```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAXPAROLA 30
#define MAXRIGA 80

int main(int argc, char *argv[])
{
  int freq[MAXPAROLA] ; /* vettore di contatori
  delle frequenze delle lunghezze delle parole */
  char riga[MAXRIGA] ;
  int i, inizio, lunghezza ;
  FILE * f ;

  for(i=0; i<MAXPAROLA; i++)
    freq[i]=0 ;

  if(argc != 2)
  {
    fprintf(stderr, "ERRORE, serve un parametro con il nome del file\n");
    exit(1);
  }
  f = fopen(argv[1], "rt") ;
  if(f==NULL)
  {
    fprintf(stderr, "ERRORE, impossibile aprire il file %s\n", argv[1]);
    exit(1);
  }

  while( fgets( riga, MAXRIGA, f ) != NULL )
```

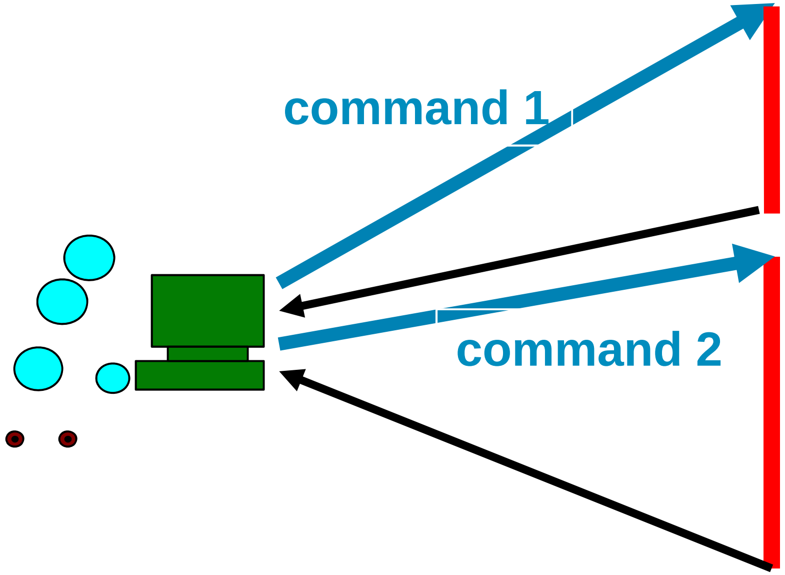
Processes

Shell commands for process management

Stefano Quer and Pietro Laface
Dipartimento di Automatica e Informatica
Politecnico di Torino

Foreground execution

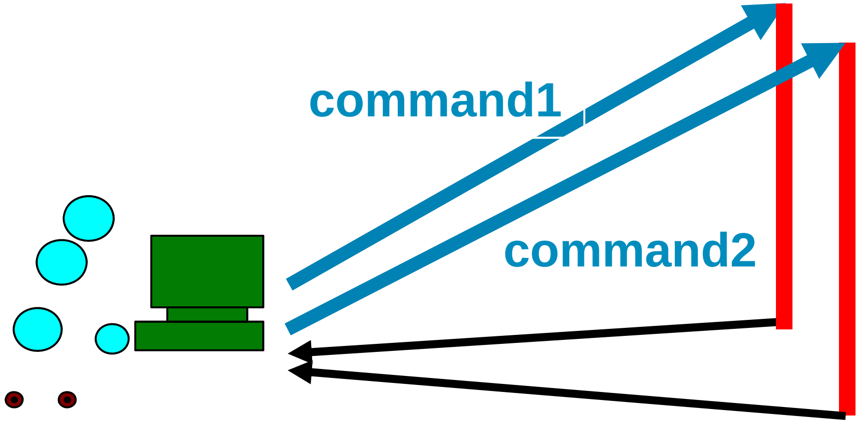
- The standard shell commands
  - Allow executing processes **sequentially**
  - Each process is executed in **foreground,** i.e., using the control terminal



```
> command1
Output of command1
> command2
Output of command2
>
```

Background execution

- The shell interpret character & as an indication to run the command in **background**
  - The process is executed in concurrency with the shell. It loses the control terminal input
  - The shell outputs immediately a new prompt
  - It is possible to run several processes in parallel



```
> command1 &
> command2 &
>

Output of command1
Output of command2
```

Process status commands

- Two main commands
  - **ps (process status)**

    ps <options>

    - List active processes and some related information li
    - Options

–e (or –A)	List all processes
–f	Extended format
r (not –r)	Shows only the "running" processes
–u <user>	Shows only the <user> processes

Process status commands

Command **top**

- Display and updates information about the system used resources, and the active processes

```
user@mahine:~/$ top

top - 10:26:58 up 57 min,  3 users,  load average: 0.00, 0.01, 0.05
Tasks: 152 total, 2 running, 150 sleeping, 0 stopped, 0 zombie
%Cpu(s): 4.0 us, 0.6 sy, 0.4 ni, 93.5 id,  1.4 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem:  8177092 total,  1382976 used,  6794116 free,   174096 buffers
KiB Swap: 10482684 total,        0 used, 10482684 free.   544664 cached Mem

 PID USER     PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
1821 user     20   0 1297200 198644  39328 S 65.6  2.4   1:59.62 compiz
1302 root     20   0  326708 101316  17712 S 13.1  1.2   0:23.63 Xorg
   1 root     20   0   33648   3028   1492 S  0.0  0.0   0:00.78 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   0:00.01 ksoftirqd/0
   4 root     20   0       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0
...
```

kill command

- **kill** allows sending signal from the shell
- Format
  - kill [–sig] pid
    - Sends signal sig to process with PID=pid
  - Option **sig** indicates the signal code
  - **pid** is the process identifier (PID) of the target process
  - The default signal of **kill** is **SIGTERM**, the standard termination command

kill command

- A signal **sig** can be indicated by means of its name or by its corresponding number
  - The list of the available signals can be obtained using the `–l` option
    - `SIGKILL = KILL = 9`
    - `SIGUSR1 = USR1 = 10`
    - `SIGUSR2 = USR2 = 12`
    - `SIGALRM = ALRM = 14`
    - etc.

kill command

- Examples

```
kill -l
kill -9 10234
kill -SIGKILL 10234
kill -KILL 10234
```

List available signals

Three commands to terminate process with PID 10234

- Shell command **`killall`** terminates all process with a specified name

```
killall -9 name
```

  - Useful to terminate all processes generated by the same program avoiding to specify their PIDs