Running a Program in the Background

- If you are running X environment, you can open a new window as follow:

```
obelix[16]% xterm -fg white -bg black &
[1] 18134
obelix[17]% xterm -fg black -bg white &
[2] 18150
```

Why is the PID not consecutive?

Running a Program in the Background

- To see the current list of jobs, jobs command is used

```
obelix[18]% jobs
[1]  Running                 xterm -fg white -bg black
[2]  Running                 xterm -fg black -bg white
```

- Jobs -l command list process IDs, in addition to the normal information

```
obelix[19]% jobs -l
[1]  18134 Running           xterm -fg white -bg black
[2]  18150 Running           xterm -fg black -bg white
obelix[20]% xterm &
[3] 18299
obelix[21]% jobs -l
[1]  18134 Running           xterm -fg white -bg black
[2]  18150 Running           xterm -fg black -bg white
[3]  18299 Running           xterm
```

Running a Program in the Background

```
obelix[22]% xterm -bg blue -fg yellow&
[4] 18408
obelix[23]% xterm &
[5] 19375
obelix[24]% jobs -l
[1]   18134 Running                 xterm -fg white -bg black
[2]   18150 Running                 xterm -fg black -bg white
[3]   18299 Running                 xterm
[4]   18408 Running                 xterm -bg blue -fg yellow
[5]   19375 Running                 xterm
```


Running a Program in the Background

- If you want to *asynchronously* communicate with a process,
  - you do so by sending a *signal* to the process(es)
- In Unix operating systems, kill is a command used to send a *signal* to a process.

  kill [-signal_name] pid...

  kill [-signal_number] pid...
- The kill command is a wrapper around the kill() system call, which sends signals to process(es) on the system

Running a Program in the Background

- Standard *signals* include:

Signal	Value	Abbrev.	Action
SIGHUP	1	HUP	**Hang-up:** sent to processes when you log out or if your terminal is disconnected
SIGINT	2	INT	Interrupt: sent when you press **^C**
SIGKILL	9	KILL	**Kill:** Immediate termination (Can ***not*** be trapped or ignored by a process)
SIGTERM	15	TERM	**Terminate:** request to terminate; (Can be trapped or ignored by a process)
SIGCONT	18	CONT	**Continue:** resume suspended process; sent by **fg** or **bg**
SIGSTOP	19	STOP	**Stop (suspend):** sent when you press **^Z**

- If no signal is specified in the kill command, by default, SIGTERM (signal 15)--*requesting* that the process exit--is sent.
- kill is something of a misnomer (misnamed);
  - the *signal sent* may *have nothing to do with process killing*.

Running a Program in the Background

- All signals except for SIGKILL and SIGSTOP can be *intercepted* (or even *ignored*) by the process
  - SIGKILL and SIGSTOP are *only seen* by the host system's ***kernel***, providing reliable ways of controlling the execution of processes.
- Unix provides security mechanisms to prevent unauthorized users from killing other processes.
  - Essentially, for a process to send a signal to another,
    - the owner of the signaling process must be the same as the owner of the receiving process or
    - be the ***super-user***.
- It is important to note that the specific mapping between numbers and signals could vary between Unix implementations.

Running a Program in the Background

- To terminate or signal processes, kill command is used

Currently, we have the following jobs:

```
[1]   18134 Running                 xterm -fg white -bg black
[2]   18150 Running                 xterm -fg black -bg white
[3]   18299 Running                 xterm
[4]   18408 Running                 xterm -bg blue -fg yellow
[5]   19375 Running                 xterm
obelix[25]% kill %3
obelix[26]%
[3]    Exit 15                      xterm
obelix[27]% kill 19375
obelix[28]%
[5]    Exit 15                      xterm
obelix[29]% jobs -l
[1]   18134 Running                 xterm -fg white -bg black
[2]   18150 Running                 xterm -fg black -bg white
[4]   18408 Running                 xterm -bg blue -fg yellow
obelix[30]%
```

Ctrl-c interrupt key can not abort a background process, but kill can

Running a Program in the Background

Currently, we have the following jobs:

```
[1]   18134 Running                 xterm -fg white -bg black
[2]   18150 Running                 xterm -fg black -bg white
[4]   18408 Running                 xterm -bg blue -fg yellow
obelix[30]%
[1]    Done                         xterm -fg white -bg black
obelix[30]%
obelix[30]%
obelix[30]%
obelix[30]% jobs -l
[2]   18150 Running                 xterm -fg black -bg white
[4]   18408 Running                 xterm -bg blue -fg yellow
```

Assuming that you exit from "xterm -fg white -bg black"

Running a Program in the Background

- To suspend (pause) a process, stop command is used
  Currently, we have the following jobs:

```
[2]   18150 Running                 xterm -fg black -bg white
[4]   18408 Running                 xterm -bg blue -fg yellow

obelix[31]% stop %2
[2]  Suspended (signal)          xterm -fg black -bg white
obelix[32]% jobs -l
[2]  18150 Suspended (signal)          xterm -fg black -bg white
[4]  18408 Running                  xterm -bg blue -fg yellow
obelix[33]% stop 18408
[4]  Suspended (signal)          xterm -bg blue -fg yellow
obelix[34]% jobs -l
[2]  18150 Suspended (signal)          xterm -fg black -bg white
[4]  18408 Suspended (signal)          xterm -bg blue -fg yellow
obelix[35]%
```

Exactly the same as kill –STOP %2

Exactly the same as kill –STOP 18408

Running a Program in the Background

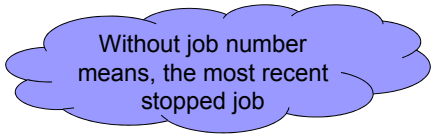
- To resume a suspended process in the *background*, bg command is used
  Currently, we have the following jobs:

```
[2]  18150 Suspended (signal)          xterm -fg black -bg white
[4]  18408 Suspended (signal)          xterm -bg blue -fg yellow
obelix[36]% bg %4
[4]  xterm -bg blue -fg yellow &
obelix[37]% jobs -l
[2]  18150 Suspended (signal)          xterm -fg black -bg white
[4]  18408 Running                      xterm -bg blue -fg yellow
obelix[38]% stop %4
[4]  Suspended (signal)          xterm -bg blue -fg yellow
obelix[39]% jobs -l
[2]  18150 Suspended (signal)          xterm -fg black -bg white
[4]  18408 Suspended (signal)          xterm -bg blue -fg yellow
```

Exactly the same as kill –CONT %4

Running a Program in the Background

```
obelix[40]% bg
[4]   xterm -bg blue -fg yellow &
obelix[41]% bg
[2]   xterm -fg black -bg white &
obelix[42]% jobs -l
[2]   18150 Running                 xterm -fg black -bg white
[4]   18408 Running                 xterm -bg blue -fg yellow
obelix[43]%
```

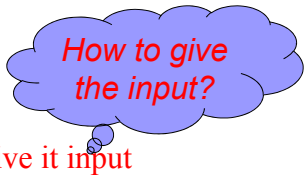


Running a Program in the Background

- fg (*foreground*) command is used to
  - transfer a process from the background to the foreground and
  - make it the current process
- When a process being in the foreground,
  - you can use Ctrl-c interrupt key to abort it

Running a Program in the Background

- If a *background* task *sends output* to the standard output, or standard error, and you do not redirect it,
  - output appears on the screen, even if you are running another job
- If a *background* task *requests input* from the standard input, and you have not redirected the standard input,
  - *Bourne-style shells (i.e., sh, bash)*
    - Supply a null string
  - *C-style shells (e.g., csh, tcsh)*
    - Suspend the job and wait for you to give it input



- You will probably want to redirect the output of a job you run in the *background* to keep it from interfering with whatever you are doing at the window

daemon

- Typically in a Unix system, hundreds of processes are running at the same time
- Some of these processes are called ***daemon***:
  - programs running in the *background*
  - completely *disconnected* from any terminal
- A ***daemon*** will wait silently in the background for something to happen (e.g., an event, a request, an interrupt, a specific time interval)
- When the trigger occurs, the ***daemon*** swings into action, doing whatever is necessary to carry out its job

daemon

- Most ***daemons*** are created during the boot sequence
- ***daemons*** are created by either
  - the ***init*** process (***process#1***), or
  - parents that terminate themselves
    - ***daemons*** become ***orphans***
      - ***Adopted*** by ***init*** (***process#1***)
- Hence, we can safely say that
  - ***process#1*** is the ***parent*** of all ***daemon*** processes

daemon

- Unix ***daemons*** include
  - ***init:*** The Unix program which spawns all other processes.
  - ***crond:*** Time-based job scheduler, runs jobs in the background.
  - ***dhcpd:*** Dynamically configure TCP/IP information for clients.
  - ***fingerd:*** Provides a network interface for the finger protocol, as used by the finger command.
  - ***ftpd:*** Services FTP requests from a remote system.
  - ***httpd:*** Web server daemon.
  - ***inetd:*** Listens for network connection requests.
  - ***lpd:*** The line printer daemon that manages printer spooling.
  - ***nfsd:*** Processes NFS operation requests from client systems.

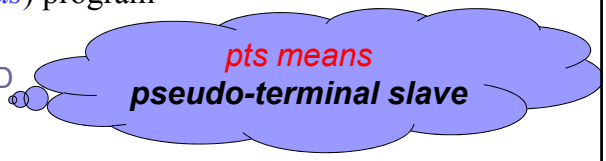
daemon

- Unix ***daemons*** include
  - ***ntpd:*** Network Time Protocol daemon that manages clock synchronization across the network.
  - ***sshd:*** Listens for secure shell requests from clients.
  - ***sendmail:*** SMTP daemon.
  - ***swapper:*** Copies process regions to swap space in order to reclaim physical pages of memory for the kernel. (Also called ***sched***.)
  - ***syslogd:*** System logger process that collects various system messages.
  - ***syncd:*** Periodically keeps the file systems synchronized with system memory.
  - ***xfsd:*** Serve X11 fonts to remote clients.

ps command

- To display information about processes, you use
  - the ***ps*** (*process status*) program

```
obelix[43]% ps
  PID TTY      TIME CMD
 7598 pts/20   0:00 tcsh
 7639 pts/20   0:00 ps
```



- ***ps*** comes with many options, yet these options depend on the Unix that you are using (AT&T Unix or BSD Unix)
  - AT&T options: ps [-aefly] [-p *pid*] [-u *userid*]
  - BSD options: ps [ajluvx] [p *pid*] [U *userid*]
- In this lecture, I will focus only on *some* of the *AT&T **ps** options*

ps command

```
obelix[44]% ps -f
    UID   PID  PPID  C    STIME TTY      TIME CMD
elsakka  7598  7588  0 11:42:12 pts/20   0:00 -tcsh
elsakka  7720  7598  0 11:57:08 pts/20   0:00 /bin/ps -f

obelix[45]% ps -fp 7588
    UID   PID  PPID  C    STIME TTY      TIME CMD
elsakka  7588  7587  0 11:42:07 ?        0:00 /usr/lib/ssh/sshd -4

obelix[46]% ps -fp 7587
    UID   PID  PPID  C    STIME TTY      TIME CMD
   root  7587   516  0 11:42:07 ?        0:00 /usr/lib/ssh/sshd -4
```


ps command

```
obelix[47]% ps -fp 516
    UID   PID  PPID  C    STIME TTY      TIME CMD
   root   516     1  0     Sep 10 ?      0:02 /usr/lib/ssh/sshd -4

obelix[48]% ps -fp 1
    UID   PID  PPID  C    STIME TTY      TIME CMD
   root     1     0  0     Sep 10 ?      0:03 /sbin/init

obelix[49]% ps -fp 0
    UID   PID  PPID  C    STIME TTY      TIME CMD
   root     0     0  0     Sep 10 ?      0:11 sched
```

ps command

```
obelix[50]% ps -ly
S   UID   PID  PPID  C PRI NI  RSS   SZ  WCHAN  TTY      TIME CMD
S 13057  7598  7588  0  50 20 2664  3584      ?  pts/20   0:00 tcsh
O 13057  7779  7598  0  50 20 1192  1840         pts/20   0:00 ps

obelix[51]% ps -lyp 1
S   UID  PID  PPID  C PRI NI  RSS   SZ  WCHAN  TTY      TIME CMD
S     0    1     0  0  40 20 1104  2984      ?  ?        0:03 init

obelix[52]% ps -lyp 0
S   UID  PID  PPID  C PRI NI  RSS   SZ  WCHAN  TTY      TIME CMD
T     0    0     0  0   0 SY    0     0         ?        0:11 sched
```

S means state-code

S means Sleeping: waiting for an event to complete

O means OnProc : currently executing

R means Ready: Process is the ready queue

T means Terminated

Command Separation and Grouping

- When you give a shell more than one command or when you write a shell script, you must ***separate*** commands from one another
- ***Command separation*** includes
  - ***Newline*** character
    - ***Initiates execution*** of the command proceeding it
  - ***;*** character
    - *Does not initiate execution* of the command proceeding it
  - ***&*** character
    - *Does not initiate execution* of the command proceeding it
    - Executes the task in the *background*
  - | character
    - *Does not initiate execution* of the command proceeding it
    - *Pipes* the output of the command to the left to the input of the command to the right
  - **( )** characters, i.e., parentheses
    - *Does not initiate execution* of the command proceeding it
    - *Groups* commands (the shell treats each group as one job)

Command Separation and Grouping

Example:

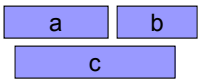
```
obelix[43]% who am I; date; pwd
elsakka    pts/2        Sep 19 02:16
Thu Sep 19 02:16:36 EDT 2013
/faculty/elsakka

obelix[44]% who am I
elsakka    pts/2        Sep 19 02:17

obelix[45]% date
Thu Sep 19 02:17:15 EDT 2013

obelix[46]% pwd
/faculty/elsakka

obelix[47]%
```


Command Separation and Grouping

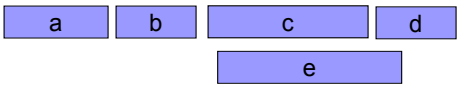
Examples:

```
ls | sort | more
ls | sort > out_file &
```

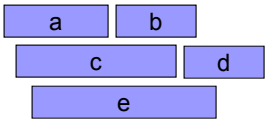
(a; b) & c



(a; b); (c; d) & e



(a; b) & (c; d) & e



(a; b)& (c; d) ; e