МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Системное программирование»

Лабораторная работа №7

**«Управление процессами. Службы»**

по дисциплине

Операционные системы

Выполнил: студент гр. БЭИ2202

Кулешов А. С.

Вариант 16

Проверил: Королькова Т.В.

Москва, 2025 г

Задание 1. Мониторинг производительности системы

1. Выполните проверку системы при помощи команды top. Отсортируйте

процессы по: объему используемой памяти; времени работы;

идентификатору; проценту использования времени процессора.

2. Запустите новый процесс в системе, найдите его в выводе команды top.

Измените приоритет запущенного процесса командой renice и убедитесь,

что изменения отражены в выводе top.

3. С помощью команд atop и atopsar выполните сортировку процессов

минимум по трем параметрам общей нагрузки.

4. При помощи команды mpstat и опции P выведите информацию по:

определенному процессору, всем процессорам.

5. Выполните проверку системы при помощи команды pidstat. Выведите по

четыре отчета статистики использования процессора для каждой активной

задачи в системе с интервалом в три секунды.

В отчете приведите ответ в виде снимков экрана по каждому пункту задания.

Задание 2. Создание скрипта для получения статистики

Напишите скрипт, осуществляющий сбор статистики использования

процессоров и статистики работы процессов средствами mpstat и pidstat,

выводящий данные в текстовый файл с именем yyyy-mm-dd\_h:m:s\_syslog.

В отчет включите текст скрипта, содержимое файла лога и вывод

команды ls -l для директории сохранения логов скрипта.

Задание 3. Создание автозапускаемой службы

С помощью утилиты systemd зарегистрируйте свою службу по сбору

статистики, запускающую скрипт из задания 2 при старте системы. Запустите

службу и проверьте ее работу.

В отчете приведите снимок экрана с выводом команды ls -l для

директории сохранения логов скрипта, статистику выполнения службы и текст

файла .service.

**ВЫПОЛНЕНИЕ**

Выполню

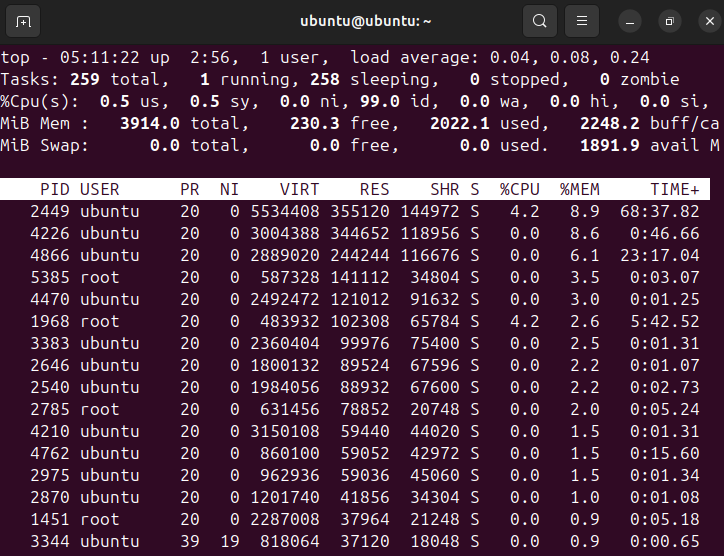


Рисунок 1 –Команда top по памяти(M)

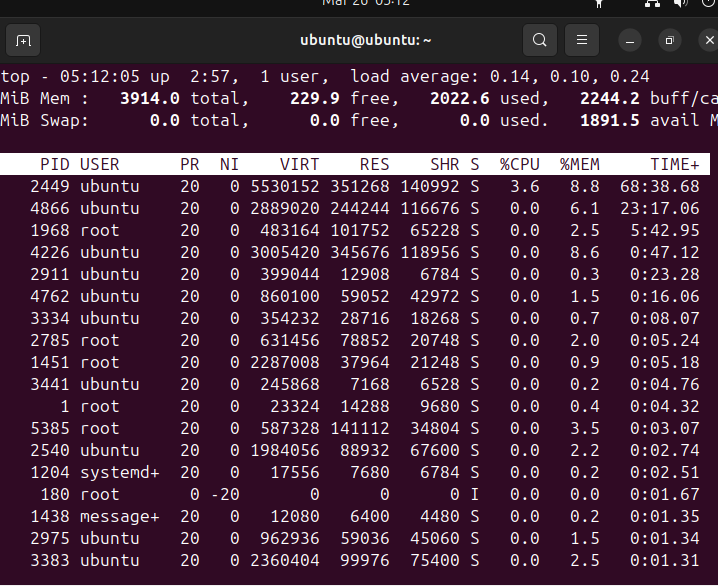


Рисунок 2 – Команда top по времени работы (T)

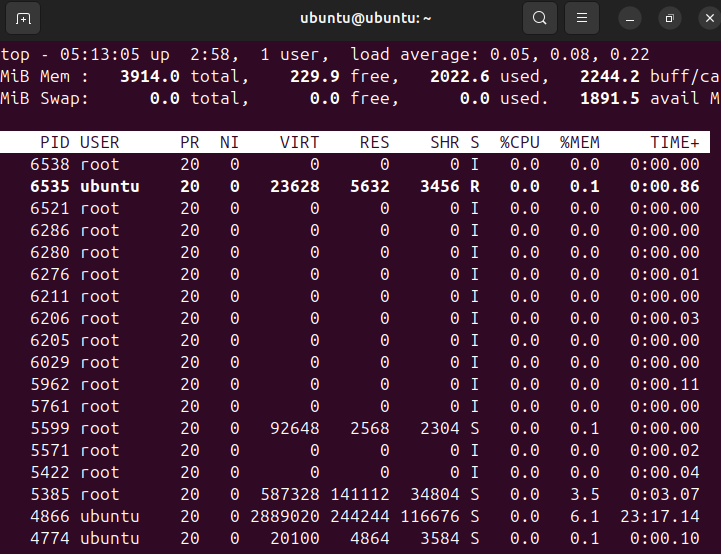


Рисунок 3 – Команда top по PID (N)

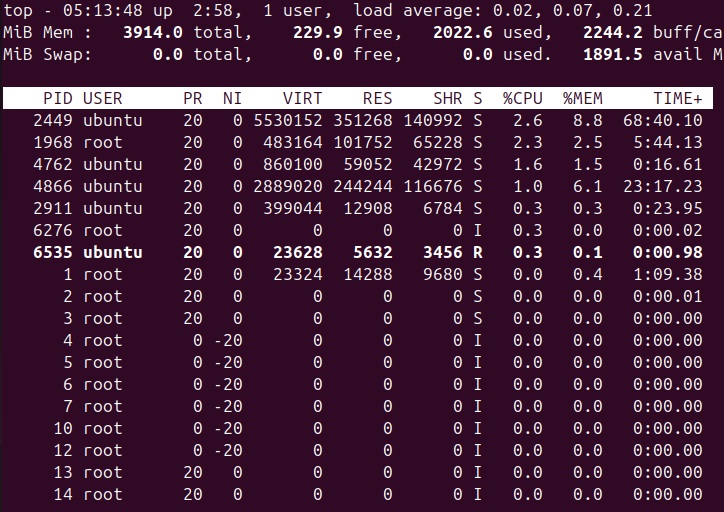


Рисунок 4 – Команда top по загрузке процессора (P)



Рисунок 5 – Создали пустой процесс

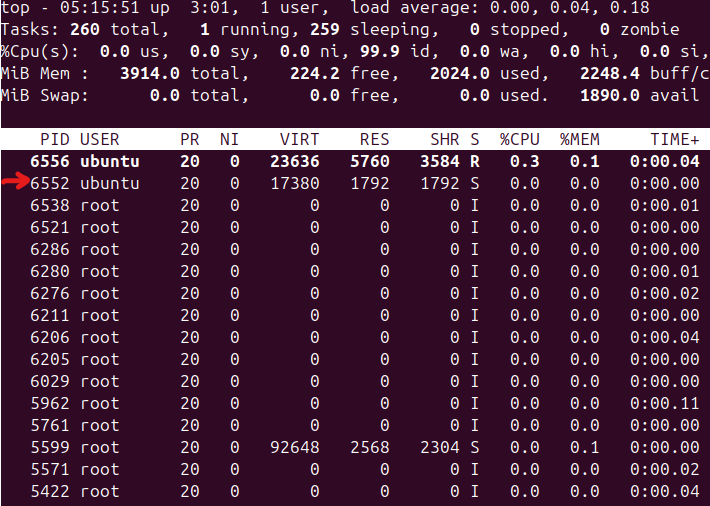


Рисунок 6 – Процесс найден в топ (отсортировано по PID)



Рисунок 7 – Изменение приоритета

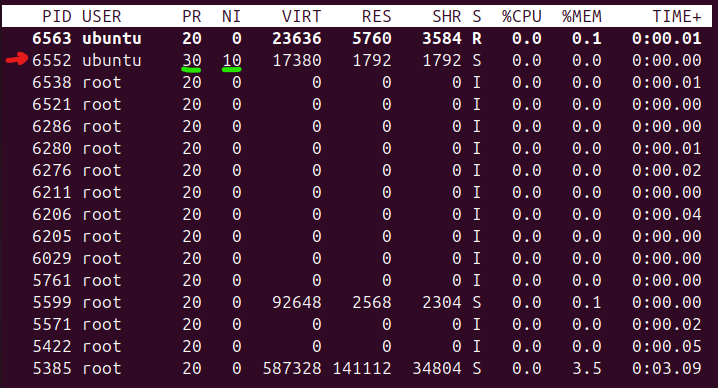


Рисунок 8 – Процесс найден в топ с новым приоритетом (отсортировано по PID)

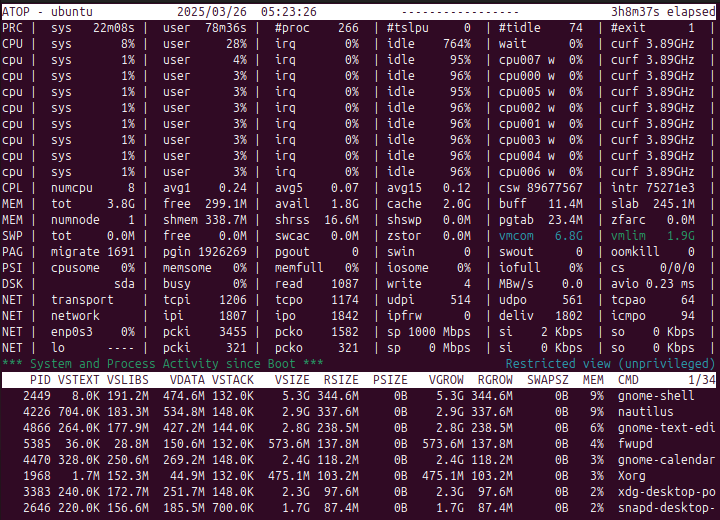


Рисунок 9 – Команда atop –m (память)

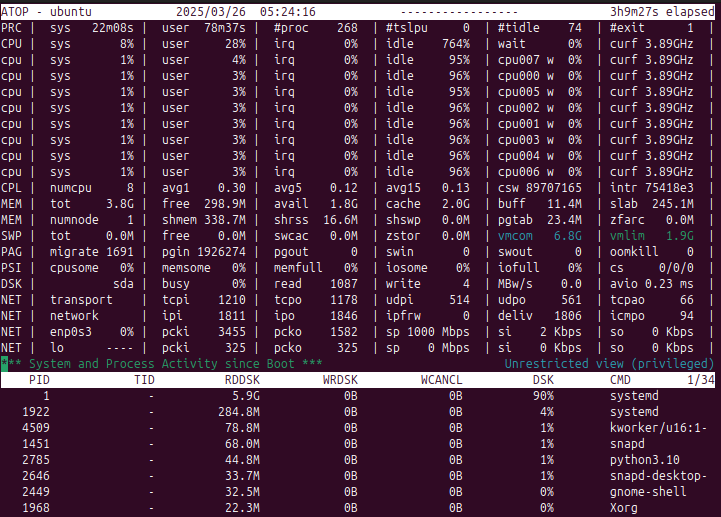


Рисунок 10 – Команда atop –d (диск)

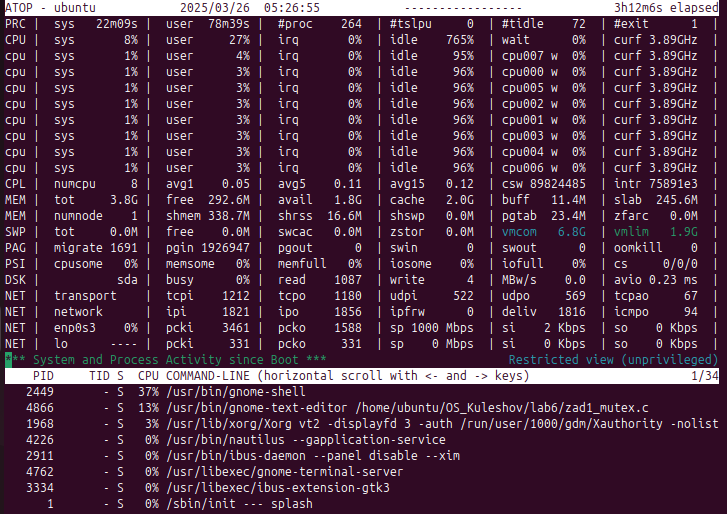


Рисунок 11 – Команда atop –c (процессор)



Рисунок 12 – Команда atop лог файл

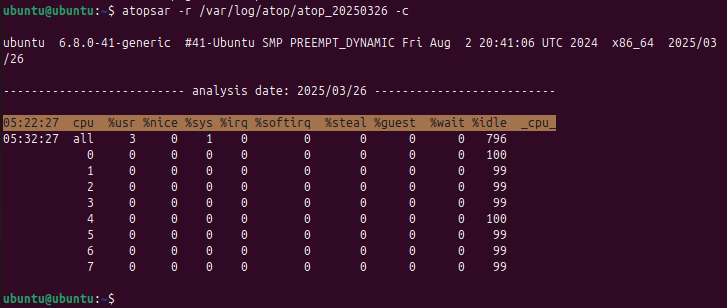


Рисунок 12 – Команда atopsar

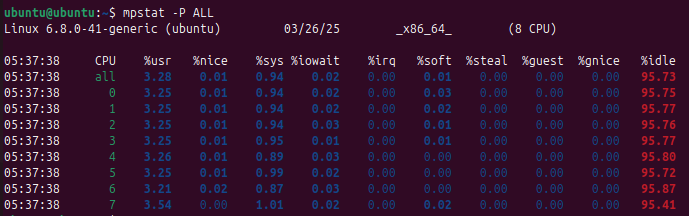


Рисунок 13 – Статистика нагрузки по всем процессорам

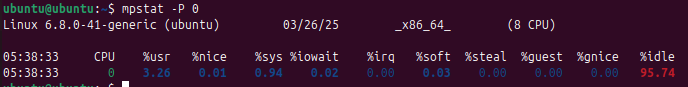


Рисунок 14 – Статистика по нулевому процессору

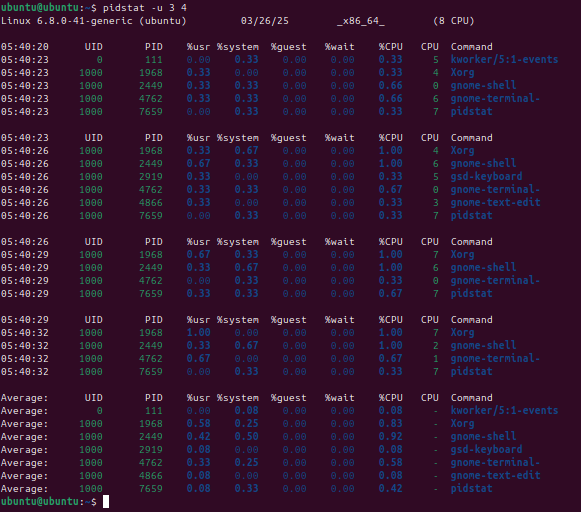


Рисунок 15 – Вывод статистике по загрузке процессора (4 раза с интервалом в 3 секунды)



Рисунок 16 – Скрипт для сбора статистики

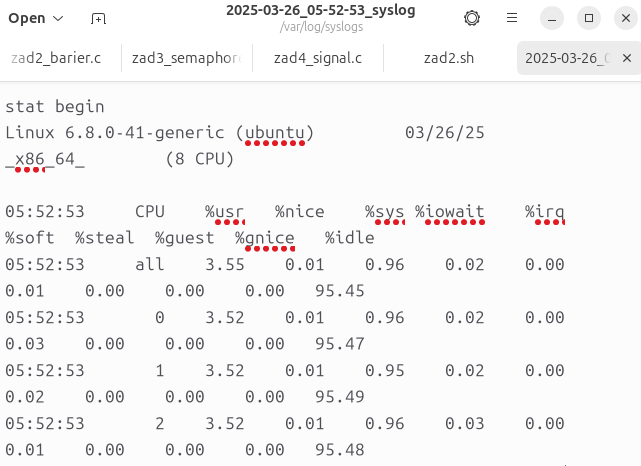


Рисунок 17 – Содержимое логов

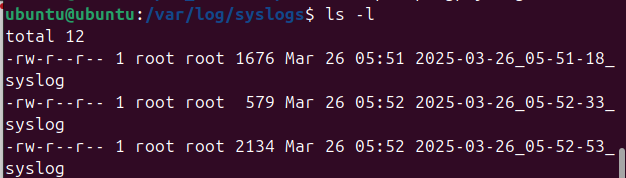


Рисунок 18 – ls –l



Рисунок 19 – Создание файла службы

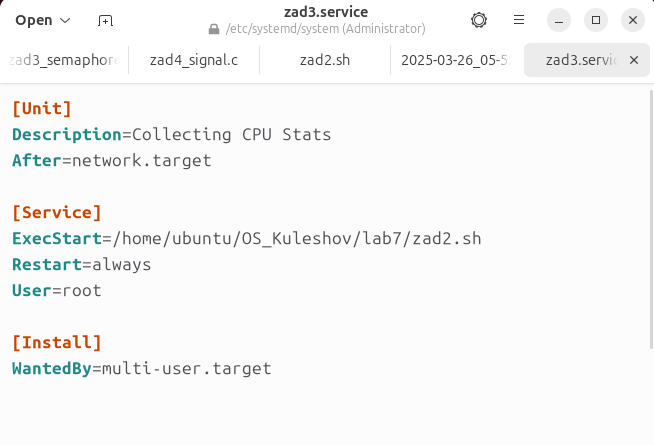


Рисунок 20 – Файл службы

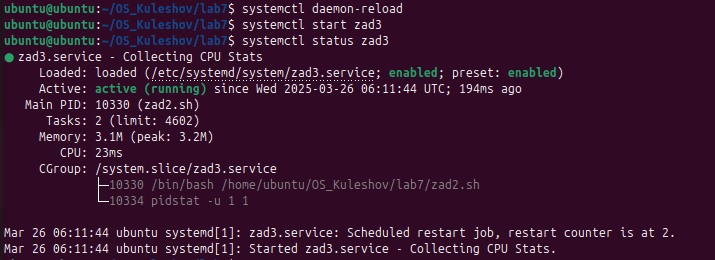


Рисунок 21 – Запуск службы

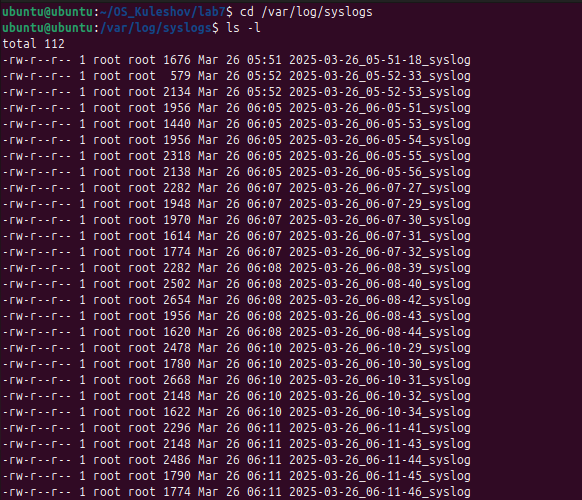


Рисунок 22 – Полученные логи

Код для задания №2.

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#define SIZE 20

#define NUM\_THREADS 4

#define GENERATIONS 50

**int** current[SIZE][SIZE];

**int** next[SIZE][SIZE];

**pthread\_barrier\_t** barrier;

**void** **initialize\_board**() {

**for** (**int** i = **0**; i < SIZE; i++) {

**for** (**int** j = **0**; j < SIZE; j++) {

current[i][j] = rand() % **2**;

}

}

}

**void** **print\_board**() {

system("clear");

**for** (**int** i = **0**; i < SIZE; i++) {

**for** (**int** j = **0**; j < SIZE; j++) {

printf(current[i][j] ? "O " : ". ");

}

printf("**\n**");

}

usleep(**200000**);

}

**int** **count\_neighbors**(**int** x, **int** y) {

**int** count = **0**;

**for** (**int** dx = -**1**; dx <= **1**; dx++) {

**for** (**int** dy = -**1**; dy <= **1**; dy++) {

**if** (dx == **0** && dy == **0**) **continue**;

**int** nx = (x + dx + SIZE) % SIZE;

**int** ny = (y + dy + SIZE) % SIZE;

count += current[nx][ny];

}

}

**return** count;

}

**void**\* **game\_of\_life**(**void**\* arg) {

**int** thread\_part = \*(**int**\*)arg;

**int** start = thread\_part \* (SIZE / NUM\_THREADS);

**int** end = (thread\_part + **1**) \* (SIZE / NUM\_THREADS);

**for** (**int** gen = **0**; gen < GENERATIONS; gen++) {

**for** (**int** i = start; i < end; i++) {

**for** (**int** j = **0**; j < SIZE; j++) {

**int** neighbors = count\_neighbors(i, j);

**if** (current[i][j] == **1**) {

next[i][j] = (neighbors == **2** || neighbors == **3**) ? **1** : **0**;

} **else** {

next[i][j] = (neighbors == **3**) ? **1** : **0**;

}

}

}

pthread\_barrier\_wait(&barrier);

// only for one thread

**if** (thread\_part == **0**) {

**for** (**int** i = **0**; i < SIZE; i++) {

**for** (**int** j = **0**; j < SIZE; j++) {

current[i][j] = next[i][j];

}

}

print\_board();

}

pthread\_barrier\_wait(&barrier);

}

**return** NULL;

}

**int** **main**() {

**pthread\_t** threads[NUM\_THREADS];

**int** thread\_ids[NUM\_THREADS];

srand(time(NULL));

initialize\_board();

print\_board();

pthread\_barrier\_init(&barrier, NULL, NUM\_THREADS);

**for** (**int** i = **0**; i < NUM\_THREADS; i++) {

thread\_ids[i] = i;

pthread\_create(&threads[i], NULL, game\_of\_life, &thread\_ids[i]);

}

**for** (**int** i = **0**; i < NUM\_THREADS; i++) {

pthread\_join(threads[i], NULL);

}

pthread\_barrier\_destroy(&barrier);

**return** **0**;

}

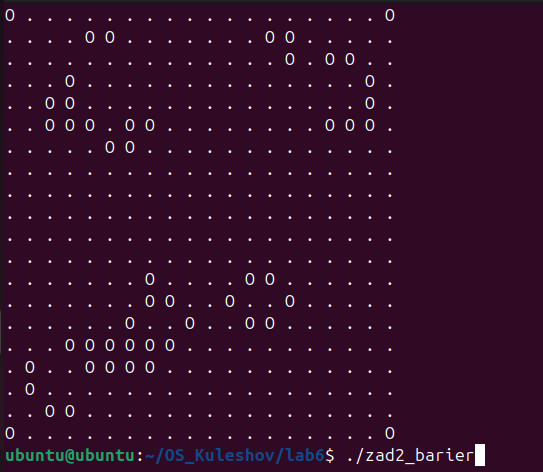


Рисунок 2 – Задание 2

Код для задания №3.

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

#define NUM\_THREADS 4

#define ARRAY\_SIZE 100

**int** array[ARRAY\_SIZE];

**int** partial\_sums[NUM\_THREADS];

**int** total\_sum = **0**;

**sem\_t** semaphore;

**void**\* **compute\_sum**(**void**\* arg) {

**int** thread\_id = \*((**int**\*)arg);

**int** start = thread\_id \* (ARRAY\_SIZE / NUM\_THREADS);

**int** end = (thread\_id == NUM\_THREADS - **1**) ? ARRAY\_SIZE : start + (ARRAY\_SIZE / NUM\_THREADS);

**int** sum = **0**;

**for** (**int** i = start; i < end; i++) {

sum += array[i];

}

partial\_sums[thread\_id] = sum;

printf("Thread %d: inter sum = %d**\n**", thread\_id, sum);

sem\_wait(&semaphore);

total\_sum += sum;

sem\_post(&semaphore);

**return** NULL;

}

**int** **main**() {

**pthread\_t** threads[NUM\_THREADS];

**int** thread\_ids[NUM\_THREADS];

srand(time(NULL));

**for** (**int** i = **0**; i < ARRAY\_SIZE; i++) {

array[i] = rand() % **100**;

}

sem\_init(&semaphore, **0**, **1**);

**for** (**int** i = **0**; i < NUM\_THREADS; i++) {

thread\_ids[i] = i;

pthread\_create(&threads[i], NULL, compute\_sum, &thread\_ids[i]);

}

**for** (**int** i = **0**; i < NUM\_THREADS; i++) {

pthread\_join(threads[i], NULL);

}

sem\_destroy(&semaphore);

printf("Total: %d**\n**", total\_sum);

**return** **0**;

}

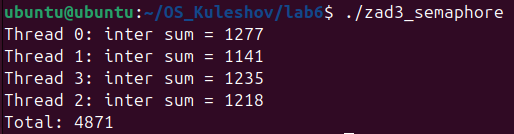


Рисунок 3 – Задание 3

Код для задания №4.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <sys/wait.h>

#include <time.h>

**const** **int** NUM\_DIRECTIONS = **4**;

**int** traffic\_light;

**void** **change\_traffic\_light**(**int** signalo) {

traffic\_light = (traffic\_light + **1**) % NUM\_DIRECTIONS;

}

**void** **road\_process**(**int** road\_id) {

srand(time(NULL) ^ getpid());

**while** (**1**) {

sleep(rand() % **5** + **1**);

**int** target\_road;

**do** {

target\_road = rand() % NUM\_DIRECTIONS;

} **while** (target\_road == road\_id);

printf("Auto: WANT %d -> %d **\n**", road\_id, target\_road);

**while** (traffic\_light != road\_id) {

usleep(**100000**);

}

printf("Auto: GO %d -> %d **\n**", road\_id, target\_road);

}

}

**int** **main**() {

traffic\_light = **0**;

signal(SIGALRM, change\_traffic\_light);

**int** childs[NUM\_DIRECTIONS];

**for** (**int** i = **0**; i < NUM\_DIRECTIONS; i++) {

**pid\_t** pid = fork();

**if** (pid == **0**) {

road\_process(i);

} **else**{

childs[i] = pid;

}

}

**while**(**1**){

kill(childs[**0**], SIGALRM);

kill(childs[**1**], SIGALRM);

kill(childs[**2**], SIGALRM);

kill(childs[**3**], SIGALRM);

change\_traffic\_light(**0**);

printf("Lights: StateChange to %d**\n**", traffic\_light);

sleep(**5**);

}

**return** **0**;

}

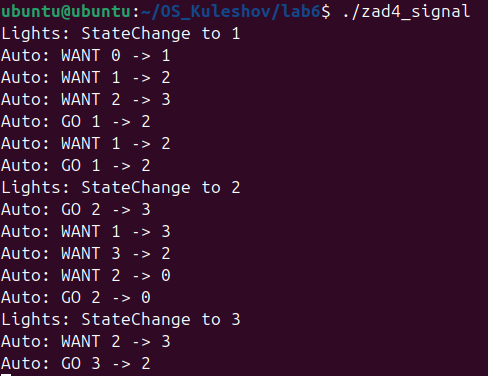


Рисунок 4 – Задание 4