1.3 Process architecture

1.3.1 The characteristics of the process architecture

The ranking of processes in a tree hierarchy base is based on the parent-child relationship can be printed with the **pstree** command.

In the creation of a child process it inherits from the father a whole series of properties and resources, and in particular one of the characteristics that distinguishes the architecture of the processes in a UNIX-like system is that all the files opened in the parent process will remain such and immediately also available for the child.

A second difference compared to other multi-user systems is that in Linux the creation of a process and the execution of a program are two separate operations, managed by two different system calls, the first of which creates a new process, identical to the parent, the which normally goes to use the second to run another program.

This feature also allows you to write a single program that executes multiple processes, with a structure in which the father executes the part that takes care of receiving requests, and for each of them he has a specially created child perform the operations necessary to provide their responses.

1.3.2 The properties of the processes

The command that allows you to get the list of active processes in the system is **ps** (**p**rocess **s**tatus), with the option:

- a: processes launched by other users will also be displayed, as long as they refer to a terminal;
- x: all processes not associated with a terminal are displayed;
- f: it allows to show the hierarchy of processes;
- r: allows you to restrict the selection to only actual running processes.

The output of the **ps** command is divided into columns:

- PID: in the first column, it shows the so-called process ID of the process. This is the number the kernel uses to uniquely identify each process; this number is assigned when the process is created, and is unique as long as the process is active;
- TTY: in the second column, it shows the so-called "control terminal" of the process. The non-interactive processes (daemons) that are used to perform a series of service tasks, having to be active even when there is no user connected, instead work in the background (for which there is no control terminal) the TTY column shows therefore a '?';
- STATUS: shows the status of the process. In fact, the system provides five different states for the processes:
    - runnable (R): the process is running or ready to run (i.e. waiting for the CPU to be assigned to it);

•sleep (S): the process is waiting for a response from the system but can be interrupted by a signal;

•uninterruptable sleep (D): the process is waiting for a response from the system (usually for I/O), and cannot be interrupted under any circumstances;

•stopped (T): the process was stopped with a SIGSTOP, or is being tracked;

•zombie (Z): the process has ended but its termination status has not yet been read by the father;

•TIME: indicates the CPU time used so far by the process

•COMMAND: reports the command line used to launch the program.

When the name of the command is in square brackets and has a very low PID it is the processes inside the kernel used by it for the management of some tasks, the notation indicates that the processes do not use memory in USERSPACE. The other letters associated with the STAT field are "<", "N" and "L" and indicate a higher or lower priority than the standard one and the presence of locked memory pages.

The **ps** options are many, but the most important are:

•u: print a list of the most relevant settings regarding the user who launched the program;

•r: print information relating to the use of virtual memory;

•s: print information on signals;

•o: allows the user to specify his own format;

•-e: allows you to select all the processes present;

•-f: allows you to have a list with more information.

The output of the **ps -ef** command is more complete, add the columns:

•UID: which is the first column, it reports a username;

•PPID: which is the third column, indicates the parent process ID. This column allows, in the case of zombies, to identify the person responsible for their production;

•C: which is the fourth column, it indicates the average value, expressed as an integer, of the percentage of CPU utilization over the entire life of the process;

•STIME: which is the fifth column, it indicates the moment in which the command was launched.

The main properties of the processes and the name of the relative column in the display performed by **ps** are:

•PID: print the process ID;

•PPID: print the father's process ID;

•UID: print the effective user ID of the process;

•GID: print the effective group ID of the process;

•CMD: print the command line with which the process was launched;

•STAT: print process status;

•NI: print the nice value of the process;

•TTY: print process control terminal;

•SID: print the session ID of the process;

•PGID: print the process group ID of the process;

•%CPU: print the percentage of the CPU time used compared to the real time of execution;

•%MEM: prints the percentage of physical memory used by the process;

•C: print the percentage of CPU on the average of the process life;

•START: print the start time of the process;

•TIME: print the total CPU time used by the process;

•USER: print the effective user of the process;

•RUSER: print the real user ID;

•GROUP: print the effective group ID of the process;

•RGROUP: print the real group ID of the process;

•COMMAND: prints the command line with which the process started.

Each process has a working directory against which it resolves relative pathnames, this characteristic is inherited in the creation of a child process. In addition to the current working directory, the process also keeps an indication of the directory it considers root, against which it resolves absolute pathnames. This directory can be changed with a **system call** (called a **chroot**) thus causing the process to be somewhat restricted within a part of the file tree.

The **ps** command, however, only prints the processes at the time of its execution, to keep the system activity under control, the **top** command is used. The command normally operates in interactive mode, however it can be launched with the options:

•-b: which runs it in batch mode, allowing output redirection;

•-n: it is used to specify the number of desired iterations;

•-d: is used to set the interval between updates (default: 1 second), requires ss.dd as parameter (where s are the seconds and d the tenths of a second);

•-p: it allows to observe a list of processes chosen by the user, who must specify it through a list of PIDs.

Five lines of system information are printed at the top:

•First line: the time, the uptime, the number of users and the average load of the machine are reported;

•Second line: statistics on the total of processes are reported;

•Third line: CPU usage statistics are reported;

•Fourth line: RAM usage statistics are reported;

•Fifth line: SWAP usage statistics are shown.

These are followed, after an empty line, by the information on the processes, sorted by decreasing CPU usage. The properties listed are:

•%CPU: print the percentage of CPU usage since the last screen update;

•%MEM: prints the percentage of memory usage by the process;

•CODE: prints the amount of physical memory used by the process for its executable code;

•DATA: prints the amount of physical memory used by the process for its data;

•RES: print the amount of physical memory used by the process (DATA + CODE);

•S: print the status of the process (similar to the STAT of **ps**);

•SHR: prints the amount of shared memory, represents the memory potentially shareable with other processes;

•SWAP: prints the amount of virtual memory of a process present in the SWAP;

•TIME+: prints the CPU time used by the start (similar to the TIME of **ps**) but gradually up to the hundredth of a second;

•VIRT: prints the total amount of virtual memory used by the process; it includes all code, data and shared libraries plus the pages that have been put into the SWAP (SWAP + REF).

If the top command is started in interactive mode, it is possible to send keyboard commands:

•K: to send a signal (default SIGTERM);

•R: to change the priority of a process;

•U: to select a user's processes;

•C: you can alternate between printing the command name and the complete line

•O: to change the update period;

•H: to print all available commands.

The initial lines shown with **top** can be obtained separately with the **free** command, which shows a general summary of the system's memory usage. The first line reports the use of physical memory, while the last one of the SWAP. The central row tells us how much memory is occupied or free from the point of view of applications, so the sum of the buffers and cached columns is subtracted and added to the value of the used and free columns in the first row.

In general, the free RAM is always low, as it makes no sense to leave the RAM unused, so it is used by the kernel buffers (to manage data transfer more effectively) and to keep temporary data. As options accept:

•-b, -k, -m: to print data in bytes, kilobytes (default) and megabytes;

•-s: to print the statistics after a certain period of time.

If you want to find the PID of a certain program, you can use **pidof**. The command takes as argument a list of program names whose PID you want to know, it can take as options:

- -s: in the case of several processes corresponding to the same program, to print a single PID;
- -o: to omit a PID.

1.3.3 The signals

Although the processes use separate and independent entities, there are many cases in which a form of communication is necessary. This communication is given by **signals**, which are also used by the kernel. The signal is a kind of warning that does not contain any information, they can be sent through the **kill** command. Running the **kill -l** command shows all 31 signals used.

Most of the signals (except **SIGKILL** and **SIGSTOP**) can be intercepted by the process, which can perform an appropriate function when they arrive. If the signal is not intercepted, a default action is performed which is specific to each of them (in most cases it consists in terminating the program). This can happen in two ways:

- Simple exit;
- Exit with copy: a directory is created, of a **core** file that contains a copy of its memory space (and for this reason it is called **core dump**) which can be used for further analysis to see where the interruption occurred.

WORK IN PROGRESS