**TASK5.3**

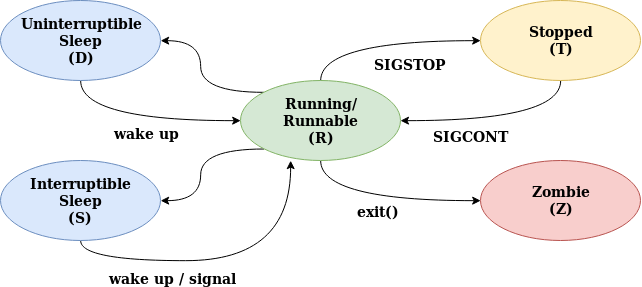
**Part1**

1. **How many states could has a process in Linux?**

Linux has basically 5 states:

* **Running/Runnable (R):** running processes are processes using a CPU core right now, a runnable process is a process that has everything to run and is just waiting for a CPU core slot.
* **Sleeping:** a sleeping process is a process waiting for a resource to be available (for example, a I/O operation to complete) or an event to happen (like a certain amount of time to pass). The difference between process in **Interruptible Sleep (S)** state and **Uninterruptible Sleep (D)** is that the former will wake up to handle signals while the former won't. We'll talk about signals in a moment, but let's suppose that a process is waiting for a I/O operation to complete before wake up. If in the meantime, it receives a signal to terminate (**SIGKILL**), it will terminate before having the chance to handle the requested data. That's why I/O operations *normally* go to uninterruptible sleep while waiting for the result: they will wake up with when the operation is ready, handle the result and, only then, check for any pending signal to handle. Processes that can be terminated before the wake up condition is fulfilled without any consequence usually go to interruptible sleep instead.
* **Stopped (T):** a process becomes stopped when it receives the **SIGSTOP**signal (not unlike when you press <ctrl>+z in the shell, although <ctrl>+z sends a **SIGTSTP** instead). When stopped, the process execution is suspended and the only signals it will handle are **SIGKILL** and **SIGCONT**. The former will remove the process permanently, while the later will put the process back to the *Running/Runnable* state (like when you run fg or bg after pressing <ctrl>+z in the shell).
* **Zombie (Z):**we briefly talked about zombie processes when we discussed [system calls](https://medium.com/@cloudchef/linux-system-calls-c2867c7c30c1). When a process finishes with exit() system call, its state needs to be "reaped" by its parent (calling wait()); in the meantime, the child process remains in *zombie* state (not alive nor dead).

The diagram below helps understand the transition between process states:



1. Examine the pstree command. Make output (highlight) the chain (ancestors) of the current process.

**pstree** is a [Linux](https://en.wikipedia.org/wiki/Linux" \o "Linux) command that shows the running [processes](https://en.wikipedia.org/wiki/Process_(computing)" \o "Process (computing)) as a [tree](https://en.wikipedia.org/wiki/Tree_(data_structure)). It is used as a more visual alternative to the [ps](https://en.wikipedia.org/wiki/Ps_(Unix)" \o "Ps (Unix)) command. The root of the tree is either [init](https://en.wikipedia.org/wiki/Init" \o "Init) or the process with the given [pid](https://en.wikipedia.org/wiki/Process_identifier" \o "Process identifier). It can also be installed in other [Unix](https://en.wikipedia.org/wiki/Unix" \o "Unix) systems.

In [BSD](https://en.wikipedia.org/wiki/BSD) systems, a similar output is created using ps -d, in Linux ps axjf[[1]](https://en.wikipedia.org/wiki/Pstree" \l "cite_note-1) produces similar output.

**3. What is a proc file system?**

The **proc filesystem** (**procfs**) is a special filesystem in [Unix-like](https://en.wikipedia.org/wiki/Unix-like" \o "Unix-like) operating systems that presents information about [processes](https://en.wikipedia.org/wiki/Process_(computing)" \o "Process (computing)) and other system information in a hierarchical file-like structure, providing a more convenient and standardized method for dynamically accessing process data held in the kernel than traditional [tracing](https://en.wikipedia.org/wiki/Tracing_(software)" \o "Tracing (software)) methods or direct access to [kernel](https://en.wikipedia.org/wiki/Kernel_(computing)" \o "Kernel (computing)) memory. Typically, it is mapped to a [mount point](https://en.wikipedia.org/wiki/Mount_point" \o "Mount point) named */proc* at boot time. The proc file system acts as an interface to internal data structures about running processes in the kernel. In [Linux](https://en.wikipedia.org/wiki/Linux_kernel), it can also be used to obtain information about the kernel and to change certain kernel parameters at runtime ([sysctl](https://en.wikipedia.org/wiki/Sysctl" \o "Sysctl)).

Many Unix-like operating systems support the proc filesystem, including [Solaris](https://en.wikipedia.org/wiki/Solaris_(operating_system)" \o "Solaris (operating system)), [IRIX](https://en.wikipedia.org/wiki/IRIX), [Tru64 UNIX](https://en.wikipedia.org/wiki/Tru64_UNIX), [BSD](https://en.wikipedia.org/wiki/Berkeley_Software_Distribution), [Linux](https://en.wikipedia.org/wiki/Linux), [IBM AIX](https://en.wikipedia.org/wiki/IBM_AIX), [QNX](https://en.wikipedia.org/wiki/QNX), and [Plan 9 from Bell Labs](https://en.wikipedia.org/wiki/Plan_9_from_Bell_Labs" \o "Plan 9 from Bell Labs). [OpenBSD](https://en.wikipedia.org/wiki/OpenBSD" \o "OpenBSD) dropped support in version 5.7, released in May 2015.

The [Linux kernel](https://en.wikipedia.org/wiki/Linux_kernel" \o "Linux kernel) extends it to non–process-related data.

The proc filesystem provides a method of communication between [kernel space](https://en.wikipedia.org/wiki/Kernel_space" \o "Kernel space) and [user space](https://en.wikipedia.org/wiki/User_space" \o "User space). For example, the [GNU](https://en.wikipedia.org/wiki/GNU) version of the process reporting utility [ps](https://en.wikipedia.org/wiki/Ps_(Unix)" \o "Ps (Unix)) uses the proc file system to obtain its data, without using any specialized [system calls](https://en.wikipedia.org/wiki/System_calls" \o "System calls).

1. Print information about the processor (its type, supported technologies, etc.).
2. Use the ps command to get information about the process. The information should be as follows: the owner of the process, the arguments with which the process was launched for execution, the group owner of this process, etc. 6. How to define kernel processes and user processes?

7. Print the list of processes to the terminal. Briefly describe the statuses of the processes. What condition are they in, or can they be arriving in?

8. Display only the processes of a specific user. 9. What utilities can be used to analyze existing running tasks (by analyzing the help for the ps command)?

**10. What information does top command display?**

Верхняя часть программы показывает краткую обобщённую информацию об использовании процессора и оперативной памяти системы.

В самой верхней строке показано: текущее время в системе, аптайм (время заботы после загрузки), общее количество пользователей и средняя нагрузка за последние 1, 5 и 15 минут.

Далее идут строки с информацией о:

* задачах
* процессоре
* оперативной памяти
* разделе подкачки

Далее идёт перечень запущенных процессов. По умолчанию выводиться следующая информация:

**PID** — уникальный идентификатор процесса

**USER** — имя пользователя, являющегося владельцем задачи

**PR** — приоритет задачи в расписании. Если вы в этом поле видите «rt», это означает, что задача запущена в расписании приоритетов в реальном времени.

**NI** — значение nice задачи. Негативное значение означает более высокий приоритет, а положительное значение nice означает более низкий приоритет

**VIRT** — общее количество используемой задачей виртуальной памяти, включает все коды, данные, совместные библиотеки, плюс страницы, которые были перенесены в раздел подкачки, и страницы, которые были размечены, но не используются

**RES** — используемая оперативная память, является подмножеством VIRT, представляет физическую память, не помещённую в раздел подкачки, которую в текущий момент использует задача. Также является суммой полей RSan, RSfd и Rssh.

**SHR** — размер совместной памяти, подмножество используемой памяти RES, которая может использоваться другими процессами

**S** — статус процесса. Может быть:

* **D** = бесперебойный сон
* **I** = простой
* **R** = запущен
* **S** = спит
* **T** = остановлен сигналом управления работой
* **t** = остановлен отладчиком во время трассировки
* **Z** = зомби

**%CPU**— использование центрального процессора, доля задачи в потреблённом процессорном времени с момента последнего обновления экрана, выражается в процентах от общего времени CPU

**%MEM** — доля задачи в использовании памяти (RES)

**TIME+** — общее время центрального процессора, которое использовала задача с момента запуска

**COMMAND** — Имя команды или Строка команды. Показывает строку команды, используемую для запуска задачи или имя ассоциированной программы

12. Display the processes of the specific user using the top command.

12. What interactive commands can be used to control the top command? Give a couple of examples.

Полезные интерактивные команды, которые можно использовать в top:

* [1] Отобразить всю статистику по всем ядрам.
* [c] Абсолютный путь расположения модуля команды и её аргументы.
* [h] Вывести справку о программе.
* [k] Уничтожить процесс. Программа запрашивает у вас код процесса и сигнал, который будет ему послан.
* [M] Сортировать по объёму используемой памяти.
* [n] Изменить число отображаемых процессов. Вам предлагается ввести число.
* [P] Сортировать по загрузке процессора.
* [r] Изменить приоритет процесса.
* [u] Сортировать по имени пользователя.
* [Z] Выбрать цвет подсветки.
* [z] Подсветить работающие процессы.
* [Пробел] Немедленно обновить содержимое экрана.

13. Sort the contents of the processes window using various parameters (for example, the amount of processor time taken up, etc.)

14. Concept of priority, what commands are used to set priority?

15. Can I change the priority of a process using the top command? If so, how? 16. Examine the kill command. How to send with the kill command

process control signal? Give an example of commonly used signals.

17. Commands jobs, fg, bg, nohup. What are they for? Use the sleep, yes command to demonstrate the process control mechanism with fg, bg.

**Part2**

1. Check the implementability of the most frequently used OPENSSH commands in the MS Windows operating system. (Description of the expected result of the commands + screenshots: command – result should be presented)

* [ssh(1)](https://man.openbsd.org/ssh) — The basic rlogin/rsh-like client program
* [sshd(8)](https://man.openbsd.org/sshd) — The daemon that permits you to log in
* [ssh\_config(5)](https://man.openbsd.org/ssh_config) — The client configuration file
* [sshd\_config(5)](https://man.openbsd.org/sshd_config) — The daemon configuration file
* [ssh-agent(1)](https://man.openbsd.org/ssh-agent) — An authentication agent that can store private keys
* [ssh-add(1)](https://man.openbsd.org/ssh-add) — Tool which adds keys to in the above agent
* [sftp(1)](https://man.openbsd.org/sftp) — FTP-like program that works over SSH1 and SSH2 protocol
* [scp(1)](https://man.openbsd.org/scp) — File copy program that acts like rcp
* [ssh-keygen(1)](https://man.openbsd.org/ssh-keygen) — Key generation tool
* [sftp-server(8)](https://man.openbsd.org/sftp-server) — SFTP server subsystem (started automatically by sshd)
* [ssh-keyscan(1)](https://man.openbsd.org/ssh-keyscan) — Utility for gathering public host keys from a number of hosts
* [ssh-keysign(8)](https://man.openbsd.org/ssh-keysign) — Helper program for host-based authentication

2. Implement basic SSH settings to increase the security of the client-server connection (at least 3. List the options for choosing keys for encryption in SSH. Implement 3 of them.

4. Implement port forwarding for the SSH client from the host machine to the guest Linux virtual machine behind NAT.

5\*. Intercept (capture) traffic (tcpdump, wireshark) while authorizing the remote client on the server using ssh, telnet, rlogin. Analyze the result.