information about every process on the system.
`ps -f`	Full	Generate a full listing with all available information on each process.
`ps -u userid`	User	Display all processes for a particular user

Used to check the PID and then **"kill"** the process if it is taking too long or has stopped

ps Command Output

Terminal

Window Edit Options Help

```
$ ps -ef | more

UID      PID   PPID   C      STIME  TTY      TIME   CMD
root       0      0   80  16:46:41  ?        0:01   sched
root       1      0   80  16:46:44  ?        0:40   /etc/init -
root       2      0   27  16:46:44  ?        0:00   pageout
root       3      0   80  16:46:44  ?        4:33   fsflush
root     236      1   80  16:48:08  ?        0:01   /usr/lib/saf/sac
root     844      1   54  12:12:10  ?        0:00   /usr/lib/lpsched
aster   1292      1   80  06:48:51  console  0:01   -ksh
root     241    236   69  16:48:14  ?        0:01   /usr/lib/saf/ttymon
rose    1400    321   80  20:03:11  ?        0:01   /usr/openwin/bin/clock
```

UID	The user ID of the user that initiated the process.
PID	The process identification number of the process. The PID is used to kill a process.
PPID	The parent process identification number of the process
C	The priority of the process
STIME	Start time for the process
TTY	Terminal type - the controlling terminal for the process
TIME	The amount of CPU time used by the process
CMD	The command name or daemon (name of the program executed)

ps Output Piped to grep

```
$ ps -e | grep lp
  225?0:01lpNet
  217?0:01psched
  260?0:01lpNet
$
$ pgrep -l lp
  225  lpNet
  217  lpsched
  260  lpNet
$
```

ps –ef listing can be quite long

By piping the output of the ps command through grep you can search for the specific process you want to terminate and determine the correct PID

pgrep command used to search for a specific process. The -l (long output) option will display the PID and names of the processes found

Parent Child Processes

```
$ ps -f
UID      PID    PPID    C      STIME   TTY      TIME    CMD
user2    785     742    0   16:46:41   pts/6    0:00    sleep 500
user2    742     689    0   16:40:27   pts/6    0:00    csh
user2    689     627    0   16:37:15   pts/6    0:00    ksh
```

First identity the PID of the lowest-level unresponsive process

Sometimes necessary to kill the Parent of process and on rare occasions even the Parent of the Parent

Killing a parent process will kill all child processes spawned by it

Look at the output to be able to trace from the child up the hierarchy to the parent processes that spawned them

Terminating Processes

$ kill -15 - soft kill

$ kill -9 - sure kill

```
$ ps
  PID    TTY        TIME     CMD
12892    console    0:01     ksh
12932    pts/0      0:01     find
12935    pts/1      0:00     ps
 <Output omitted>

$ kill 12932
[1] + Terminatedfind / -name core -exec rm {} \; &
$
```

$ pkill cmd name -

Saves you having to look up the PID

```
$ sleep 100&
$ ps
472     pts/3   0:01    ksh
12418   pts/3   0:00    sleep

$ pkill sleep
$ ps
472     pts/3   0:01    ksh
```

KDE System Guard

Process Table [modified] - KDE System Guard

File Edit Settings Help

Sensor Browser | Sensc

localhost

System Load | Process Table

localhost: Running Processes

Search: User Processes

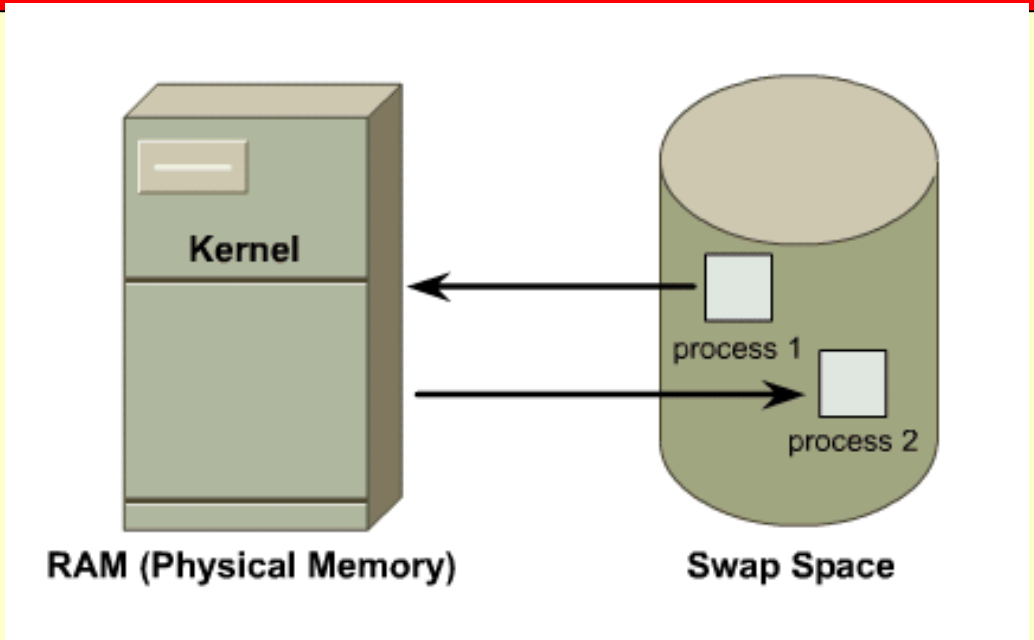
Name	PID	Status	User%	System%	Nice	VmSize	VmRss	Login	Command
init	1	sleeping	0.00	0.00	0	808	312	root	init [5]
dbus-daemon	2489	sleeping	0.00	0.00	0	14,920	1,088	messagebus	/bin/dbus-daemon
hald	2656	sleeping	0.00	0.00	0	31,348	4,640	haldaemon	/usr/sbin/hald
hald-runner	2657	sleeping	0.00	0.00	0	15,492	1,204	root	hald-runner
hald-addon-acpi	2885	sleeping	0.00	0.00	0	24,888	1,116	haldaemon	hald-addon-acpi: lister
kdm	2920	sleeping	0.00	0.00	0	28,260	752	root	/opt/kde3/bin/kdm
kdm	2941	sleeping	0.00	0.00	0	68,468	2,084	root	-:0
kde	3409	sleeping	0.00	0.00	0	14,620	1,664	strider	/bin/sh
ssh-agent	3472	sleeping	0.00	0.00	0	48,928	864	strider	/usr/bin/ssh-agent
kwrapper	3528	sleeping	0.00	0.00	0	3,716	440	strider	kwrapper
avahi-daemon	3065	sleeping	0.00	0.00	0	27,508	1,632	avahi	avahi-daemon: running
gpg-agent	3471	sleeping	0.00	0.00	0	15,936	520	strider	/usr/bin/gpg-agent
dbus-launch	3475	sleeping	0.00	0.00	0	19,612	808	strider	/usr/bin/dbus-launch
dbus-daemon	3476	sleeping	0.00	0.00	0	6,636	440	strider	/bin/dbus-daemon
start_kdeinit	3514	sleeping	0.00	0.00	0	3,584	160	strider	start_kdeinit
kdeinit	3515	sleeping	0.00	0.00	0	110,744	7,220	strider	kdeinit Running...
klauncher	3520	sleeping	0.00	0.00	0	117,636	6,940	strider	klauncher [kdeinit] --n
kwin	3532	sleeping	0.00	0.00	0	136,816	13,084	strider	kwin [kdeinit] -session

Tree | Refresh | Kill

131 Processes | Memory: 449,452 KB used, 1,613,584 KB free | Swap: 0 KB used, 5,116,692 KB free

Start menu > system > monitor > Ksysguard

Physical Memory & Swap Space



When the system starts, RAM is empty, it then loads the **kernel**

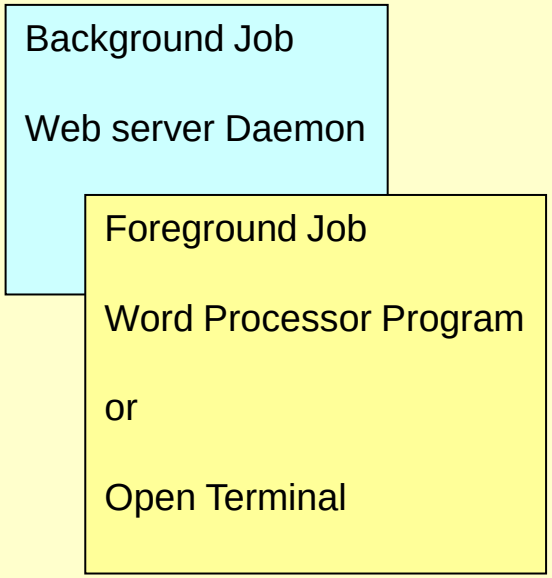
UNIX sets aside an area on the hard disk called **swap space**

Programs are copied from the hard disk into system RAM and divided into **pages**

Programs not recently used are paged out to the swap area of the hard disk to allow more programs to fit into RAM

RAM + swap space = virtual memory

Foreground and Background



Foreground Job

The one that the user is currently communicating with. They must wait until the current one has finished before they can issue another command.

Background Job

Running without communicating directly with user. Place ampersand at end of command.

e.g. if you have a webserver program called httpd

```
$ httpd&
```

will cause it to run in the background, and allow the user to issue another command immediately.

Foreground / Background Commands

Command	Meaning
`Ctrl-z or stop %job#n`	Suspends (not terminates) the foreground process.
`jobs`	Displays all background jobs.
`bg %job#n`	Places job# in the background and restores the shell prompt.
`fg %job#n`	Places job# in the foreground.
`Ctrl-c`	Cancels the current foreground job.
`kill %job#n`	Terminates job#.

Example

```
$sleep 500&
$jobs
[1] + Running sleep 500
$fg %1
$sleep 500
^Z
[1] + Stopped sleep 00
$jobs
[1] + Stopped sleep 500
$bg %1
[1] + Running sleep 500
$kill %1
[1] Terminated sleep 500
$jobs
$
```

```
$ sleep 500&

$ jobs
 [1] + Running sleep 500

$ fg %1

$ sleep 500
^Z
[1] + Stopped sleep 500

$ jobs
[1] + Stopped sleep 500

$ bg %1
[1] + Running sleep 500

$ kill %1
[1] Terminated sleep 500

$ jobs

$
```

The at Command

The **at** command enables the student to run programs, commands, or shell scripts at some future date and time.

For example, the student would like to e-mail a friend the contents of a file on January 1 at exactly 12:01 p.m./1201, using the

mail *filename* command.

Or, to be considerate, the student schedules the **find** command to search the entire hard drive for a missing file to run at 3:00 a.m./0300, when user activity is low.

To schedule a one time job using the at **command, perform these steps:**

Enter the **at** command followed by a time specification.

eg

- $at 10:30am today
- $at midnight
- $at 12:01 1 Jan 2002

At the **at** prompt (**at>**), enter the first command to run at the specified time, and then press Enter.

Enter the next command to run, followed by Enter or press Ctrl-D, if finished.

An **at** job number is assigned to track the **at** job.

Use the **atq** command to view the **at** job queue.

Use the **atrm [at_job_#]** command to remove a scheduled job.

The crontab Utility

The crontab utility allows a user to schedule a command or program to run at scheduled intervals.

This utility is useful for scheduling backups, finding and removing core files in the user's home directory tree, or even emailing a friend a birthday note automatically on their birthday.

The **crontab** command is used to view and edit a users crontab file that stores scheduled program information.

On some systems, a user can edit their own crontab file that initially is empty. On some UNIX systems the user might not be permitted to have a crontab file for security reasons and will have to ask the system administrator to grant the user permission.

The crontab file

Each crontab entry in the crontab file consists of six fields separated by spaces or tabs. For example:

```
0 17 * * 5 /usr/bin/banner "Weekend Is Here!" > /dev/console
```

In the example above, the message "Weekend Is Here" would be displayed in the console window (Workspace Menu -> Hosts -> Console Terminal) at 5:00 p.m./1700 every Friday of the week of every month.

Field number and name	Event
1 - The minute field	Represents the minutes past the hour between 0 and 59.
2 - The hour field	Represents the hour of the day between 0 and 23.
3 - The day-of-the-month field	Represents the day of the month between 1 and 31.
4 - The month field	Represents the month between 1 and 12.
5 - The day-of-the-week field	Represents the day of the week between 0 (Sunday) and 6 (Saturday).
6 - The command field	Represents the name of the command or script to be run.

Editing the crontab file

The first 5 fields can have the following:

- A single value
- Multiple values that is 1,3,5 in the sixth field, which would mean Monday, Wednesday, Friday
- A range of values that is 1-5 in the sixth field which would mean Monday thru Friday
- A "*" to mean any or all values

To edit the crontab file use the **crontab -e** (edit) command.

To list the entries in the student's crontab file use the command **crontab -l** (list).

To delete the crontab file use the **crontab -r** (remove) command.

The cron daemon is started when the system boots and checks all user crontab files once every minute for commands to run at that time.