Processes and Job Control

Chonnam National University
School of Electronics and Computer Engineering

Kyungbaek Kim

Process

- The kernel considers each program running on your system to be a process
  - **Lives** : as it executes
  - **Dies** : when it terminates
- The kernel identifies each process by a number known as a process id (PID)
- The kernel keeps track of various properties of each process

dnslab.jnu.ac.kr - kbkim@dnslab - SSH Secure Shell

File Edit View Window Help

Quick Connect Profiles

```
top - 09:51:23 up 889 days, 18:20,  2 users,  load average: 0.01, 0.09, 0.10
Tasks: 245 total,   1 running, 244 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.1%us,  0.1%sy,  0.0%ni, 99.8%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4108652k total,  3856272k used,   252380k free,   457924k buffers
Swap:  9789432k total,      840k used,  9788592k free,  2964428k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3175 kbkim     20   0  125m  42m 2948 S    0  1.1  12:51.71 gnome-settings-
14408 root      20   0 50308  22m 5592 S    0  0.6  12:18.51 Xorg
 3331 kbkim     30  10 82160  19m 4848 S    0  0.5   5:21.25 update-manager
27586 root      20   0 23940  10m 4032 S    0  0.3   6:02.67 apache2
 3324 kbkim     20   0 39200 9800 3348 S    0  0.2   7:50.69 update-notifier
 3318 kbkim     20   0 31484 9256 2744 S    0  0.2   0:00.42 python
 8200 www-data  20   0 24468 8644 1172 S    0  0.2   0:00.08 apache2
 8201 www-data  20   0 24468 8632 1172 S    0  0.2   0:00.07 apache2
 8346 www-data  20   0 24468 8620 1172 S    0  0.2   0:00.05 apache2
 8344 www-data  20   0 24468 8608 1172 S    0  0.2   0:00.06 apache2
 8199 www-data  20   0 24108 8456 1176 S    0  0.2   0:00.08 apache2
 8202 www-data  20   0 24108 8456 1176 S    0  0.2   0:00.06 apache2
 8345 www-data  20   0 24108 8456 1176 S    0  0.2   0:00.06 apache2
 8244 www-data  20   0 24108 8452 1176 S    0  0.2   0:00.09 apache2
 8282 www-data  20   0 24108 8452 1176 S    0  0.2   0:00.05 apache2
17203 www-data  20   0 24108 8428 1176 S    0  0.2   0:00.01 apache2
 3259 kbkim     20   0 49052 7976 4204 S    0  0.2   4:52.82 indicator-apple
21893 root      20   0 20504 7564 6408 S    0  0.2   0:02.65 smbd
 3198 kbkim     20   0 68992 7336 3388 S    0  0.2   0:00.95 nautilus
 2420 root      20   0 20320 6980 5932 S    0  0.2   0:02.61 smbd
 3193 kbkim     20   0 39980 6784 3544 S    0  0.2   9:34.54 gnome-panel
 3238 kbkim     20   0 41588 6300 3332 S    0  0.2 259:48.92 wnck-applet
```

Connected to dnslab.jnu.ac.kr | SSH2 - aes128-cbc - hmac-md5 - | 84x29

Starting and ending Process

- Processes are created …
  - When the system boots
  - By the actions of another process (more later)
  - By the actions of a user
  - By the actions of a batch manager
- Processes terminate …
  - Normally – exit
  - Voluntarily on an error
  - Involuntarily on an error
  - Terminated (killed) by the actions a user or a process

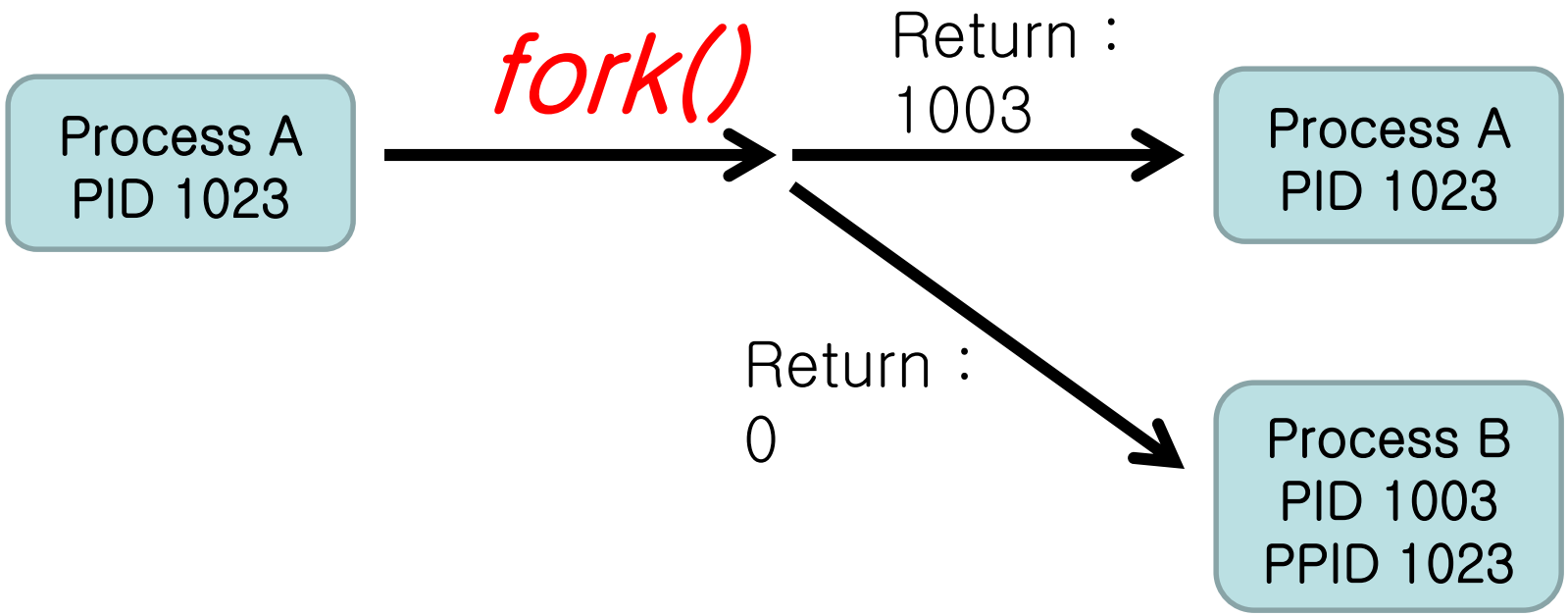
Creating new processes

- fork()
  - Creates a new process (child process) that is identical to the calling process (parent process)
    - Allocates new PCB
    - Clones the PCB of the calling process (almost exactly)
  - Returns 0 to the child process
  - Returns child's pid to the parent process

```
if (fork() == 0) {
   printf("hello from child\n");
}
else {
   printf("hello from parent\n");
}
```

Fork is interesting (and often confusing) because it is called *once* but returns *twice*

PID and fork


Process A
PID 1023
fork()
Return :
1003
Process A
PID 1023
Return :
0
Process B
PID 1003
PPID 1023

Running new programs

- int execl(char *path, char *arg0, char *arg1, …, 0)
  - Loads and runs executable at path with args arg0, arg1, …
    - path is the complete path of the process
    - arg0 becomes the name of the process
      - Typically arg0 is either identical to path, or else it contains only the executable filename from path
    - "real" arguments to the executable start with arg1, …
    - List of args is terminated by a (char *)0 argument
  - Returns –1 if error, otherwise doesn't return

```
main() {
   if (fork() == 0) {
      execl("/usr/bin/cp","cp","foo","bar",0);
   }
   wait(NULL);
   printf("copy completed\n");
   exit();
}
```

Other possible functions → execlp, execle, execv, execvp, execve

Processes in Linux

- Processes are represented by entries in a Process Table
- PID is an identifier to a Process Table Entry
- A Process Table Entry is called as a Process Control Block (PCB)
- PCB is a large data structure that contains or points to all info about the process
  - In Linux, PCB is defined in *task_struct*
    - See *include/linux/sched.h*
  - Windows XP → EPROCESS structure

Example of task_struct

kbkim@ubuntu: /usr/src/linux-headers-2.6.32-38/include/linux

File Edit View Terminal Help

```
struct task_struct {
        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;

        int lock_depth;         /* BKL lock depth */

#ifdef CONFIG_SMP
#ifdef __ARCH_WANT_UNLOCKED_CTXSW
        int oncpu;
#endif
#endif

        int prio, static_prio, normal_prio;
        unsigned int rt_priority;
        const struct sched_class *sched_class;
        struct sched_entity se;
        struct sched_rt_entity rt;

#ifdef CONFIG_PREEMPT_NOTIFIERS
"sched.h" [readonly] 2624 lines --47%--                    1241,1        46%
```

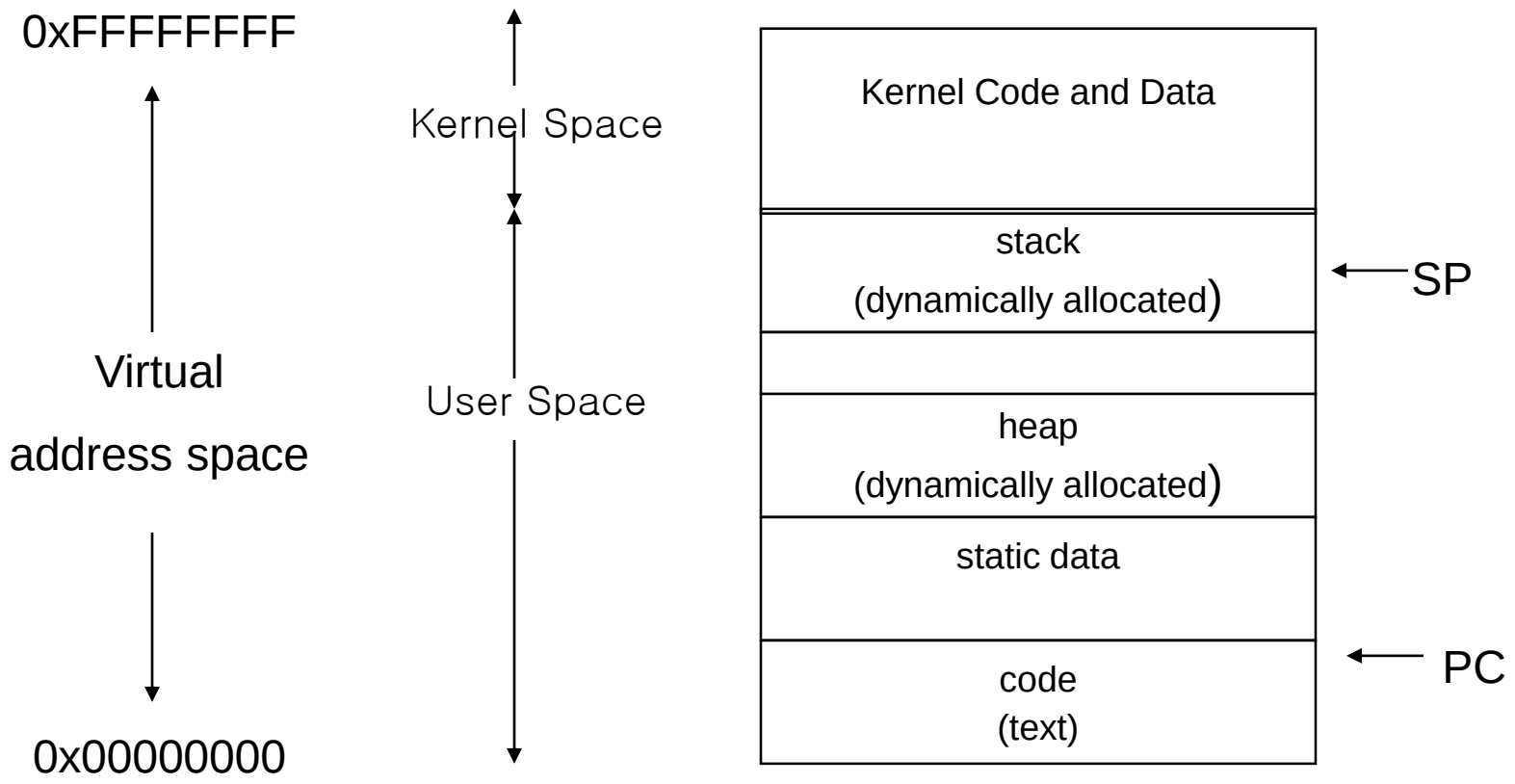
Information of Process

- Information of works
  - Process state, pointer to memory area, current directory, pointers to file descriptors, signals received, scheduling priority of works
- Information of users
  - User id (UID), group id (GID)
  - Effective user id (eUID), effective group id (eGID)
- Information of relationship between processes
  - Process ID (PID), Parent Process ID (PPID), Process Group ID (PGID)

Process components

- Address space – mapped into memory
- Code – program
- Data – initial data of program
- Execution Stack and Stack Pointer
  - Point the current running location of data
- Counter
  - Point the current running location of code
- A set of processor Registers
- A set of system resources
  - Files, pipes, privileges

32-bit based Address Space



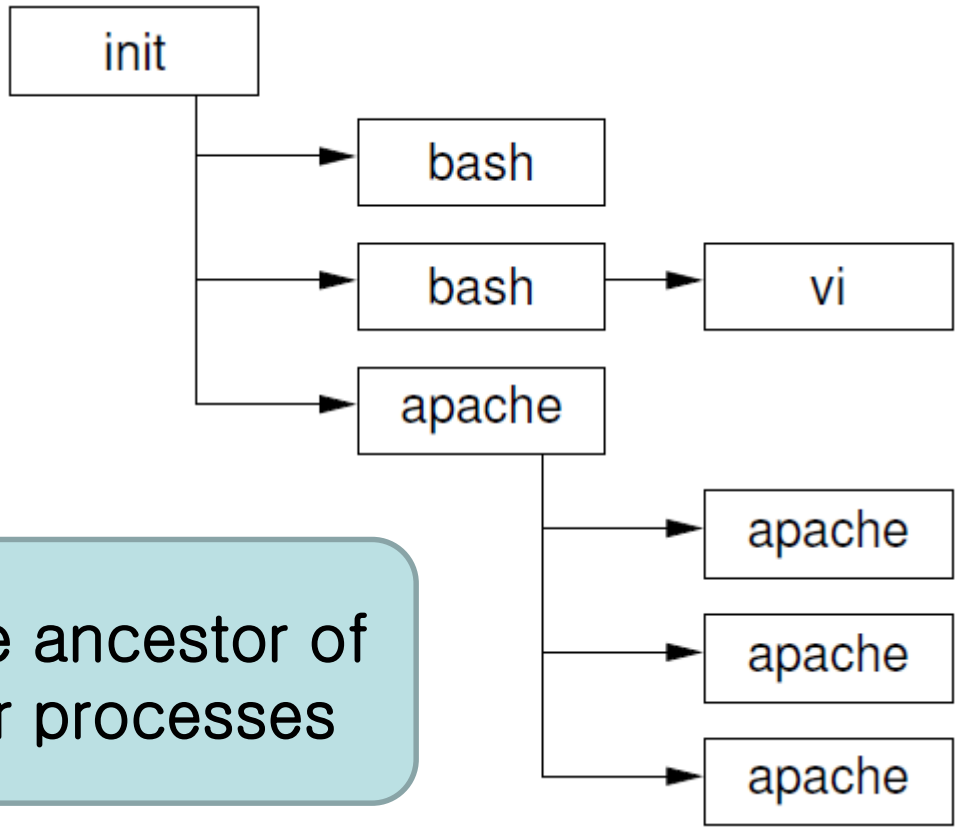
Process – PID and PPID

- Process ID – PID
  - A process ID is a unique identifier assigned to a process while it runs
  - Each time you run a process, it has a different PID
    - General PID range : 0 ~ 65535
    - It takes a long time for a PID to be reused by the system
- Parent Process ID – PPID
  - A process that creates a new process is called a parent process, the new process is called a child process
  - The parent process becomes associated with the new child process when it is created

Process – PGID

- Process Group ID – PGID
  - Each process belongs to a process group
  - A process group is a collection of one or more processes
    - If a command starts just one process, its PID and PGID are the same
  - Each process group has a unique process group ID
  - It is possible to send a signal to every process in the group just by sending the signal to the process group leader

Process Tree



init is the ancestor of all other processes

pstree

- Display a snapshot of running processes
- Always uses a tree-like display
- By default shows only the name of each command
- Normally shows all processes
  - Specify a pid as an argument to show a specific process and its descendants
  - Specify a user name as an argument to show process trees owned by that user

Example of “pstree –p”

kbkim@ubuntu: ~

File Edit View Terminal Help

```
init(1)-+-NetworkManager(746)-+-dhclient(2821)
        |                     `-{NetworkManager}(761)
        |-acpid(833)
        |-atd(838)
        |-bluetoothd(848)
        |-bonobo-activati(1848)---{bonobo-activat}(1849)
        |-clock-applet(1872)
        |-console-kit-dae(1451)-+-{console-kit-da}(1452)
        |                       |-{console-kit-da}(1453)
        |                       |-{console-kit-da}(1454)
        |                       |-{console-kit-da}(1455)
        |                       |-{console-kit-da}(1456)
        |                       |-{console-kit-da}(1457)
        |                       |-{console-kit-da}(1458)
        |                       |-{console-kit-da}(1459)
        |                       |-{console-kit-da}(1460)
        |                       |-{console-kit-da}(1461)
        |                       |-{console-kit-da}(1462)
        |                       |-{console-kit-da}(1463)
        |                       |-{console-kit-da}(1464)
        |                       |-{console-kit-da}(1465)
        |                       |-{console-kit-da}(1466)
        |                       |-{console-kit-da}(1467)
--More--
```

“ps” command

- Gives a snapshot of the processes running on a system at a given moment in time
- Very flexible in what it shows and how
  - Normally shows a fairly brief summary of each process
  - Normally shows only processes which are both owned by the current user and attached to a terminal
- Unfortunately, it doesn’t use standard option syntax

Options of ps

Option	Description
a	Show processes owned by other users
f	Display process ancestors in a tree-like format
u	Use the “user” output format, showing user names and process start times
w	Use a wider output format.
x	Include processes which have no controlling terminal
-e	Show information of all processes
-l	Use a long output format
-f	Use a full output format
-C cmd	Show only processes named cmd
-U user	Show only processes owned by user

“ps -help” for more details.

“ps -o pid,ppid,pgid,cmd” : -o option for modifying output format

Example of ps

TTY : associated terminal
TIME : cumulated CPU time
CMD : executable/command name
%CPU : portion of CPU usage
%MEM : portion of memory usage
VSZ : Virtual memory usage
RSS : Physical memory usage
STAT : process state
START : starting time of the process

kbkim@ubuntu: ~

File Edit View Terminal Help

```
kbkim@ubuntu:~$ ps
  PID TTY          TIME CMD
 3547 pts/0    00:00:00 bash
 4194 pts/0    00:00:00 ps
kbkim@ubuntu:~$ ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
kbkim     3547  0.0  0.3   5764  3132 pts/0    Ss   03:43   0:00 bash
kbkim     4195  0.0  0.1   2716  1056 pts/0    R+   05:26   0:00 ps u
kbkim@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 3547 pts/0    Ss     0:00 bash
 4196 pts/0    R+     0:00  \_ ps f
kbkim@ubuntu:~$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
kbkim     3547  3545  0 03:43 pts/0    00:00:00 bash
kbkim     4198  3547  0 05:26 pts/0    00:00:00 ps -f
kbkim@ubuntu:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  3547  3545  0  80   0 -  1441 wait   pts/0    00:00:00 bash
0 R  1000  4200  3547  0  80   0 -   626 -      pts/0    00:00:00 ps
kbkim@ubuntu:~$
```

top

- Show full-screen, continuously-updated snapshots of process activity
  - Waits a short period of time between each snapshot to give the illusion of real-time monitoring
- Processes are displayed in descending order of how much processor time they are using
- Also displays system uptime, load average, CPU status, and memory information
- Options
  - -d delay : wait delay
  - -i : ignore idle processes

Interactive commands of top

Key	Behavior
q	Quit the program
Ctrl+l	Repaint the screen
h	Show a help screen
k	Prompt for a pid and a signal and send that signal to that process
n	Prompts for the number of processes to show information
r	Change the priority of a process
s	Change the number of second delay
M	Sort the processes based on memory usage
P	Sort the processes based on CPU usage

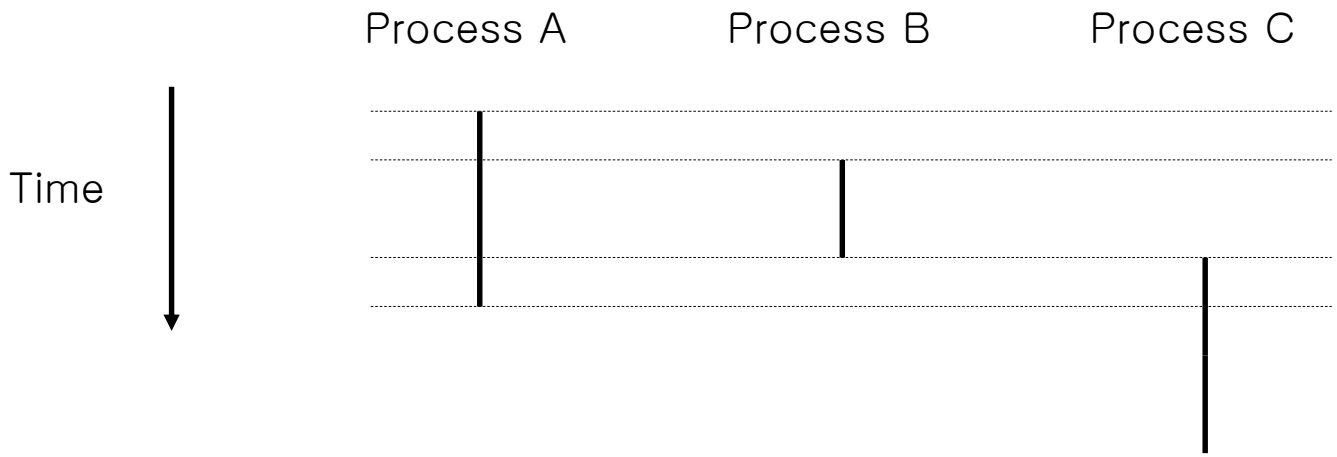
Example of top

```
kbkim@ubuntu: ~
File  Edit  View  Terminal  Help
top - 05:38:07 up  4:11,  2 users,  load average: 0.23, 0.19, 0.21
Tasks: 141 total,   2 running, 139 sleeping,   0 stopped,   0 zombie
Cpu(s):  5.0%us,  4.6%sy,  0.0%ni, 90.1%id,  0.0%wa,  0.3%hi,  0.0%si,  0.0%st
Mem:   1025960k total,   718804k used,   307156k free,    48912k buffers
Swap:   916472k total,        0k used,   916472k free,   519720k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1553 root      20   0 60332  17m 8064 S  4.3  1.8   5:05.25 Xorg
 1801 kbkim     20   0 57256  20m  15m S  1.0  2.0   1:28.61 vmtoolsd
 1348 root      20   0 35224 3656 2908 S  0.7  0.4   0:34.64 vmtoolsd
 4264 kbkim     20   0  2548 1212  920 R  0.7  0.1   0:00.32 top
 1439 root      20   0 12476 1488 1268 S  0.3  0.1   0:00.92 tpvmlp
 1691 root      20   0  3616 1232 1052 S  0.3  0.1   0:07.63 hald-addon-stor
 1845 root      20   0  5188  864  592 S  0.3  0.1   0:04.32 udisks-daemon
 3545 kbkim     20   0 44852  12m 9860 S  0.3  1.2   0:06.08 gnome-terminal
    1 root      20   0  2808 1644 1196 S  0.0  0.2   0:02.06 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.01 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:00.11 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      20   0     0    0    0 R  0.0  0.0   0:00.89 events/0
    7 root      20   0     0    0    0 S  0.0  0.0   0:00.00 cpuset
    8 root      20   0     0    0    0 S  0.0  0.0   0:00.00 khelper
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.00 netns
```

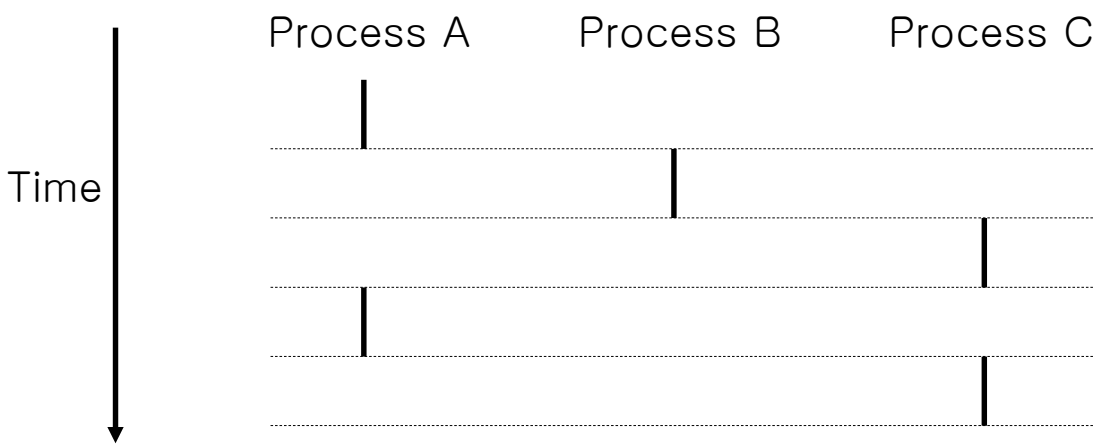
User view of Concurrent processes

- Users think of concurrent processes are running in parallel with each other.
- However, control flows for concurrent processes are physically disjoint in time



Logical Control Flow of Processes

- Each process has its own logical control flow
- Two processes **run concurrently (are concurrent)** if their flows overlap in time
- Otherwise, they are **sequential**

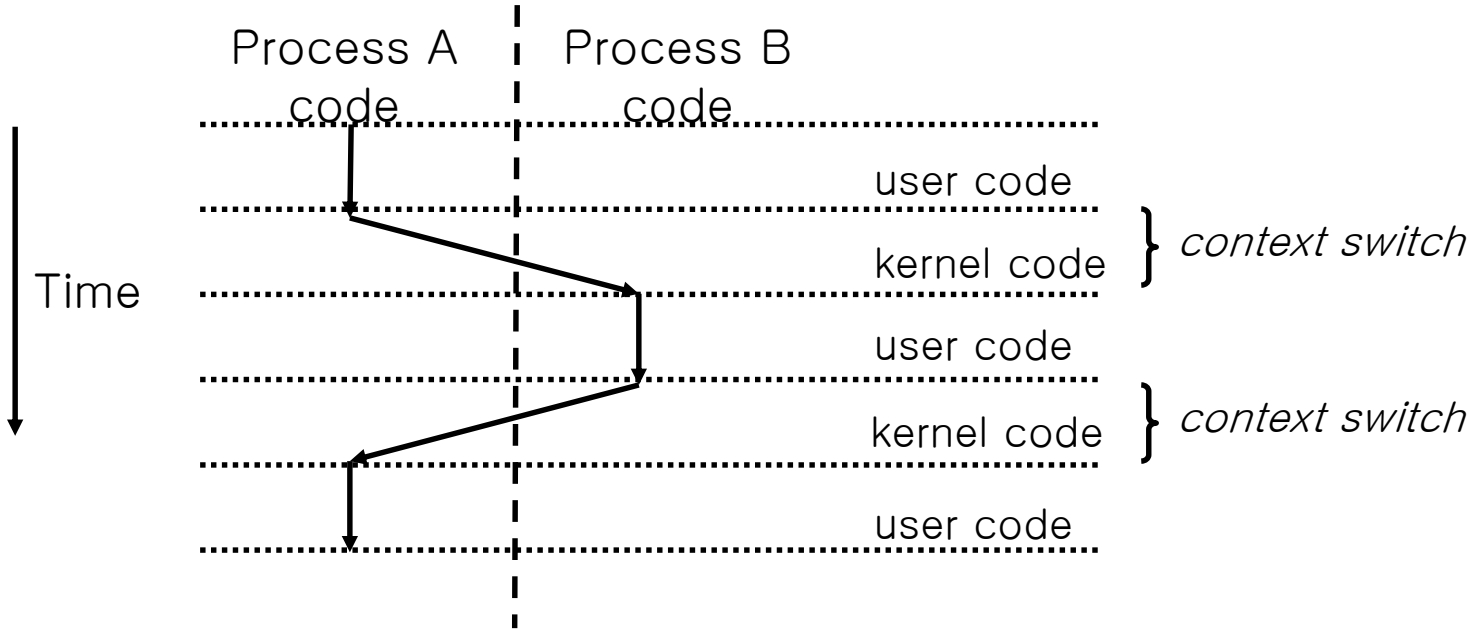


Concurrent : A & B, A & C

Sequential : B & C

Context switching

- Control flow passes from one process to another via a context switch
- During context switching, kernel code stores the PCB of the previous process and restores the PCB of the next process



Signaling Process

- A process can be sent a signal by the kernel or by another process
- Each signal is a very simple message:
  - A small whole number
  - With a mnemonic name
- Signal names are all-capitals, like **INT**
  - They are often written with SIG as part of the name : **SIGINT**
- Some signals are treated specially by the kernel
  - Others have a conventional meaning
- There are about 30 signals available

Available Signals

kbkim@ubuntu: ~

File Edit View Terminal Help

```
kbkim@ubuntu:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
kbkim@ubuntu:~$
```

Common Signals

name	number	meaning
INT	2	Interrupt – Stop running. Sent by the kernel when Ctrl+C is pressed in a terminal
TERM	15	“please terminate” used to ask a process to exit gracefully
KILL	9	“Die!” Forces the process to stop running; it is given no opportunity to clean up after itself
TSTP	20	Requests the process to stop itself temporarily. Sent by the kernel when Ctrl+Z is pressed in a terminal
HUP	1	Hang up. Sent by the kernel when you log out. Conventionally used by many daemons as an instruction to re-read configuration file
CHLD	17	Notify signal to its parent process

Sending signals : “kill” command

- The “kill” command is used to send a signal to a process
  - Not just to terminate a running process.
- It is a normal executable command, but many shells also provide it as a built-in
- Use “kill -HUP pid” or “kill -s HUP pid” to send a SIGHUP to the process with that pid
- If you miss out the signal name, kill will send a SIGTERM
- Special kill : **kill -9 -1** → kill all processes you can kill
- Process Group kill : **kill -- -<pgid>**
  → kill all processes under the given pgid

Example of kill

kbkim@ubuntu: ~

File Edit View Terminal Help

```
kbkim@ubuntu:~$ ps
  PID TTY          TIME CMD
 3547 pts/0    00:00:00 bash
 4431 pts/0    00:00:00 xterm
 4447 pts/0    00:00:00 xterm
 4490 pts/0    00:00:00 ps
kbkim@ubuntu:~$ ps
  PID TTY          TIME CMD
 3547 pts/0    00:00:00 bash
 4431 pts/0    00:00:00 xterm
 4447 pts/0    00:00:00 xterm
 4491 pts/0    00:00:00 ps
kbkim@ubuntu:~$ kill -9 4431
kbkim@ubuntu:~$
[1]-  Killed                  xterm
kbkim@ubuntu:~$ ps
  PID TTY          TIME CMD
 3547 pts/0    00:00:00 bash
 4447 pts/0    00:00:00 xterm
 4493 pts/0    00:00:00 ps
kbkim@ubuntu:~$ kill -9 -1
```

Sending Signals to Daemons

- Daemons → long-lived processes that provide some services
- Daemons typically have a configuration file which affects their behavior
- Many daemons read their configuration file only at startup
- If the configuration changes, you have to explicitly tell the daemon by sending it a SIGHUP signal
- e.g.) kill -HUP $(pidof /usr/bin/ssh-agent)
  - "pidof" → find the pid of the program

Modify process execution priorities

- Not all tasks requires the same amount of execution time
- Linux has the concept of execution priority to deal with this
- Process priority is dynamically altered by the kernel
- You can view the current priority by looking at **top** or **ps –l** and looking at the **PRI** column
- The priority can be biased using nice
  - The current bias can be seen in the NI column in top

```
kbkim@ubuntu:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  4798  4796  0  80   0 -  1443 wait   pts/0    00:00:00 bash
0 S  1000  4913  4798  0  80   0 -  2546 poll_s pts/0    00:00:00 xterm
```

“nice” command

- Starts a program with a given priority bias
- Peculiar name : ‘nicer’ processes require fewer resources
  - Niceness ranges from +19 (very nice) to -20 (not very nice)
  - Non-root user can only specify values from 1 to 19
  - the root user specify the full range of values
  - Default niceness when using nice is 10
- To run a command at increased niceness (lower priority)
  - nice -10 long-running-command &
  - nice -n 10 long-running-command &
- To run a command at decreased niceness (higher priority)
  - nice --15 important-command &
  - nice -n -15 important-command &

Example of nice

kbkim@ubuntu: ~

File Edit View Terminal Help

```
kbkim@ubuntu:~$ nice --15 firefox &
[1] 5276
kbkim@ubuntu:~$ nice: cannot set niceness: Permission denied

[1]+  Exit 1                  nice --15 firefox
kbkim@ubuntu:~$ nice -1 firefox &
[1] 5278
kbkim@ubuntu:~$ nice -10 xterm &
[2] 5300
kbkim@ubuntu:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  4798  4796  0  80   0 -  1443 wait   pts/0    00:00:00 bash
0 S  1000  5278  4798  8  81   1 - 66093 poll_s pts/0    00:00:01 firefox
0 S  1000  5300  4798  1  90  10 -  2546 poll_s pts/0    00:00:00 xterm
0 R  1000  5318  4798  0  80   0 -   626 -      pts/0    00:00:00 ps
kbkim@ubuntu:~$
```

Job control

- Most shells offer job control
  - The ability to stop, restart and background a running process
- The shell lets you put & on the end of a command line to start in the **background**
- You can hit Ctrl+z to **suspend** a running foreground job
- Suspended and backgrounded jobs are **given numbers** by the shell
- These numbers can be given to shell job-control built-in commands
- Job-control commands : **jobs, fg**, and **bg**

jobs

- Prints a listing of active jobs and their job numbers
- Job numbers are given in square brackets
  - But when you use them with other job-control builtins, you need to write them with percentage signs
- The jobs marked + and - may be accessed as %+ or %- as well as by numbers
  - %+ is the shell's idea of the current job
    - The most recently active job
  - %- is the previous current job

Example of jobs

kbkim@ubuntu: ~/test

File Edit View Terminal Help

```
kbkim@ubuntu:~/test$ vi test_while

[1]+  Stopped                 vim test_while
kbkim@ubuntu:~/test$ jobs
[1]+  Stopped                 vim test_while
kbkim@ubuntu:~/test$ xterm&
[2] 3433
kbkim@ubuntu:~/test$ jobs
[1]+  Stopped                 vim test_while
[2]-  Running                 xterm &
kbkim@ubuntu:~/test$ firefox&
[3] 3452
kbkim@ubuntu:~/test$ jobs
[1]+  Stopped                 vim test_while
[2]   Running                 xterm &
[3]-  Running                 firefox &
kbkim@ubuntu:~/test$ vi test_for

[4]+  Stopped                 vim test_for
kbkim@ubuntu:~/test$ jobs
[1]-  Stopped                 vim test_while
[2]   Running                 xterm &
[3]   Running                 firefox &
[4]+  Stopped                 vim test_for
kbkim@ubuntu:~/test$
```

fg

- Bring a backgrounded job into the foreground
- Re-starts a suspended job, running it in the foreground
- fg %1 will foreground job number 1
- fg with no arguments will operate on the current job

Example of fg

kbkim@ubuntu: ~/test

File Edit View Terminal Help

```
Retype password:
^Z
[5]+  Stopped                 ./test_while
kbkim@ubuntu:~/test$ jobs
[1]   Stopped                 vim test_while
[2]   Running                 xterm &
[3]   Running                 firefox &
[4]-  Stopped                 vim test_for
[5]+  Stopped                 ./test_while
kbkim@ubuntu:~/test$ fg %1
vim test_while

[1]+  Stopped                 vim test_while
kbkim@ubuntu:~/test$ jobs
[1]+  Stopped                 vim test_while
[2]   Running                 xterm &
[3]   Running                 firefox &
[4]   Stopped                 vim test_for
[5]-  Stopped                 ./test_while
kbkim@ubuntu:~/test$ fg %5
./test_while

Mismatched!! Try again
Retype password:
```

bg

- Re-starts a suspended job, running it in the background
- bg %1 will background job number1
- bg with no arguments will operate on the current job

Example of bg

kbkim@ubuntu: ~/test

File Edit View Terminal Help

```
kbkim@ubuntu:~/test$ jobs
[1]-  Stopped                 vim test_while
[2]   Running                 xterm &
[3]   Running                 firefox &
[4]   Stopped                 vim test_for
[5]+  Stopped                 ./test_while
kbkim@ubuntu:~/test$ bg
[5]+ ./test_while &
kbkim@ubuntu:~/test$ jobs
[1]-  Stopped                 vim test_while
[2]   Running                 xterm &
[3]   Running                 firefox &
[4]   Stopped                 vim test_for
[5]+  Stopped                 ./test_while
kbkim@ubuntu:~/test$ bg %1
[1]- vim test_while &
kbkim@ubuntu:~/test$ jobs
[1]+  Stopped                 vim test_while
[2]   Running                 xterm &
[3]   Running                 firefox &
[4]   Stopped                 vim test_for
[5]-  Stopped                 ./test_while
kbkim@ubuntu:~/test$
```

Running commands in the future

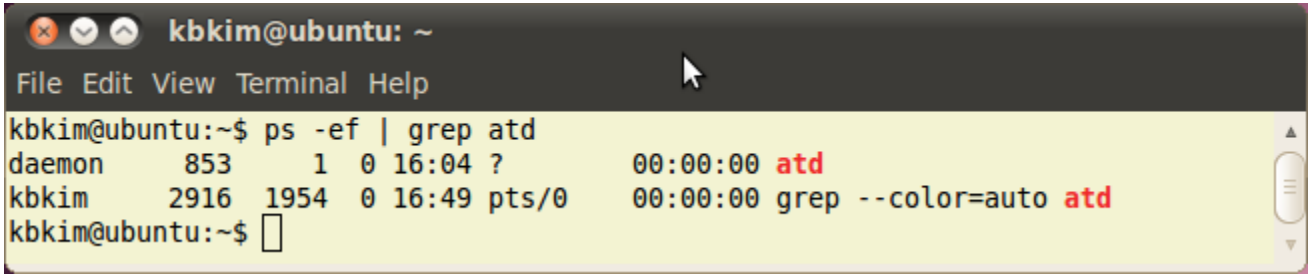
- There is sometimes a need for commands not to be run immediately, but scheduled to run later
- One-off commands:
  - “at 10:00 tomorrow, email me this reminder message”
  - These are known as “**at commands**”
- Regularly repeating commands
  - “every night, rebuild the database used by the locate command”
  - These are known as “**cron jobs**”

“at” command

- “at” command
  - To define a “at command”
- The time the command should run is given as a parameter to “at”
- “at” then prompts for the command itself
  - Command(s) exactly as they would be typed in the shell
  - Press “Ctrl+D” to finish
- The “at” daemon will run the command at the specified time

Commands run by “at” daemon

- A command executed by the at daemon
  - Has the permissions of its owner
  - Runs in the directory it was set up
  - Has the environment in which it was set up
  - Does not run in a terminal
- Output from the command
  - Cannot be included in a terminal window



Specification of “at” command

- A command may be specified on standard input instead of interactive console input
  - From a file
    - e.g.) $ at 16:30 < clean_process.sh
- Opening Windows
  - The $DISPLAY environment variable must be specified
    - e.g.)echo “DISPLAY=:0.0 xclock &” | at 11:00

Date&Time specification

- Unadorned times are in the next 24 hours:
  - $ at 09:30
- Tomorrow can be specified explicitly:
  - $ at 17:00 tomorrow
- A specific date can be used
  - $ at 11:00 Nov 11
  - $ at 00:30 16.04.12
- Relative times can be specified in minutes, hours, days, or weeks
  - $ at now + 45 minutes
  - $ at 16:00 + 3 days

Managing at commands

- “atq” command
  - List any pending at commands
  - The number at the start of each line identifies that at command
- “at –c number”
  - Display the corresponding at command including the environment.
- “atrm number”
  - Remove the corresponding at command

Example

kbkim@ubuntu: ~

File Edit View Terminal Help

```
kbkim@ubuntu:~$ echo "DISPLAY=$DISPLAY xclock&" | at 21:00
warning: commands will be executed using /bin/sh
job 4 at Sun May 20 21:00:00 2012
kbkim@ubuntu:~$ atq
3       Sun May 20 20:00:00 2012 a kbkim
4       Sun May 20 21:00:00 2012 a kbkim
kbkim@ubuntu:~$ atrm 3
kbkim@ubuntu:~$ atq
4       Sun May 20 21:00:00 2012 a kbkim
kbkim@ubuntu:~$
kbkim@ubuntu:~$
```

kbkim@ubuntu: ~

File Edit View Terminal Help

```
LESSCLOSE=/usr/bin/lesspipe\ %s\ %s; export LESSCLOSE
XAUTHORITY=/var/run/gdm/auth-for-kbkim-w7iCB0/database; export XAUTHORITY
COLORTERM=gnome-terminal; export COLORTERM
OLDPWD=/etc; export OLDPWD
cd /home/kbkim || {
         echo 'Execution directory inaccessible' >&2
         exit 1
}
DISPLAY=:0.0 xclock&

kbkim@ubuntu:~$
```

Cron jobs

- Periodical jobs
- Specification of simple cron jobs
  - /etc/cron.daily → for jobs to be run daily
  - /etc/cron.hourly → for jobs to be run hourly
  - /etc/cron.monthly → for jobs to be run monthly
  - /etc/cron.weekly → for jobs to be run weekly
  - Each file in the directory is run
    - The files are typically shell scripts
    - With root permissions

Example

```
kbkim@ubuntu: ~
File  Edit  View  Terminal  Help
kbkim@ubuntu:~$ ls /etc/cron.*
/etc/cron.d:
anacron

/etc/cron.daily:
0anacron  apt       bsdmainutils  logrotate  mlocate            quota
apport    aptitude  dpkg          man-db     popularity-contest  standard

/etc/cron.hourly:

/etc/cron.monthly:
0anacron  standard

/etc/cron.weekly:
0anacron  apt-xapian-index  man-db
kbkim@ubuntu:~$
```

More complex specification of cron jobs

- Sometimes more control is needed
  - To run jobs at a non-standard time
  - To run jobs as a user other than root
- /etc/cron.d
  - Each file in this directory must contain lines in a specific format
    - When the command should run
    - For which user the command should be run
    - The command to be run
  - Such a file is known as a cron table or crontab

Crontab format

- Blank lines are ignored
- Comments are lines starting with a hash(#)
- Environment variables can be set
  - e.g.) PATH=/usr/local/bin
- Example of cron job specification

  30 9 * * * root /usr/local/bin/check_logins

  → at 09:30

  → on all days

  → For the root user

  → Run “/usr/local/bin/check_logins”

Crontab date & time specification

- Order of the data and time fields
  - Minute (0-59)
  - Hour (0-23)
  - Day of the month (1-31)
  - Month (1-12)
  - Day of the week (0-7, 0 and 7 are Sunday)
- Fields almost in ascending order
- The command is run when the fields match the current time
- A field containing an asterisk (*) always matches

Example

kbkim@ubuntu: ~

File Edit View Terminal Help

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --repor
t /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --repor
t /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --repor
t /etc/cron.monthly )
#
```

"/etc/crontab" [readonly] 17L, 724C 1,1 All

More complex crontab dates and times

- A list of alternative values for a field are specified by commas:
  - Run at :15 and :45 each hour
  - 15, 45 * * * * httpd /usr/local/bin/generate-stats-page
- A range is specified with a hyphen
  - Run every half hour 09:15 – 17:45 Mon-Fri
  - 15,45 9-17 * * 1-5 root /usr/loca/bin/check-faxes
- Numbers rather than names must be used for months and days in lists and ranges
- A step through a range is specified with a slash
  - Run every two hours 08:30-18:30 Mon-Fri
  - 30 8-18/2 * * 1-5 root /usr/local/bin/check-faxes

User crontabs

- Sometimes non-root users need to set up cron jobs
- Each user has a crontab file
  - This is not edited directly
  - The “crontab” command manipulates it
  - Use “crontab -e” to edit the crontab
  - Use “crontab -l” to display the crontab

Example of crontab for a normal user

```
kbkim@ubuntu: ~
File  Edit  View  Terminal  Help
kbkim@ubuntu:~$ crontab -e
no crontab for kbkim - using an empty one

Select an editor.  To change later, run 'select-editor'.
  1. /bin/ed
  2. /bin/nano        <---- easiest
  3. /usr/bin/vim.basic
  4. /usr/bin/vim.tiny

Choose 1-4 [2]: 2
```

```
kbkim@ubuntu: ~
File  Edit  View  Terminal  Help
  GNU nano 2.2.2        File: /tmp/crontab.gjhOtI/crontab          Modified

# m h  dom mon dow   command
01 01 * * 1 tar cvf work.tar ~/work

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
 Y Yes
 N No           ^C Cancel
```

Example of crontab for a normal user

root@ubuntu: /var/spool/cron

File Edit View Terminal Help

```
kbkim@ubuntu:/var/spool/cron$ ls -l
total 12
drwxrwx--T 2 daemon daemon  4096 2012-05-20 21:00 atjobs
drwxrwx--T 2 daemon daemon  4096 2012-05-20 21:00 atspool
drwx-wx--T 2 root   crontab 4096 2012-05-20 22:23 crontabs
kbkim@ubuntu:/var/spool/cron$ su
Password:
root@ubuntu:/var/spool/cron# ls crontabs/
kbkim
root@ubuntu:/var/spool/cron# cat crontabs/kbkim
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (/tmp/crontab.gjhOtI/crontab installed on Sun May 20 22:23:04 2012)
# (Cron version -- $Id: crontab.c,v 2.13 1994/01/17 03:20:37 vixie Exp $)
# m h  dom mon dow   command
01 01 * * 1 tar cvf work.tar ~/work
root@ubuntu:/var/spool/cron#
```

Permission of “at” command and “cron” job

- Non-root users can be prohibited from having crontabs
  - If /etc/cron.allow exists then only users listed in it may have a crontab
  - If it does not exist but /etc/cron.deny does, then users not listed in the latter may have a crontab
  - If neither exist, then all users may have crontabs
- Permissions for running at commands are similar
  - /etc/at.allow and /etc/at.deny

Example

root@ubuntu: /etc

File Edit View Terminal Help

```
root@ubuntu:/etc# cat at.deny
alias
backup
bin
daemon
ftp
games
gnats
guest
irc
lp
mail
man
nobody
operator
proxy
qmaild
qmaill
qmailp
qmailq
qmailr
qmails
sync
sys
www-data
root@ubuntu:/etc#
```