Министерство Образования, Культуры и Исследований

**Молдавский Государственный Университет**

**Факультет Математики и Информатики**

**Департамент Информатики**

**Лабораторная работа 5**

**“Межпроцессное взаимодействие Linux”**

Научный руководитель: А. Prepelita

Автор: студент группы I2102 (РУ), Juracovschi Vladislav

**Кишинев, 2023**

Оглавление

[Просмотр общей информации о ядре 3](#_Toc8681)

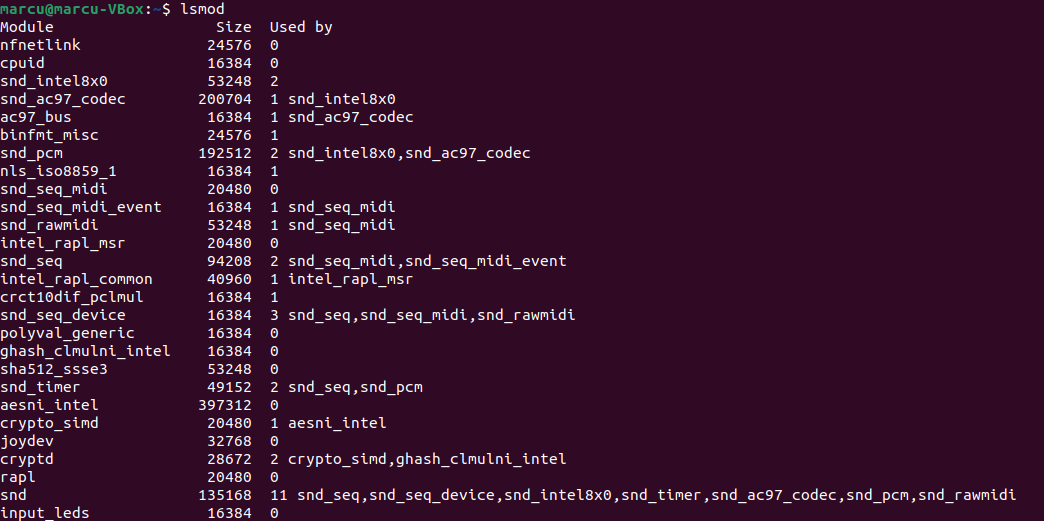
[Состояния процессов Linux/Unix Zombie, Сироты 4](#_Toc126)

[Функции в языке программирования С 7](#_Toc23092)

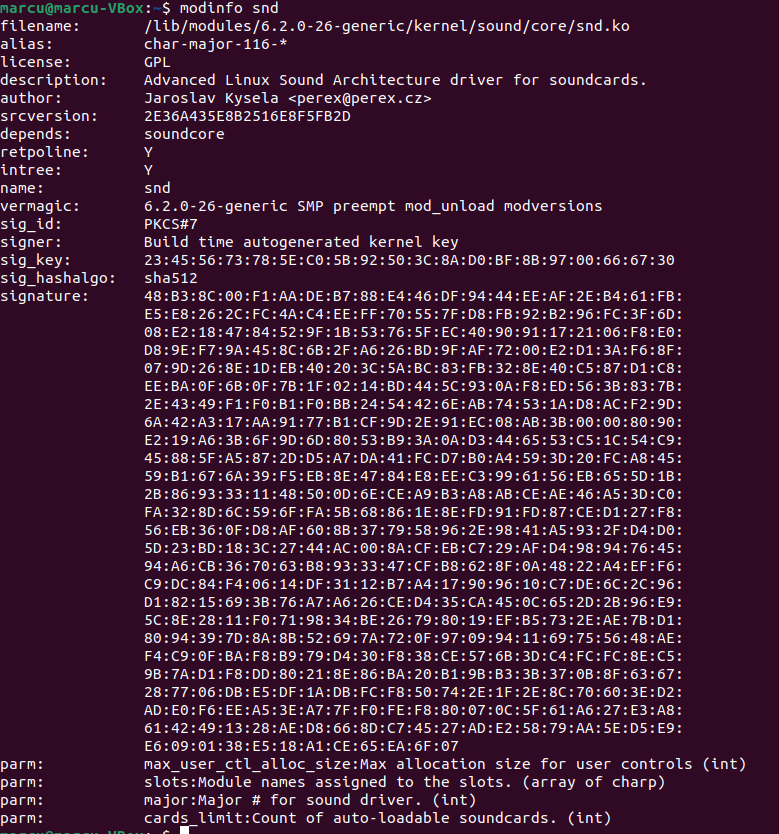
Просмотр общей информации о ядре



Эта команда выводит информацию о системе, включая имя ядра, версию ядра, тип процессора и другие параметры.

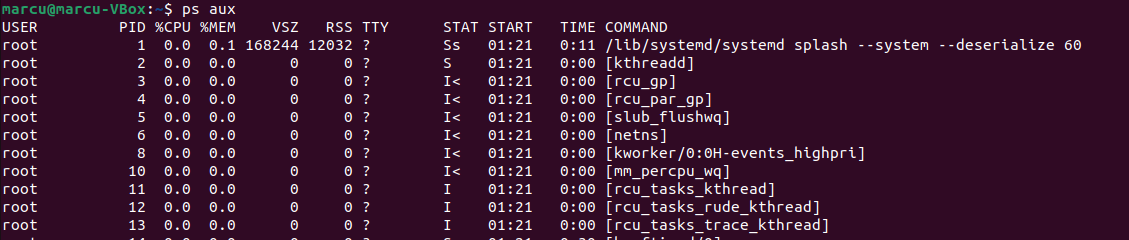


lsmod показывает загруженные модули ядра в системе.



modinfo предоставляет подробную информацию о загруженном модуле ядра. Заменить <module\_name> именем конкретного модуля.

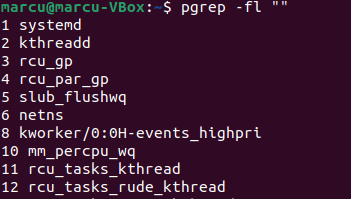
# Состояния процессов Linux/Unix Zombie, Сироты



ps aux выводит информацию о текущих процессах в системе



grep -w Z: Фильтрует вывод, оставляя только строки, в которых столбец STAT содержит значение Z, что соответствует Zombie/-процессам.



Эта команда показывает список процессов вместе с их именами в состоянием Сирота.

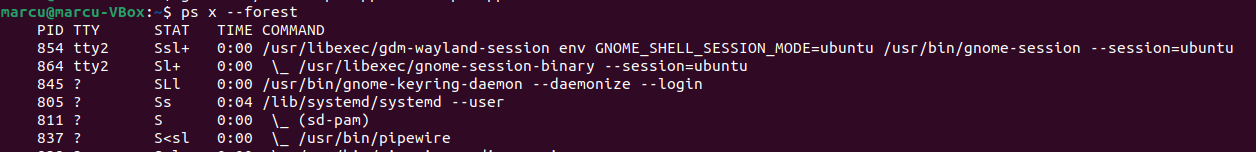
1. Команды для компиляции с использованием GCC и G++:



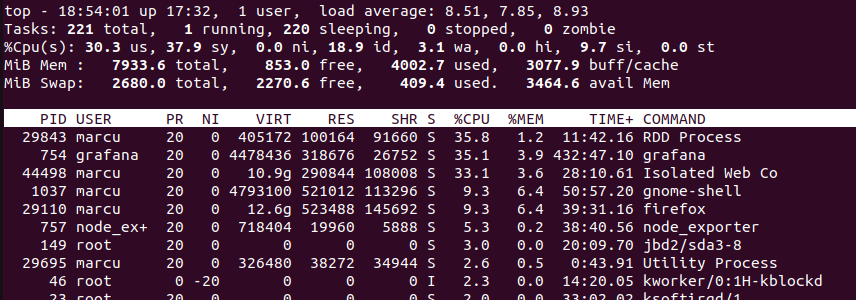
Компиляция C-программы:



Компиляция C++-программы:



Вывод процессов в виде дерева:



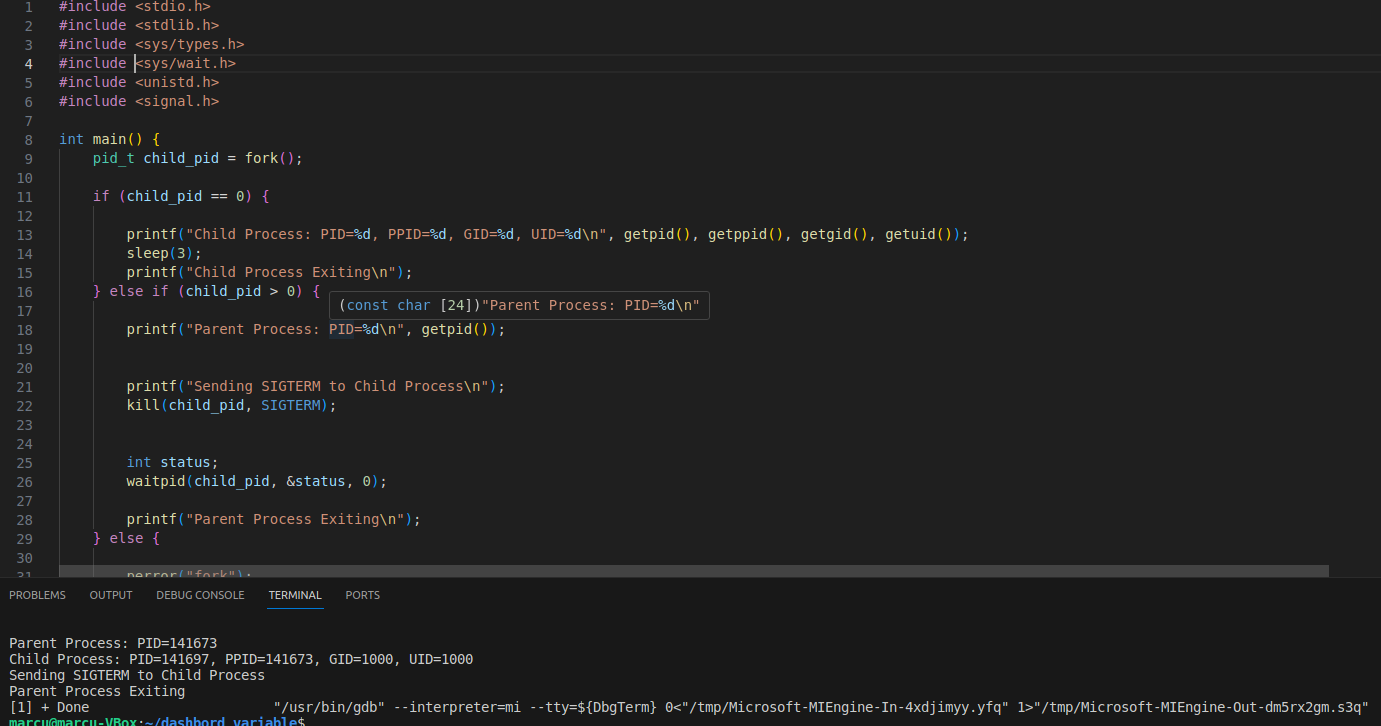
top предоставляет динамическое представление текущей активности системы и списка процессов.



kill используется для выключение сигналов процессам.

# Функции в языке программирования С

fork() создает новый процесс, который является копией текущего процесса.



#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <signal.h>

#include <sys/wait.h>

int main() {

pid\_t child\_pid = fork();

if (child\_pid == 0) {

printf("Child Process: PID=%d, PPID=%d, GID=%d, UID=%d\n", getpid(), getppid(), getgid(), getuid());

sleep(3);

printf("Child Process Exiting\n");

} else if (child\_pid > 0) {

printf("Parent Process: PID=%d\n", getpid());

printf("Sending SIGTERM to Child Process\n");

kill(child\_pid, SIGTERM);

int status;

waitpid(child\_pid, &status, 0);

printf("Parent Process Exiting\n");

} else {

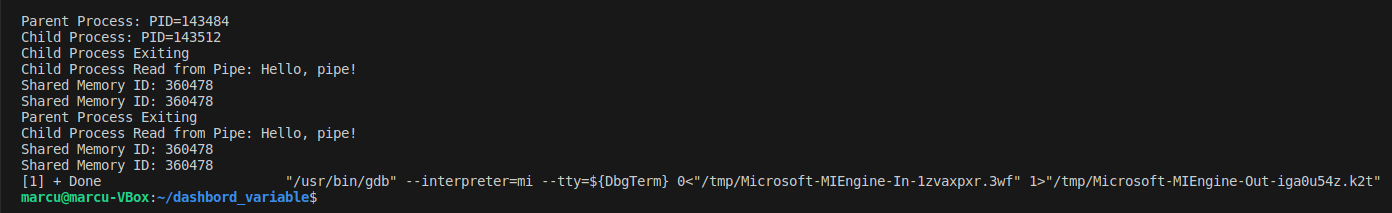
perror("fork");

exit(EXIT\_FAILURE);

}

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <fcntl.h>

#include <sys/stat.h>

int main() {

// Example of wait() and waitpid()

pid\_t child\_pid = fork();

if (child\_pid == 0) {

// Child process code

printf("Child Process: PID=%d\n", getpid());

sleep(3);

printf("Child Process Exiting\n");

} else if (child\_pid > 0) {

// Parent process code

printf("Parent Process: PID=%d\n", getpid());

// Using wait() to wait for any child process to exit

wait(NULL);

printf("Parent Process Exiting\n");

} else {

// Error handling

perror("fork");

exit(EXIT\_FAILURE);

}

// Example of pipe(), read(), and write()

int my\_pipe[2];

char buffer[20];

if (pipe(my\_pipe) == -1) {

perror("pipe");

exit(EXIT\_FAILURE);

}

pid\_t child\_pipe\_pid = fork();

if (child\_pipe\_pid == 0) {

// Child process code for pipe

close(my\_pipe[1]); // Close the write end

// Read data from the pipe

read(my\_pipe[0], buffer, sizeof(buffer));

printf("Child Process Read from Pipe: %s\n", buffer);

close(my\_pipe[0]); // Close the read end

} else if (child\_pipe\_pid > 0) {

// Parent process code for pipe

close(my\_pipe[0]); // Close the read end

// Write data to the pipe

write(my\_pipe[1], "Hello, pipe!", 12);

close(my\_pipe[1]); // Close the write end

} else {

// Error handling

perror("fork");

exit(EXIT\_FAILURE);

}

// Example of mkfifo, fopen/fclose (or open/close), and shmget

// Using mkfifo to create a named pipe

mkfifo("my\_fifo", 0666);

// Using fopen/fclose to work with a file

FILE \*file = fopen("my\_file.txt", "w");

if (file != NULL) {

fprintf(file, "Hello, file!\n");

fclose(file);

} else {

perror("fopen");

exit(EXIT\_FAILURE);

}

// Using shmget to create a shared memory segment

key\_t key = ftok("shm\_example", 'R');

int shm\_id = shmget(key, 1024, IPC\_CREAT | 0666);

if (shm\_id == -1) {

perror("shmget");

exit(EXIT\_FAILURE);

} else {

printf("Shared Memory ID: %d\n", shm\_id);

}

return 0;

}