**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

**Лабораторная работа №1**

по дисциплине: Операционные системы

тема: «Базовая работа с процессами в ОС Linux. Сигналы»

Выполнил: ст. группы ПВ-233

Мовчан Антон Юрьевич

Проверили:

пр. Четвертухин Виктор Романович

Белгород 2025 г.

**Лабораторная работа №1  
Структура команд процессора  
Вариант 8**

**Цель работы:** Изучить основы работы с процессами в ОС Linux, а также освоить сигналы как универсальное средство межпроцессного взаимодействия.

8. Построить полное бинарное дерево процессов глубины D, в котором любой лист по

специальному сигналу может инициировать корректное завершение всех процессов дерева, причём оповещения (сигналы) допускается распространять только вдоль рёбер дерева междусмежными процессами (ребёнок ⟷ родитель). Пересылка между несмежными вершинами запрещена.

**Код программы:**

#define \_POSIX\_C\_SOURCE 200809L

#include *<stdio.h>*

#include *<stdlib.h>*

#include *<unistd.h>*

#include *<signal.h>*

#include *<sys/wait.h>*

#include *<string.h>*

#define D 3 *// глубина дерева, можно изменить*

**volatile** sig\_atomic\_t stop\_flag = 0;

void create\_tree(int depth, int max\_depth, pid\_t parent\_pid);

void gf\_handler(int signo)

{

stop\_flag = 1;

}

pid\_t create\_child(int depth, int max\_depth, pid\_t parent\_pid)

{

pid\_t pid = fork();

**if** (pid == 0)

{

create\_tree(depth + 1, max\_depth, getppid());

exit(0);

}

**if** (pid < 0)

perror("Ошибка порождения процесса!");

**return** pid;

}

void leaf\_logic(pid\_t parent\_pid)

{

printf("[Leaf %d] Ожидаю SIGINT для завершения**\n**", getpid());

**while** (!stop\_flag)

pause();

printf("[Leaf %d] Получил SIGINT, отправляю родителю %d**\n**", getpid(), parent\_pid);

**if** (parent\_pid > 0)

kill(parent\_pid, SIGINT);

printf("[Leaf %d] Завершаюсь**\n**", getpid());

exit(0);

}

void node\_logic(pid\_t left, pid\_t right, pid\_t parent\_pid, int depth)

{

**while** (!stop\_flag)

pause();

**if** (left > 0)

kill(left, SIGINT);

**if** (right > 0)

kill(right, SIGINT);

**if** (parent\_pid > 0)

kill(parent\_pid, SIGINT);

**if** (left > 0)

waitpid(left, NULL, 0);

**if** (right > 0)

waitpid(right, NULL, 0);

printf("[PID %d] Завершаюсь на глубине %d**\n**", getpid(), depth);

exit(0);

}

void create\_tree(int depth, int max\_depth, pid\_t parent\_pid)

{

pid\_t left = -1, right = -1;

**if** (depth < max\_depth)

{

left = create\_child(depth, max\_depth, parent\_pid);

right = create\_child(depth, max\_depth, parent\_pid);

}

printf("[PID %d] Стартовал на глубине %d, parent=%d, left=%d, right=%d**\n**", getpid(), depth, parent\_pid, left, right);

**if** (depth == max\_depth)

leaf\_logic(parent\_pid);

node\_logic(left, right, parent\_pid, depth);

}

int main()

{

setvbuf(stdout, NULL, \_IONBF, 0);

signal(SIGINT, gf\_handler);

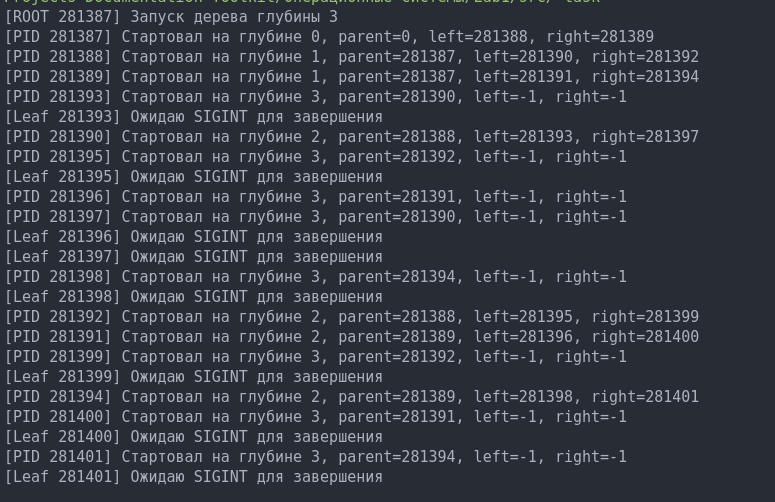
printf("[ROOT %d] Запуск дерева глубины %d**\n**", getpid(), D);

create\_tree(0, D, 0);

**return** 0;

}

Вывод программы:



**Вывод по коду:**

Код реализует создание бинарного процесса-дерева с помощью системных вызовов fork и сигналов POSIX. Вот краткий обзор:

Программа строит дерево процессов заданной глубины (D, по умолчанию 3).

Каждый процесс (узел дерева) порождает двух потомков, пока не достигнута максимальная глубина.

Листовые процессы (на последнем уровне) ждут сигнала SIGINT, после чего уведомляют родителя и завершаются.

Внутренние узлы после получения SIGINT пересылают сигнал своим детям и родителю, дожидаются завершения потомков и завершаются сами.

Ключевые функции:

create\_tree — рекурсивно строит дерево процессов, порождая левого и правого потомка.

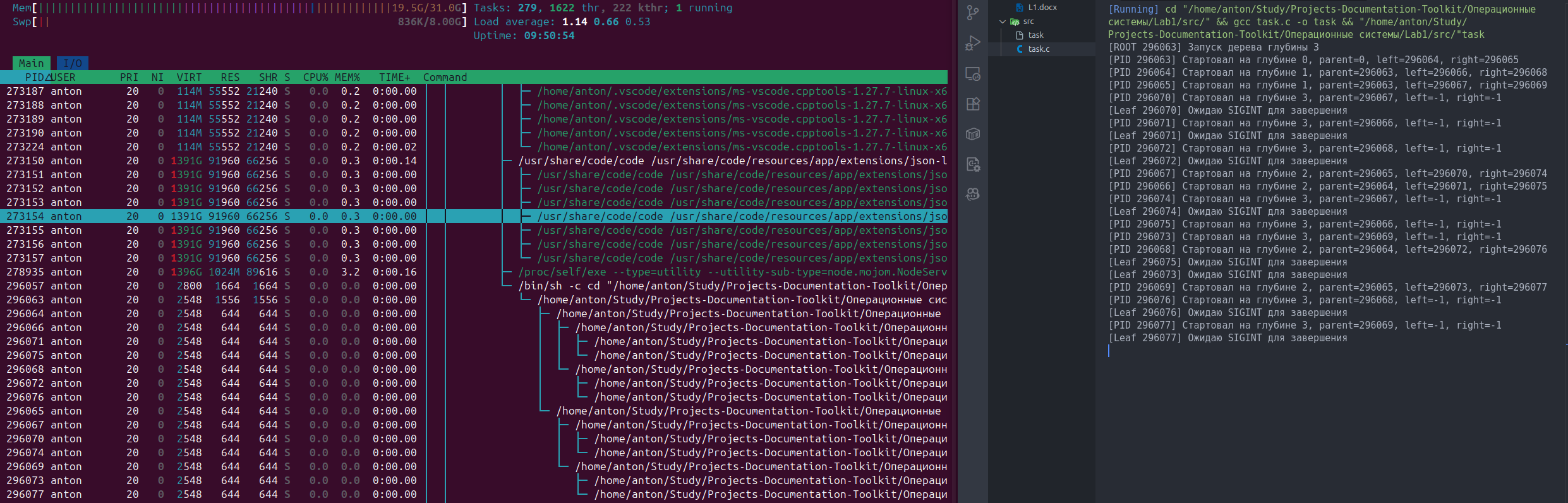
create\_child — обертка над fork, запускает новый процесс и вызывает для него create\_tree.

leaf\_logic — логика листового процесса: ожидание SIGINT, уведомление родителя, завершение.

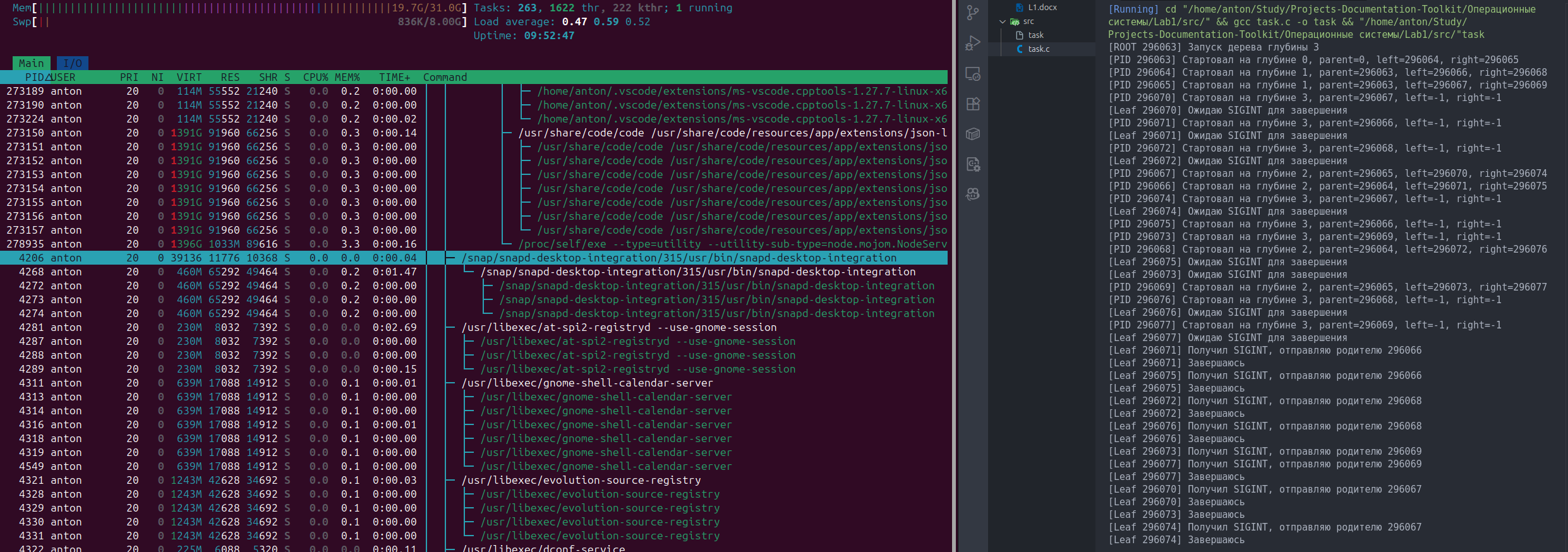
node\_logic — логика внутреннего узла: ожидание SIGINT, пересылка сигнала детям и родителю, ожидание завершения детей, завершение.

gf\_handler — обработчик SIGINT, устанавливает флаг завершения.

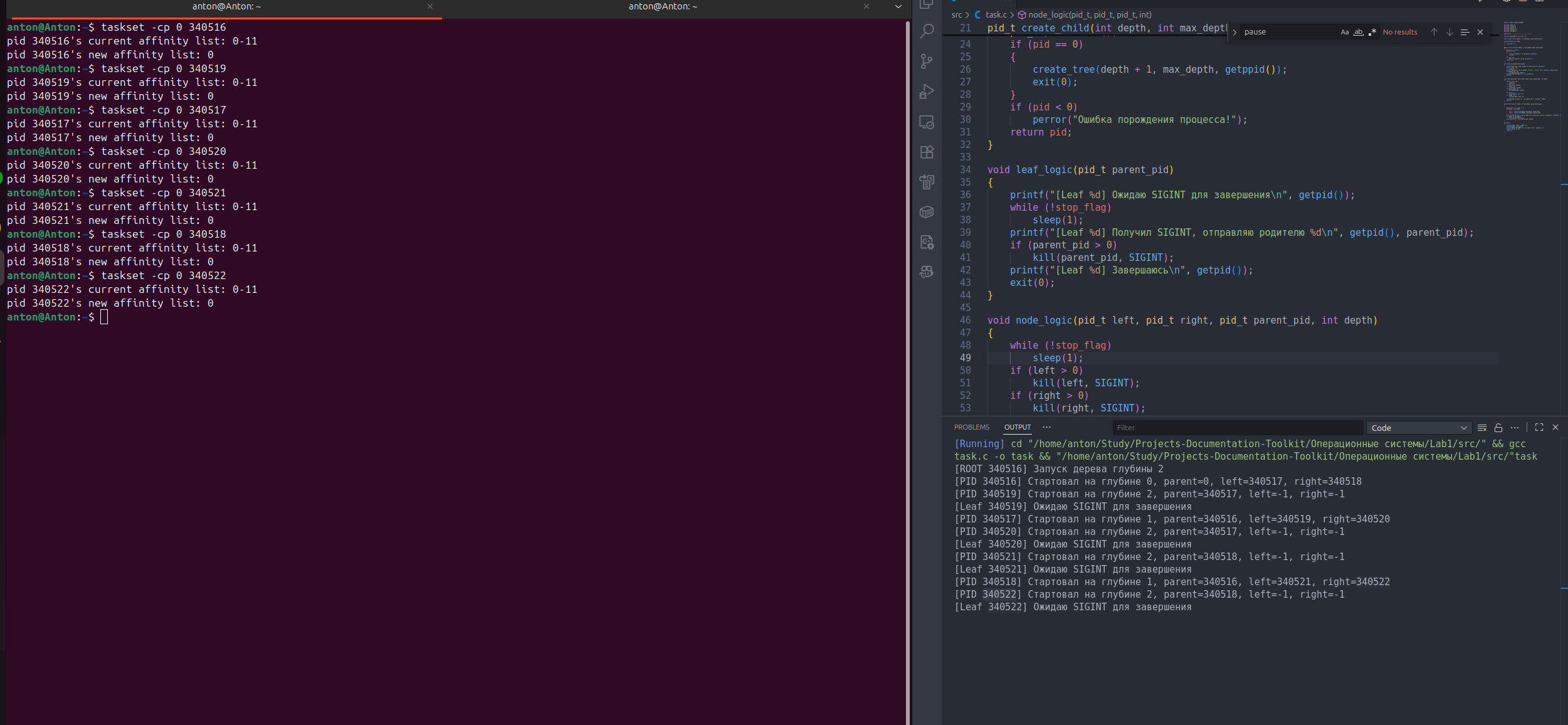
Вывод htop при работающей программе:



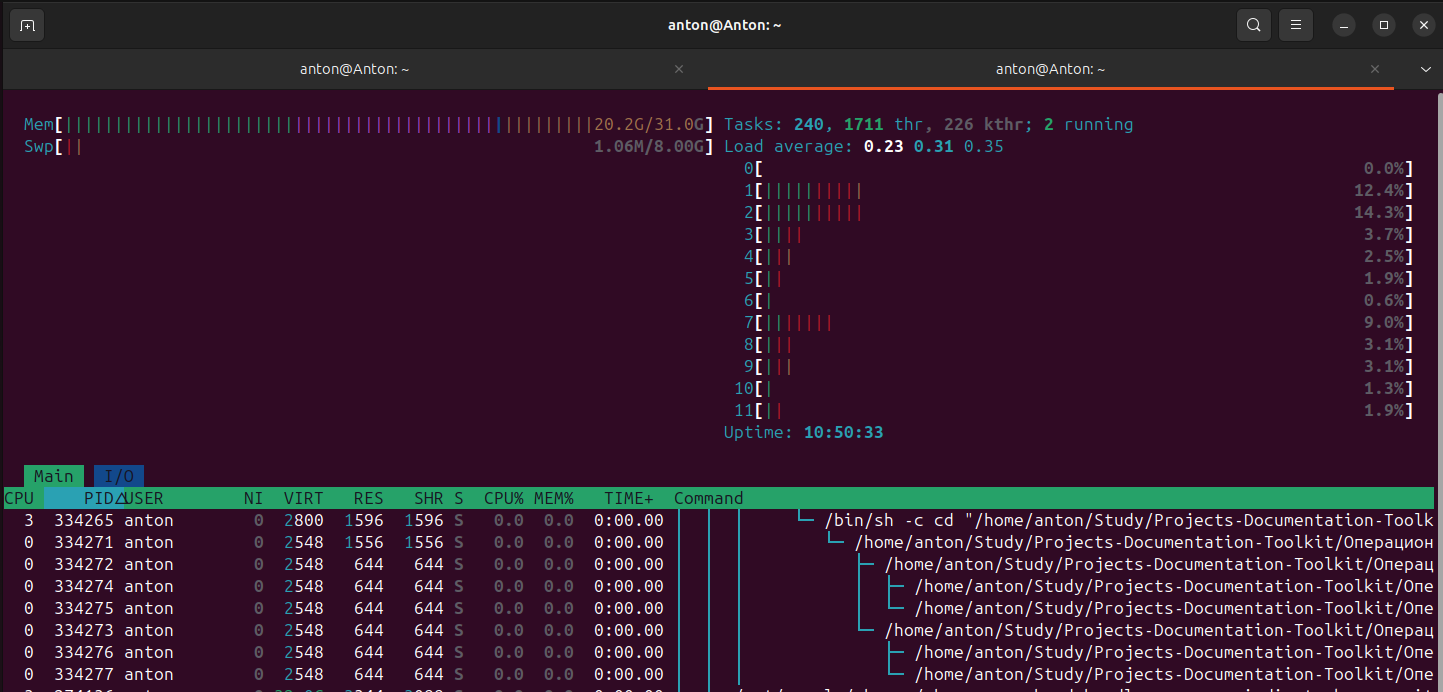
Вывод htop после закрыти узла дерева через SIGINT (необходимо выбрать процесс, нажать F9 и выбрать 2.SIGINT, далее нажать ENTER):

s

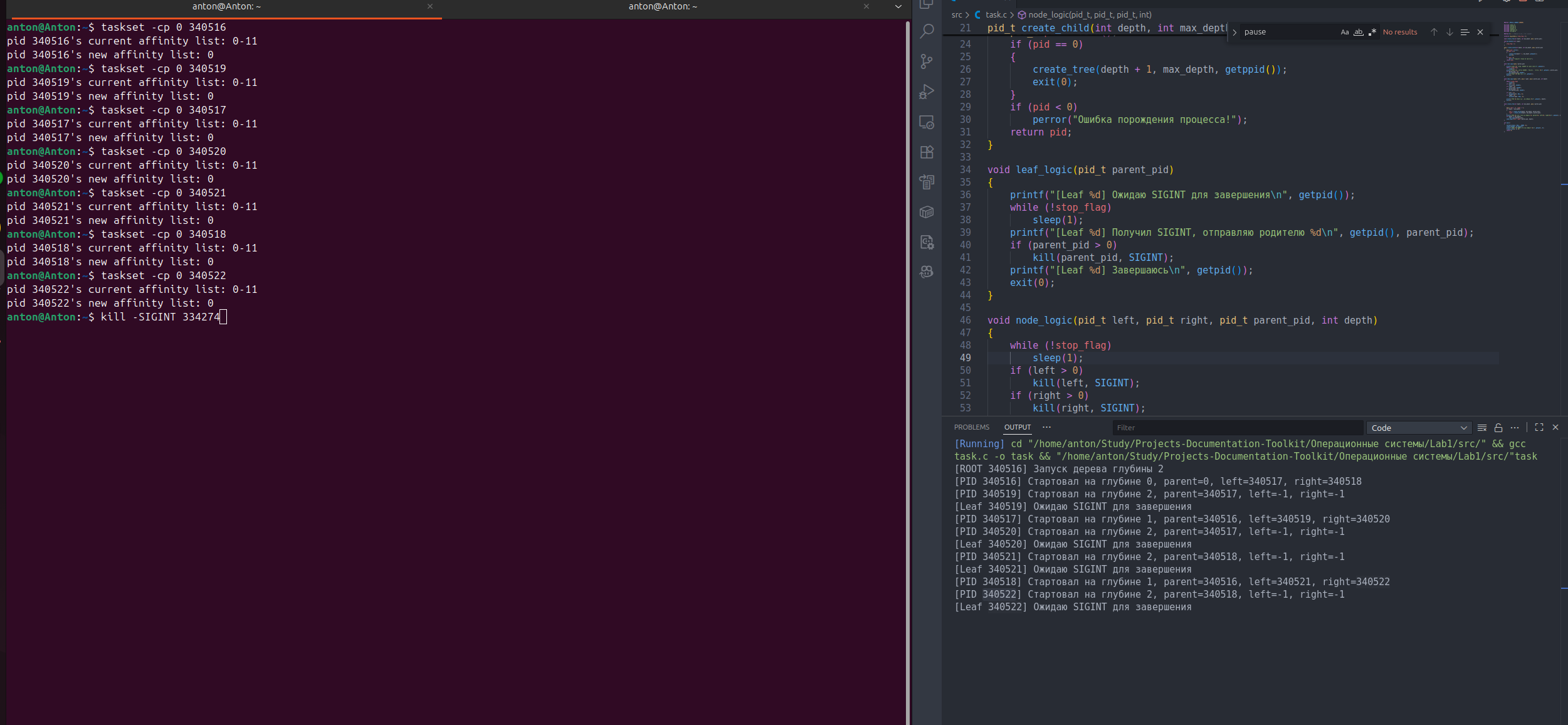
Через утилиту taskset задал PIDам нужный номмер ядра (0).



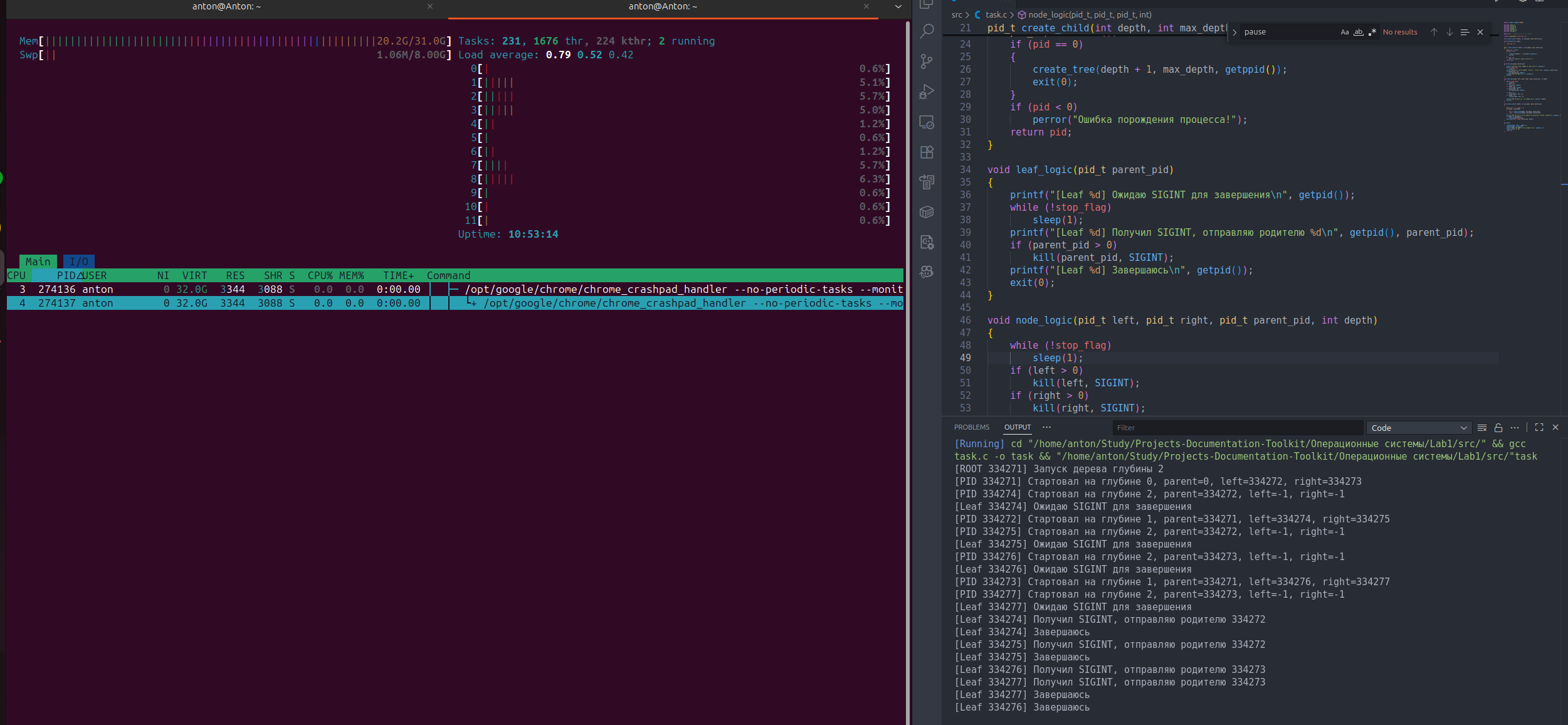
В htop отобразил задачи и номера ядра на которых они работаю.



С помощью команды killall -SIGINT (PID процеса), завершил любой процесс.



После выполнения команды, все связанные с ним процессы были также закрыты.

**Вывод:** в ходе л.р изучил структуру команд процессора, научился составлять машинный код простейших команд.

Полученные результаты изменились. Когда планировщик ОС самостоятельно распределяет процессы по ядрам, нагрузка равномерно распределяется между всеми доступными ядрами, что позволяет эффективнее использовать ресурсы процессора и уменьшает время выполнения процессов за счёт параллелизма.

При ручной привязке всех процессов к одному ядру (через sched\_setaffinity или taskset), все процессы конкурируют за вычислительное время только на одном ядре. Это приводит к увеличению времени выполнения каждого процесса и снижению общей производительности из-за очередей на выполнение и переключения контекста.

Ограничение всех процессов одним ядром устраняет преимущества многопроцессорных систем, снижая степень параллелизма и увеличивая нагрузку на отдельное ядро, что негативно сказывается на производительности.

**Вывод:** в ходе выполнения л.р я изучил основы работы с процессами в ОС Linux, а также освоил сигналы как универсальное средство межпроцессного взаимодействия.