Санкт-Петербургский Национальный Исследовательский Университет Информационных Технологий, Механики и Оптики
ФКТиУ, кафедра Вычислительной техники

Лабораторная работа №3

по дисциплине

«Системное программное обеспечение»

Выполнил:
Tran Trung Duc
Студент группы P3202

Преподаватель: доцент кафедры ВТ
Дергачев Андрей Михайлович

2019

How commands work in background

Some shell commands take a long time to execute. These commands can be run in the background using &, freeing the terminal for other tasks. The general format for running commands in the background is as follows:

```
command &
```

Note: Interactive shell commands (for example, read) cannot be run in the background.

In order to find jobs running in the background, you can run the following command:

```
jobs
```

Example output:

```
[1]-  Stopped                 vi /etc/passwd
[2]+  Stopped                 vi /etc/passwd
```

New command

1) ps [options]
Report a snapshot of the current processes
Basic key: • -A : all process; • -a : all processes except both session leaders and processes not associated with a terminal; • -d : all processes except session leaders; • r : restrict the selection to only running processes; Display information: • UID: user ID; • PID: process ID; • PPID: parent process ID; • C: processor utilization; • START: starting time or date of the process; • TTY: controlling tty (terminal) • TIME: cumilative CPU time, "[DD-]HH:MM:SS" format; • CMD: see args; • STAT: multi-character process state.

2) crontab
Cron - maintain crontab files for individual users Crontab - is the program used to install, deinstall or list the tables used to drive the cron. crontab [-u user] file crontab [-u user] [-i] { -e | -l | -r }
Basic key: **-r** the current crontab to be removed. **-l** the current crontab to be displayed on standard output. See the note under DEBIAN SPECIFIC below

Each line in file *etc*/crontab has format :
minute hour day month dayofweek command

In addition, variables were invented for some frequently used sets, so they are:
- **@reboot** – at boot time, only once;
- **@yearly, @annually** – once in year;
- **@monthly** – once in month;
- **@weekly** – once in week;

3) at <time> <data>

at and **batch** read commands from standard input or a specified file which are to be executed at a later time, using /bin /sh
at executes commands at a specified time.

Basic key:
After executing the at command with the time and date, you will be moved to the simplest interactive command shell to set the execution script.
- m Send mail to the user when the job has completed even if there was no output.

Example:
```
echo "ping -c 4 www.google.com" | at -m now +1 minute
```

4) nice [OPTION] [COMMAND [ARG]...]

Nice - run a program with modified scheduling priority
The nice priority and the priority of the OS kernel process scheduler are different numbers. The nice number is the priority the user would like to assign to the process. Scheduler priority is the actual priority assigned to the process by the scheduler.

Basic key:
```
-n, --adjustment=N
    add integer N to the niceness (default 10)
```
Example:
```
$ nice -n 15 yes > /dev/null &
```

5) nohup

Nohup- run a command immune to hangups, with output to a non-tty
nohup COMMAND [ARG]...
nohup OPTION

Example:
```
$ nohup yes > /dev/null &
```
(Kill process with kill utilities)

6) fg

The command to resume the task and transfer control to it in POSIX compatible shells. The command allows you to resume a paused process or take it out of the background.

Example
```
$ fg %1
```

```
yes > /dev/null
```

7) bg

The resuming execution of the suspended process in the background, the resuming process continues to run in the background without the user entering any commands from the terminal. Enabling the bg command is a prerequisite for any OS, so that the latter can be considered POSIX-compatible.

Example:

```
$ yes > /dev/null &
$ Ctrl-Z
$ bg
```

8) kill

The Kill command sends the specified signal to the specified process. If no signal is specified, a SIGTERM signal is sent. The SIGTERM signal terminates only those processes that do not process its arrival. For other processes, it may be necessary to send a SIGKILL signal, since this signal cannot be intercepted.

Basic key:

-l output list signal names

-s Specify the signal to be sent. The signal can be specified by using name or number

9) jobs

Display status of jobs..

Examples

```
$ jobs
[1]-    Stopped                                 vi  /etc/passwd    (wd:
~/Desktop/Fouthsem/SPO/lab1)
[2]+    Stopped                                 vi  /etc/passwd    (wd:
~/Desktop/Fouthsem/SPO/lab1)
```

10) &

when added at the end of a command, launching it (the command) takes place in the background. Exceptions are interactive commands that require user interaction.

Example:

```
yes > /dev/null &
```

About command kill

When you execute the "kill" command, in fact, you send a signal to the system to force it to terminate incorrectly the application itself.

In total, up to 60 different signals can be used, but the most frequently used ones are SIGTERM (15) and SIGKILL (9). (The various signals sent by this OS command can be viewed using the -l key. **Kill -l**)

Signals - software interrupts. They are used for communication between processes in UNIX and UNIX-like operating systems, such as Linux, Mac OS.

SIGTERM - this signal requests to stop the process. It can be ignored. The process is given time to correct completion. If the program finishes correctly, it means that it used this time to save its state or results of work and free up resources. In other words, she was not forced to stop.

SIGKILL - this signal causes the process to stop working immediately. The program cannot ignore this signal. Unsaved results will be lost.

Groups processes and sessions

A process group is a group of related processes that are considered together for the purpose of managing a job. Processes with the same process group

Setsid () - creates a new session if the calling process does not create a group. The calling process becomes the lead in the group, the lead process of the new session and does not have a controlling terminal. The identifiers of the process group and session at installation will be equal to the identifier of the calling process. The calling process will be the only one in this group and session.

Processes, dispatcher and scheduler

The term "process" first appeared in the development of the Multix operating system and has several definitions that are used depending on context. The process is:

1. program at the implementation stage
2. "object" to which processor time is allocated
3. asynchronous operation

To describe the state of the process uses several models. The simplest model is the three-state model. The model consists of:

Execution is an active state during which the process has all the resources it needs.

Waiting is a passive state, during which the process is blocked, it cannot be executed, because it is waiting for some event, for example, data entry or release of the device it needs.

Readiness is also a passive state, the process is also blocked, but unlike the waiting state, it is blocked not for internal reasons, but for external, process-independent reasons.

The process can go into the ready state if, during its execution, the run-time quantum is “exited”. In other words, the operating system has a special program - the scheduler, which ensures that all processes run their allotted time.

Init process

This process is created immediately when the system is started, that is, all other processes are its descendants.

During the boot process, after the kernel is initialized, the kernel starts / sbin / init as the first user mode process. init is responsible for further system boot. To do this, it runs the so-called startup scripts that check and mount file systems, run the necessary daemons, configure the kernel (including loading kernel modules according to the installed equipment, setting up IP addresses, routing tables, etc.), launching the graphical shell and other actions.

On Unix / Linux operating systems, with init you can change the initialization level. Initialization level - the degree of loading of the operating system. Here is how the system initializes: the init process starts and analyzes the / etc / inittab file.

By default, the system uses 7 levels of initialization:

- 0 - system stop
- 1 - boot in single user mode
- 2 - download in multi-user mode without network support
- 3 - load in multi-user mode with network support
- 4 - not used
- 5 - load in multi-user mode with network support and graphical login
- 6 - reboot

Start and put into background

- example of running the command `vi / etc / passwd 'in the background

```
> vi / etc / passwd &
```

- example of transferring the command `vi / etc / passwd 'to background mode

```
> vi / etc / passwd
Ctrl-Z
> bg
```