BasicOS

Processes

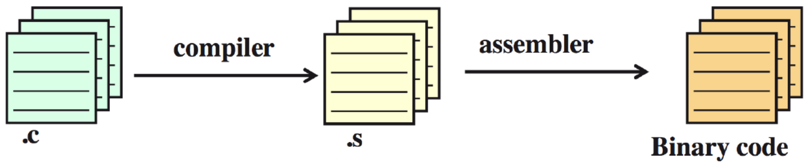
Ludovic Apvrille ludovic.apvrille@telecom-paris.fr
Eurecom, office 470

https://perso.telecom-paris.fr/apvrille/BasicOS/

Program

Abstraction

- Program is usually written in a high level language
- Compilers / interpreters convert high level languages into binary code



```
$ gcc -Wall -o writeToFile writeToFile.c
```

Process Definition

Definition of a process

Program in execution

Programs and processes

- One execution of a program = one process
- Two executions of the same program = two processes

Computer system = set of processes

- Operating System processes
- User processes

Running Processes

Executing a process

```
$ ls /home
Admin_Data  eurecom  Local_Data  lost+found
```

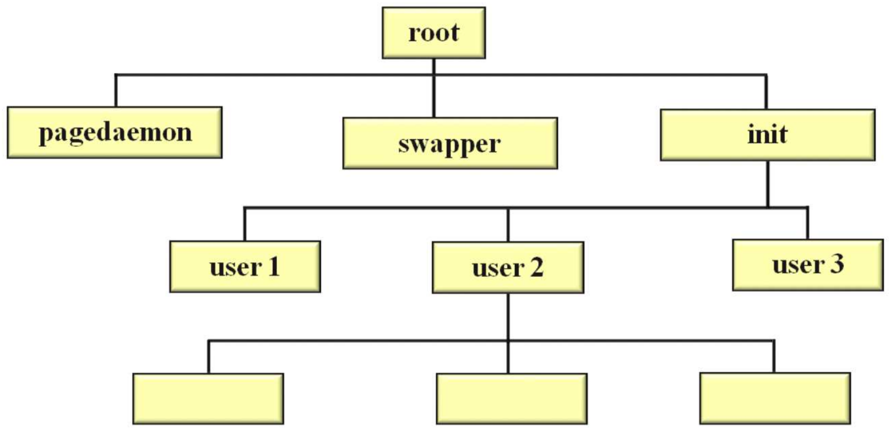
Executing 2 processes one after the other

```
$ ls /wrongdir ; ls /home
ls: cannot access '/wrongdir': No such file or directory
Admin_Data  eurecom  Local_Data  lost+found
```

Executing a second process only if the first one succeeds

```
$ ls /wrongdir&&ls /home
ls: cannot access '/wrongdir': No such file or directory
```

UNIX: Hierarchy of Processes



Init = process spawner, Swapper = scheduler, Pagedaemon = memory manager

Processes: Listing, User, pid, ppid, Killing

ps bash command

```
$ man ps
ps − report a snapshot of the current processes.
...
−e  Display information about other users processes,
including those without controlling terminals.
...
−f Display the uid, pid, parent pid, recent CPU usage,
process start time, controlling tty, elapsed CPU usage, and
the associated command
```

Processes: Listing, User, pid, ppid, Killing (Cont.)

kill bash command

```
$ man kill
kill - send a signal to a process
kill [options] <pid> [...]
...
Particularly useful signals include HUP, INT, KILL,
STOP, CONT, and 0. Alternate signals may be specified
in three ways: -9, -SIGKILL or -KILL
...
kill -9 -1
   Kill all processes you can kill.
```

Processes: Listing, User, pid, ppid, Killing (Cont.)

```
$ ssh apvrille@megantic
$ bash
$ ps -ef|grep apvrille
 ...
apvrille  525653  525519  0 08:59 ?      00:00:00 sshd: ...
apvrille  525671  525653  0 08:59 pts/0  00:00:00 -bash
apvrille  525684  525533  0 08:59 ?      00:00:00 /usr/libexec...
apvrille  525763  525671  0 09:00 pts/0  00:00:00 bash
apvrille  525869  525763  0 09:00 pts/0  00:00:00 ps -aef
apvrille  525870  525763  0 09:00 pts/0  00:00:00 grep apvrille

$ kill -9 525869
bash: kill: (525869) - No such process

$ kill -9 525763
Killed
```

Processes: Listing, User, pid, ppid, Killing (Cont.)

[Only one bash remains]

```
$ ps -ef|grep apvrille
apvrille  525653  525519  0 08:59 ?       00:00:00 sshd: ...
apvrille  525671  525653  0 08:59 pts/0   00:00:00 -bash
apvrille  525684  525533  0 08:59 ?       00:00:00 /usr/libexec...
apvrille  526070  525671  0 09:14 pts/0   00:00:00 ps -aef
apvrille  526071  525671  0 09:14 pts/0   00:00:00 grep apvrille
```

[Killing a root process without being root]

```
$ ps -ef
...
root 1017 1  0 Jul01 ?   00:14:49 /usr/bin/dockerd -H ...
...
$ kill -9 1017
-bash: kill: (1017) - Operation not permitted
```

Processes: Listing, User, pid, ppid, Killing (Cont.)

[Killing all processes (authorized to be killed): session is closed]

```
$ kill -9 -1
Connection to eurecom1 closed by remote host.
```

[CTRL-D: means an end of file. The current terminal exists because it waits for data from the input terminal until this input stream ends]

```
$ ssh apvrille@megantic

$ <CTRL-D> logout
Connection to eurecom1 closed.
```

Foreground and Background Processes

Foreground processes

Example:

```
$ cmd
```

- Default behavior
- Not possible to use the shell until the process completes or is suspended
- Process terminates when shell or terminal exists

Background processes

Example:

```
$ cmd &
```

- Shell can be used while process is running
- Process continues when its *shell* exits
- Process is killed when its terminal exits (there are a few exceptions to this)

Foreground and Background Processes: Example #1

[Starting a process from a terminal, then pausing with CTRL-Z]

```
$ sleep 100
^Z
[1]+  Stopped                 sleep 100
```

[Listing jobs and continuing]

```
$ jobs -l
[1]+ 527692 Stopped                 sleep 100

$ fg %1
sleep 100
...
```

Foreground and Background Processes: Example #2

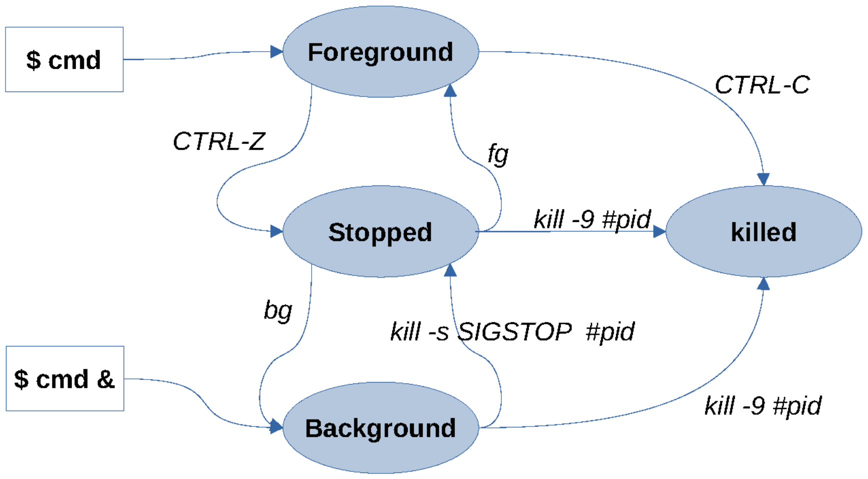
[Starting a process from a terminal, then pausing with CTRL-Z]

```
$ sleep 100
^Z
[1]+  Stopped                 sleep 100
```

[Continuing in backgroud]

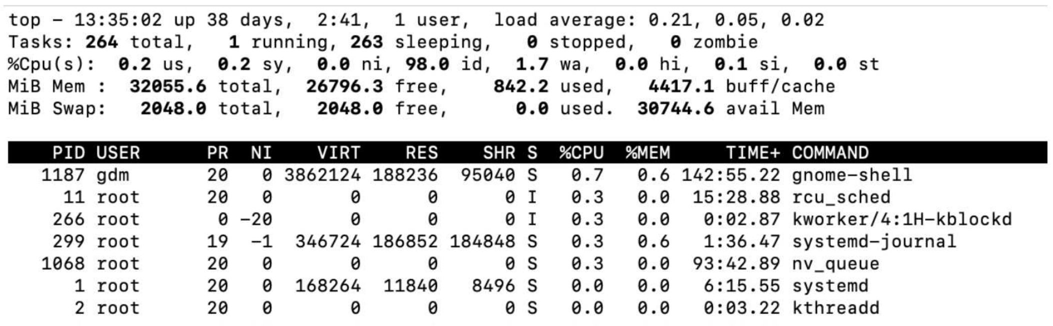
```
$ bg
[1]+ sleep 100 &
```

Foreground and Background Processes: A summary



Monitoring Processes

```
$ top
```



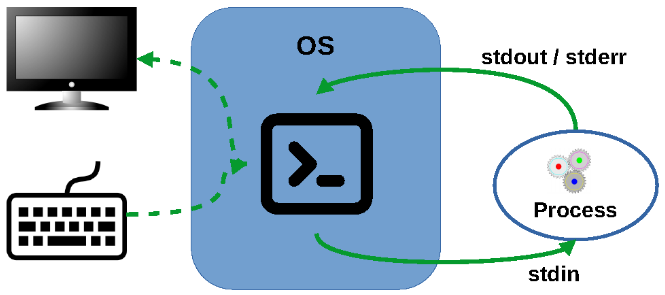
```
top - 13:35:02 up 38 days,  2:41,  1 user,  load average: 0.21, 0.05, 0.02
Tasks: 264 total,   1 running, 263 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.2 sy,  0.0 ni, 98.0 id,  1.7 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :  32055.6 total,  26796.3 free,    842.2 used,   4417.1 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.  30744.6 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1187 gdm       20   0 3862124 188236  95040 S   0.7   0.6 142:55.22 gnome-shell
     11 root      20   0       0      0      0 I   0.3   0.0  15:28.88 rcu_sched
    266 root       0 -20       0      0      0 I   0.3   0.0   0:02.87 kworker/4:1H-kblockd
    299 root      19  -1  346724 186852 184848 S   0.3   0.6   1:36.47 systemd-journal
   1068 root      20   0       0      0      0 S   0.3   0.0  93:42.89 nv_queue
      1 root      20   0  168264  11840   8496 S   0.0   0.0   6:15.55 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:03.22 kthreadd
```

Data Streams of (GNU/Linux) Processes

Three default streams per process

A stream is attached to the corresponding terminal

Name	File descriptor	Comment
stdin	0	input stream
stdout	1	output stream
sdterr	2	error stream



Data Streams of Processes: Redirection vs. Pipe

cmd > file (or cmd < file)

Output stream of *cmd* is sent to a file (or: input stream given as input to *cmd*)

```
/home$ ls > /tmp/foo

/home$ cat /tmp/foo
Admin_Data
eurecom
Local_Data
lost+found
```

cmd1 | cmd2

Ouput stream of *cmd*1 is forwarded to the input stream of *cmd*2
Two processes are created

```
/home$ ls|grep ata
Admin_Data
Local_Data
```

Data Streams of Processes: Advanced Redirections

Redirecting both *stdout* and *stderr* to two different files

```
$ cmd 1> output.txt  2> error.txt
```

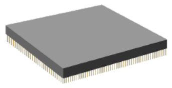
Redirecting *stderr* to *stdout*:

```
$ cmd  2>&1 > file
$ cmd > file 2>&1
```

Beware:

1. First command: stderr goes to terminal, stdout to *file*.
2. Second command: both streams go to *file*.

CPU Protection



Goal

The OS must be sure to periodically gain control

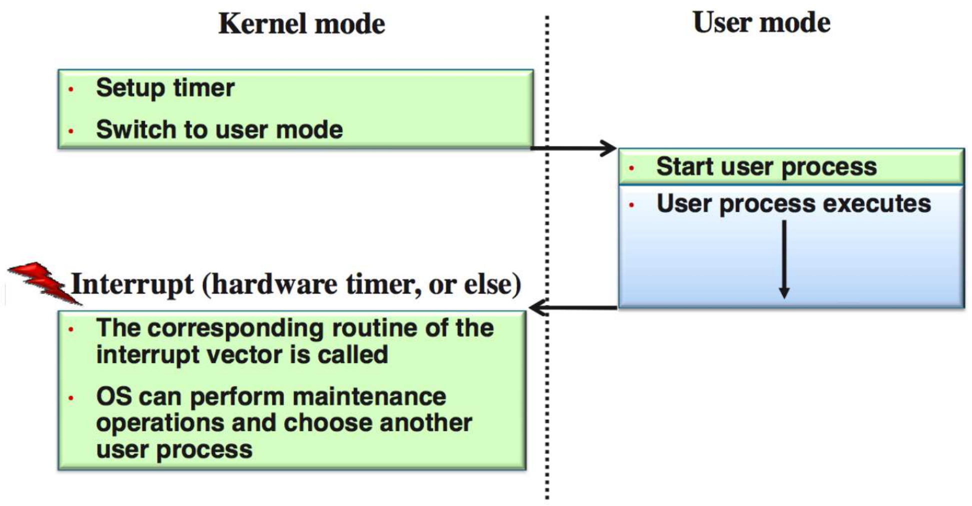
- Ensure CPU fairness between processes
- Prevent a process from stucking the system
  - e.g., infinite loop

Example of mechanisms

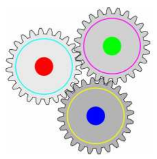
1. A hardware timer is set before a process is given the CPU
2. The timer interrupts the process after a specified period

Of course, instructions for settling the timer are privileged

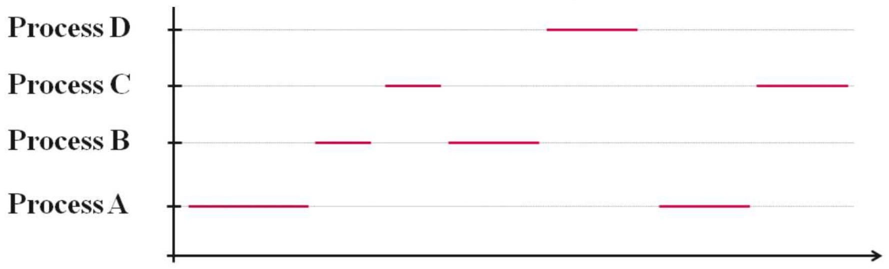
Example of CPU Protection


Kernel mode
User mode
Setup timer
Switch to user mode
Start user process
User process executes
Interrupt (hardware timer, or else)
The corresponding routine of the interrupt vector is called
OS can perform maintenance operations and choose another user process

Selection of Processes: Scheduler



- One processor with one execution core
  - Pseudo-parallelism: 1 process running at a time
    - So, either the OS or a user process is running



- Multiprocessor or one processor with several cores
  - A process can be running on each processor / core