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: «Системные вызовы. Базовая работа с процессами в ОС Linux.»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
доц. Островский Алексей Мичеславович
асс. Четвертухин Виктор Романович

Белгород 2024 г.

Цель работы: изучить основы работы с системными вызовами и процессами в операционной системе Linux (Ubuntu).

Условие индивидуального задания: Создать путем порождения процессов двоичное дерево из 7-ми вершин (процессов) со связями «родитель-потомок» путем последовательных вызовов функции fork(). В этом дереве каждый процесс (кроме листьев) должен порождать двух потомков. Превратить дерево в граф, путем замещения одного листа корнем. Корректно завершить все процессы. Осуществлять проверку программы путем мониторинга процессов через утилиты (ps или top).

Ход выполнения работы

Текст программы:

main.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>

#define MAX_LEVEL 3
#define ARRAY_SIZE 20
#define DELAY 400
#define FIFO_NAME "./os_lab_1_1_%d"

// Сортировка вставками, искусственная нагруженность
// создается при помощи usleep
int insertion_sort(int *array, int size)
{
    for (int i = 0; i < size; i++)
    {
        int j = i;
        while (j >= 1 && array[j] < array[j - 1])
        {
            int tmp = array[j];
            array[j] = array[j - 1];
            array[j - 1] = tmp;
            usleep(DELAY * 1000);
            j--;
        }
    }
}

void saveFifo(int id, int *array, int arraySize)
{
    char *name = malloc(sizeof(char) * 30);
    sprintf(name, FIFO_NAME, id);
    FILE *fp = fopen(name, "w");
```

```
    for (int i = 0; i < arraySize; i++)
    {
        fprintf(fp, "%d ", array[i]);
    }
    fflush(fp);
    fclose(fp);
    free(name);
}

int retreiveFifo(int id, int *array, int arraySize)
{
    char *name = malloc(sizeof(char) * 30);
    sprintf(name, FIFO_NAME, id);
    FILE *fp = fopen(name, "r");
    int element;
    int i;
    for (i = 0; i < arraySize && (fscanf(fp, "%d ", &element) != EOF); i++)
    {
        array[i] = element;
    }
    fclose(fp);
    free(name);
    return i;
}

int tree_rec(int id, int *workersId, int workersSize)
{
    // Получаем текущий pid
    int currentPid = getpid();
    printf("Поддерево %d: начало свою работу.\n", currentPid);
    pid_t left = -1, right = -1;
    // Если необходимо, создаём левое поддерево
    if (id * 2 + 1 < workersSize)
    {
        left = fork();
        if (left == 0)
            tree_rec(id * 2 + 1, workersId, workersSize);

        printf("Поддерево %d: инициализация левого поддерева с pid = %d.\n", currentPid, left);
    }

    // Если необходимо, создаём правое поддерево
    if (id * 2 + 2 < workersSize)
    {
        right = fork();
        if (right == 0)
            tree_rec(id * 2 + 2, workersId, workersSize);

        printf("Поддерево %d: инициализация правого поддерева с pid = %d.\n", currentPid, right);
    }

    for (int i = 0; i < workersSize; i++)
    {
        // Создаём буффер для элементов массива
```

```
        int *buffer = malloc(sizeof(int *) * 1000);
        if (id == workersId[i])
        {
            // Читаем числа из массива
            printf("Поддерево %d: обнаружено задание с id = %d.\n", currentPid, i + 1);
            int bufSize = retreiveFifo(i + 1, buffer, 1000);

            // Выполняем сортировку
            insertion_sort(buffer, bufSize);
            // Сохраняем элементы в файл
            saveFifo(i + 1, buffer, bufSize);
        }
        free(buffer);
    }

    // Ожидаем, когда свою работу закончат поддеревья. Если происходит ошибка, выходим с ошибкой
    printf("Поддерево %d: завершило свои задачи и ожидает дочерние поддеревья.\n", currentPid);
    int status;
    if (left != -1)
    {
        printf("Поддерево %d: ожидаем окончания поддерева с pid = %d.\n", currentPid, left);
        waitpid(left, &status, 0);
        if (status)
        {
            printf("Поддерево %d: поддерево %d завешилось с ошибкой\n", currentPid, left);
            exit(status);
        }
    }

    if (right != -1)
    {
        printf("Поддерево %d: ожидаем окончания поддерева с pid = %d.\n", currentPid, right);
        waitpid(right, &status, 0);
        if (status)
        {
            printf("Поддерево %d: поддерево %d завешилось с ошибкой\n", currentPid, left);
            exit(status);
        }
    }

    exit(0);
}

int tree()
{
    // Создаём корень
    pid_t root = fork();
    if (root == 0)
    {
        // Прочитываем файл, который указывает, какому
        // элементу дерева какой массив сортировать
        int *workers = malloc(sizeof(int *) * 1000);
        int workersSize = retreiveFifo(0, workers, 1000);
```

```
        // Запускаем рекурентную сортировку
        tree_rec(0, workers, workersSize);

        // Освобождение ресурсов
        free(workers);
        exit(0);
    }

    int status;
    waitpid(root, &status, 0);
    if (status)
    {
        exit(status);
    }
}

int main()
{
    // Наша задача - отсортировать массив чисел. Инициалазируем его
    // и заполняем случайными числами под количетсво поддеревьев и листьев
    int arraysAmount = ((1 << MAX_LEVEL) - 1);
    int **sortArray = malloc(sizeof(int *) * arraysAmount);
    for (int i = 0; i < arraysAmount; i++)
    {
        sortArray[i] = malloc(sizeof(int) * ARRAY_SIZE);
        for (int j = 0; j < ARRAY_SIZE; j++)
        {
            sortArray[i][j] = rand() % 1000;
        }
    }

    // Задаём соответствие, какой id поддерева/листа какой массив сортирует
    int *workersId = malloc(sizeof(int) * arraysAmount);
    for (int i = 0; i < arraysAmount; i++)
    {
        workersId[i] = i;
    }

    // Запишем, какому дереву соответствует какой массив в файл
    saveFifo(0, workersId, arraysAmount);
    // Запишем в файлы массивы для сортировки
    for (int i = 0; i < arraysAmount; i++)
    {
        saveFifo(i + 1, sortArray[i], ARRAY_SIZE);
    }

    printf("********************\n");
    printf("Пейлоад задач:\n");
    for (int i = 0; i < arraysAmount; i++) {
        printf("%d ", workersId[i]);
    }
    printf("\nМассивы:\n");
    for (int i = 0; i < arraysAmount; i++) {
        printf("%d: ", i);
```

```
        for (int j = 0; j < ARRAY_SIZE; j++) {
            printf("%d ", sortArray[i][j]);
        }
        printf("\n");
    }
    printf("********************\n");


    printf("Выполняется сортировка...\n");
    // Запускаем сортировку
    tree(sortArray, workersId);
    printf("Сортировка выполнена\n");
    printf("********************\n");
    printf("Массивы:\n");
    for (int i = 0; i < arraysAmount; i++) {
        printf("%d: ", i);
        int bufSize = retreiveFifo(i + 1, sortArray[i], ARRAY_SIZE);
        for (int j = 0; j < ARRAY_SIZE; j++) {
            printf("%d ", sortArray[i][j]);
        }
        printf("\n");
    }
    printf("********************\n");
    return 0;
}
```

modified_for_loop.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

#define MAX_LEVEL 3

int main() {
    for (int current_level = 0; current_level < MAX_LEVEL - 1; current_level++) {
        /*
        Выводим текущую глубину дерева, PID и PPID.
        */
        printf("Мы находимся на глубине %d\nТекущий pid = %d и ppid = %d\n", current_level, getpid(), getppid());

        /*
        Если это дочерний процесс, переходим к следующему шагу цикла, увеличивая глубину
        */
        pid_t left = fork();
        if (left == 0) continue;

        /*
        Аналогичная проверка для правого поддерева
        */
        pid_t right = fork();
```

```
        if (right == 0) continue;

        /*
        Ожидаем окончание листов/корней
        */
        int status = 0;

        waitpid(left, &status, 0);
        if (status) {
            printf("Лист/корень завершился с ошибкой!\n");
            exit(status);
        }

        waitpid(right, &status, 0);
        if (status) {
            printf("Лист/корень завершился с ошибкой!\n");
            exit(status);
        }

        exit(0);
    }

    printf("Мы находимся на глубине %d\nТекущий pid = %d и ppid = %d\n", MAX_LEVEL - 1, getpid(), getppid());
    sleep(30);
    exit(0);

    return 0;
}
```

modified_graph.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>

#define MAX_LEVEL 3
#define ARRAY_SIZE 20
#define DELAY 400
#define FIFO_NAME "./os_lab_1_2_%d"

// Сортировка вставками, искусственная нагруженность
// создается при помощи usleep
int insertion_sort(int *array, int size)
{
    for (int i = 0; i < size; i++)
    {
        int j = i;
        while (j >= 1 && array[j] < array[j - 1])
```

```
        {
            int tmp = array[j];
            array[j] = array[j - 1];
            array[j - 1] = tmp;
            usleep(DELAY * 1000);
            j--;
        }
    }
}

void saveFifo(int id, int *array, int arraySize)
{
    char *name = malloc(sizeof(char) * 30);
    sprintf(name, FIFO_NAME, id);
    FILE *fp = fopen(name, "w");
    for (int i = 0; i < arraySize; i++)
    {
        fprintf(fp, "%d ", array[i]);
    }
    fflush(fp);
    fclose(fp);
    free(name);
}

int retreiveFifo(int id, int *array, int arraySize)
{
    char *name = malloc(sizeof(char) * 30);
    sprintf(name, FIFO_NAME, id);
    FILE *fp = fopen(name, "r");
    int element;
    int i;
    for (i = 0; i < arraySize && (fscanf(fp, "%d ", &element) != EOF); i++)
    {
        array[i] = element;
    }
    fclose(fp);
    free(name);
    return i;
}

int tree_rec(int id, int *workersId, int workersSize)
{
    // Получаем текущий pid
    int currentPid = getpid();
    printf("Поддерево %d: начало свою работу.\n", currentPid);
    pid_t left = -1, right = -1;
    // Если необходимо, создаём левое поддерево
    if (id * 2 + 1 < workersSize)
    {
        left = fork();
        if (left == 0)
            tree_rec(id * 2 + 1, workersId, workersSize);

        printf("Поддерево %d: инициализация левого поддерева с pid = %d.\n", currentPid, left);
```

```
}

// Если необходимо, создаём правое поддерево
if (id * 2 + 2 < workersSize)
{
    right = fork();
    if (right == 0)
        tree_rec(id * 2 + 2, workersId, workersSize);

    printf("Поддерево %d: инициализация правого поддерева с pid = %d.\n", currentPid, right);
}

for (int i = 0; i < workersSize; i++)
{
    // Создаём буффер для элементов массива
    int *buffer = malloc(sizeof(int *) * 1000);
    if (id == workersId[i])
    {
        // Читаем числа из массива
        printf("Поддерево %d: обнаружено задание с id = %d.\n", currentPid, i + 1);
        int bufSize = retreiveFifo(i + 1, buffer, 1000);

        // Выполняем сортировку
        insertion_sort(buffer, bufSize);
        // Сохраняем элементы в файл
        saveFifo(i + 1, buffer, bufSize);
    }
    free(buffer);
}

// Ожидаем, когда свою работу закончат поддеревья. Если происходит ошибка, выходим с ошибкой
printf("Поддерево %d: завершило свои задачи и ожидает дочерние поддеревья.\n", currentPid);
int status;
if (left != -1)
{
    printf("Поддерево %d: ожидаем окончания поддерева с pid = %d.\n", currentPid, left);
    waitpid(left, &status, 0);
    if (status)
    {
        printf("Поддерево %d: поддерево %d завешилось с ошибкой\n", currentPid, left);
        exit(status);
    }
}

if (right != -1)
{
    printf("Поддерево %d: ожидаем окончания поддерева с pid = %d.\n", currentPid, right);
    waitpid(right, &status, 0);
    if (status)
    {
        printf("Поддерево %d: поддерево %d завешилось с ошибкой\n", currentPid, left);
        exit(status);
    }
}
```

```
    exit(0);
}

int tree()
{
    // Создаём корень
    pid_t root = fork();
    if (root == 0)
    {
        // Прочитываем файл, который указывает, какому
        // элементу дерева какой массив сортировать
        int *workers = malloc(sizeof(int *) * 1000);
        int workersSize = retreiveFifo(0, workers, 1000);

        // Запускаем рекурентную сортировку
        tree_rec(0, workers, workersSize);

        // Освобождение ресурсов
        free(workers);
        exit(0);
    }

    int status;
    waitpid(root, &status, 0);
    if (status)
    {
        exit(status);
    }
}

int main()
{
    // Наша задача - отсортировать массив чисел. Инициалазируем его
    // и заполняем случайными числами под количетсво поддеревьев и листьев
    int arraysAmount = ((1 << MAX_LEVEL) - 1);
    int **sortArray = malloc(sizeof(int *) * arraysAmount);
    for (int i = 0; i < arraysAmount; i++)
    {
        sortArray[i] = malloc(sizeof(int) * ARRAY_SIZE);
        for (int j = 0; j < ARRAY_SIZE; j++)
        {
            sortArray[i][j] = rand() % 1000;
        }
    }

    // Задаём соответствие, какой id поддерева/листа какой массив сортирует
    int *workersId = malloc(sizeof(int) * arraysAmount);
    for (int i = 0; i < arraysAmount; i++)
    {
        workersId[i] = i;
    }
    // Делегируем задачу листа корню
    workersId[arraysAmount - 1] = 0;
```

```
    // Запишем, какому дереву соответствует какой массив в файл
    saveFifo(0, workersId, arraysAmount);
    // Запишем в файлы массивы для сортировки
    for (int i = 0; i < arraysAmount; i++)
    {
        saveFifo(i + 1, sortArray[i], ARRAY_SIZE);
    }

    printf("********************\n");
    printf("Пейлоад задач:\n");
    for (int i = 0; i < arraysAmount; i++) {
        printf("%d ", workersId[i]);
    }
    printf("\nМассивы:\n");
    for (int i = 0; i < arraysAmount; i++) {
        printf("%d: ", i);
        for (int j = 0; j < ARRAY_SIZE; j++) {
            printf("%d ", sortArray[i][j]);
        }
        printf("\n");
    }
    printf("********************\n");


    printf("Выполняется сортировка...\n");
    // Запускаем сортировку
    tree(sortArray, workersId);
    printf("Сортировка выполнена\n");
    printf("********************\n");
    printf("Массивы:\n");
    for (int i = 0; i < arraysAmount; i++) {
        printf("%d: ", i);
        int bufSize = retreiveFifo(i + 1, sortArray[i], ARRAY_SIZE);
        for (int j = 0; j < ARRAY_SIZE; j++) {
            printf("%d ", sortArray[i][j]);
        }
        printf("\n");
    }
    printf("********************\n");
    return 0;
}
```

Протоколы, логи, скриншоты, графики:

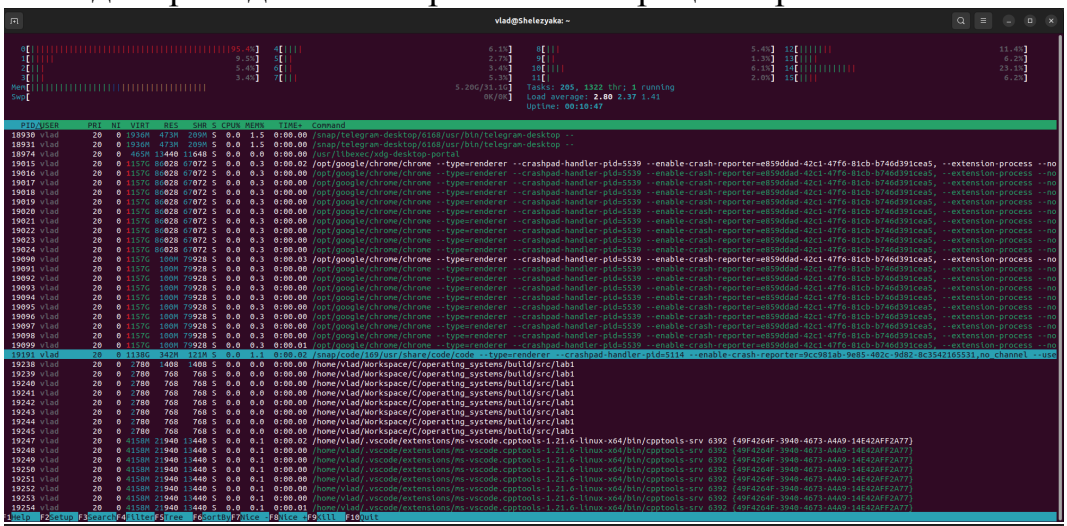
Первая программа:

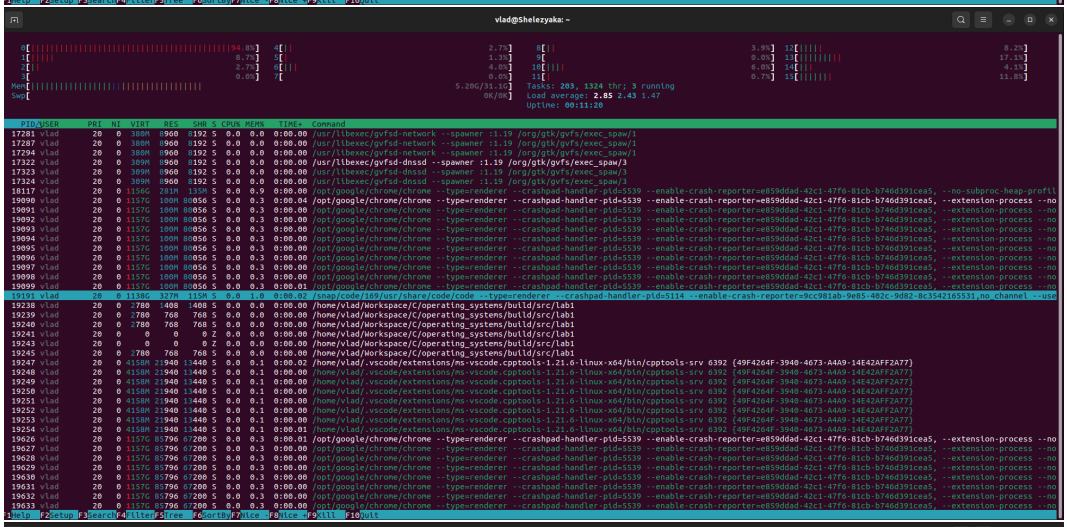
Вывод программы:

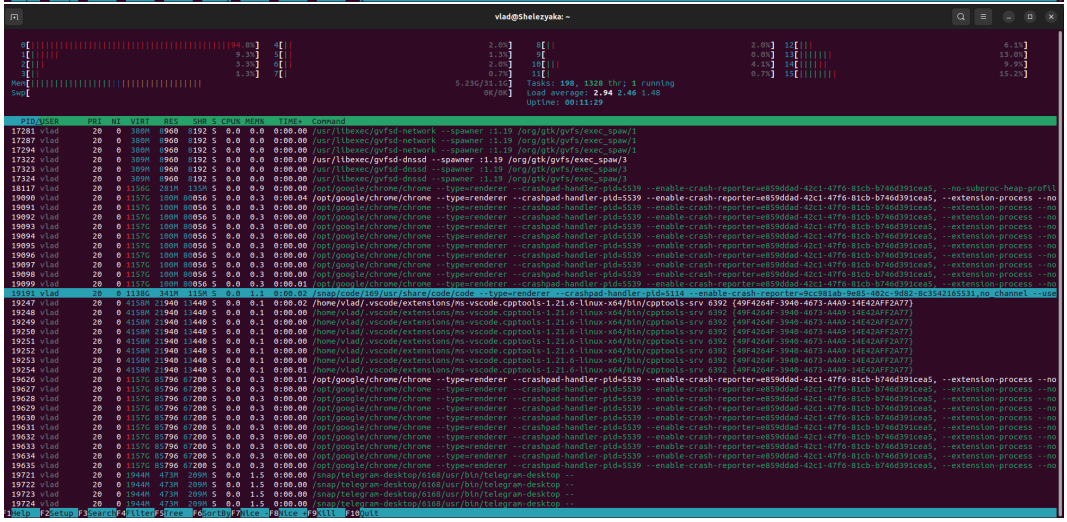
```
vlad@Shelezyaka:~/Workspace/C/operating_systems/build/src$ /home/vlad/Workspace/C/operating_systems/build/src/lab1
********************
Пейлоад задач:
0 1 2 3 4 5 6
```

```
Массивы:
0: 383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426 172 736
1: 211 368 567 429 782 530 862 123 67 135 929 802 22 58 69 167 393 456 11 42
2: 229 373 421 919 784 537 198 324 315 370 413 526 91 980 956 873 862 170 996 281
3: 305 925 84 327 336 505 846 729 313 857 124 895 582 545 814 367 434 364 43 750
4: 87 808 276 178 788 584 403 651 754 399 932 60 676 368 739 12 226 586 94 539
5: 795 570 434 378 467 601 97 902 317 492 652 756 301 280 286 441 865 689 444 619
6: 440 729 31 117 97 771 481 675 709 927 567 856 497 353 586 965 306 683 219 624
********************
Выполняется сортировка...
Поддерево 19239: поддерево с id = 0 начало свою работу.
Поддерево 19239: инициализация левого поддерева с pid = 19240.
Поддерево 19239: инициализация правого поддерева с pid = 19241.
Поддерево 19239: обнаружено задание с id = 1.
Поддерево 19240: поддерево с id = 1 начало свою работу.
Поддерево 19241: поддерево с id = 2 начало свою работу.
Поддерево 19241: инициализация левого поддерева с pid = 19242.
Поддерево 19240: инициализация левого поддерева с pid = 19243.
Поддерево 19241: инициализация правого поддерева с pid = 19244.
Поддерево 19243: поддерево с id = 3 начало свою работу.
Поддерево 19241: обнаружено задание с id = 3.
Поддерево 19242: поддерево с id = 5 начало свою работу.
Поддерево 19243: обнаружено задание с id = 4.
Поддерево 19242: обнаружено задание с id = 6.
Поддерево 19240: инициализация правого поддерева с pid = 19245.
Поддерево 19244: поддерево с id = 6 начало свою работу.
Поддерево 19240: обнаружено задание с id = 2.
Поддерево 19245: поддерево с id = 4 начало свою работу.
Поддерево 19244: обнаружено задание с id = 7.
Поддерево 19245: обнаружено задание с id = 5.
Поддерево 19241: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 19241: ожидаем окончания поддерева с pid = 19242.
Поддерево 19244: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 19243: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 19242: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 19241: ожидаем окончания поддерева с pid = 19244.
Поддерево 19245: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 19239: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 19239: ожидаем окончания поддерева с pid = 19240.
Поддерево 19240: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 19240: ожидаем окончания поддерева с pid = 19243.
Поддерево 19240: ожидаем окончания поддерева с pid = 19245.
Поддерево 19239: ожидаем окончания поддерева с pid = 19241.
Сортировка выполнена
********************
Массивы:
0: 27 59 172 335 362 383 386 421 426 492 540 649 690 736 763 777 793 886 915 926
1: 11 22 42 58 67 69 123 135 167 211 368 393 429 456 530 567 782 802 862 929
2: 91 170 198 229 281 315 324 370 373 413 421 526 537 784 862 873 919 956 980 996
3: 43 84 124 305 313 327 336 364 367 434 505 545 582 729 750 814 846 857 895 925
4: 12 60 87 94 178 226 276 368 399 403 539 584 586 651 676 739 754 788 808 932
5: 97 280 286 301 317 378 434 441 444 467 492 570 601 619 652 689 756 795 865 902
6: 31 97 117 219 306 353 440 481 497 567 586 624 675 683 709 729 771 856 927 965
********************
```

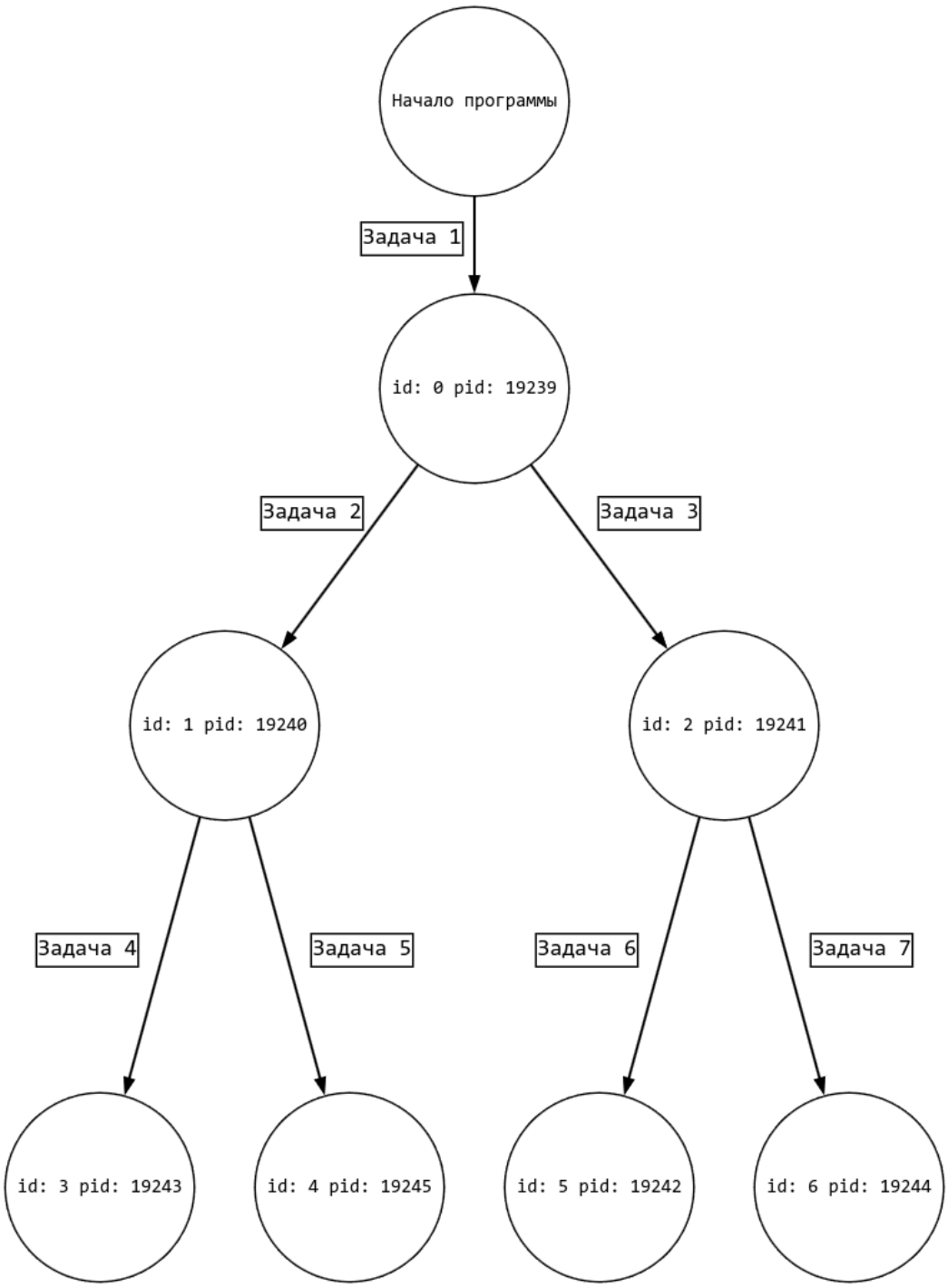
Вывод htop когда листья ”работают в процессе работы и после:







Полученное дерево:



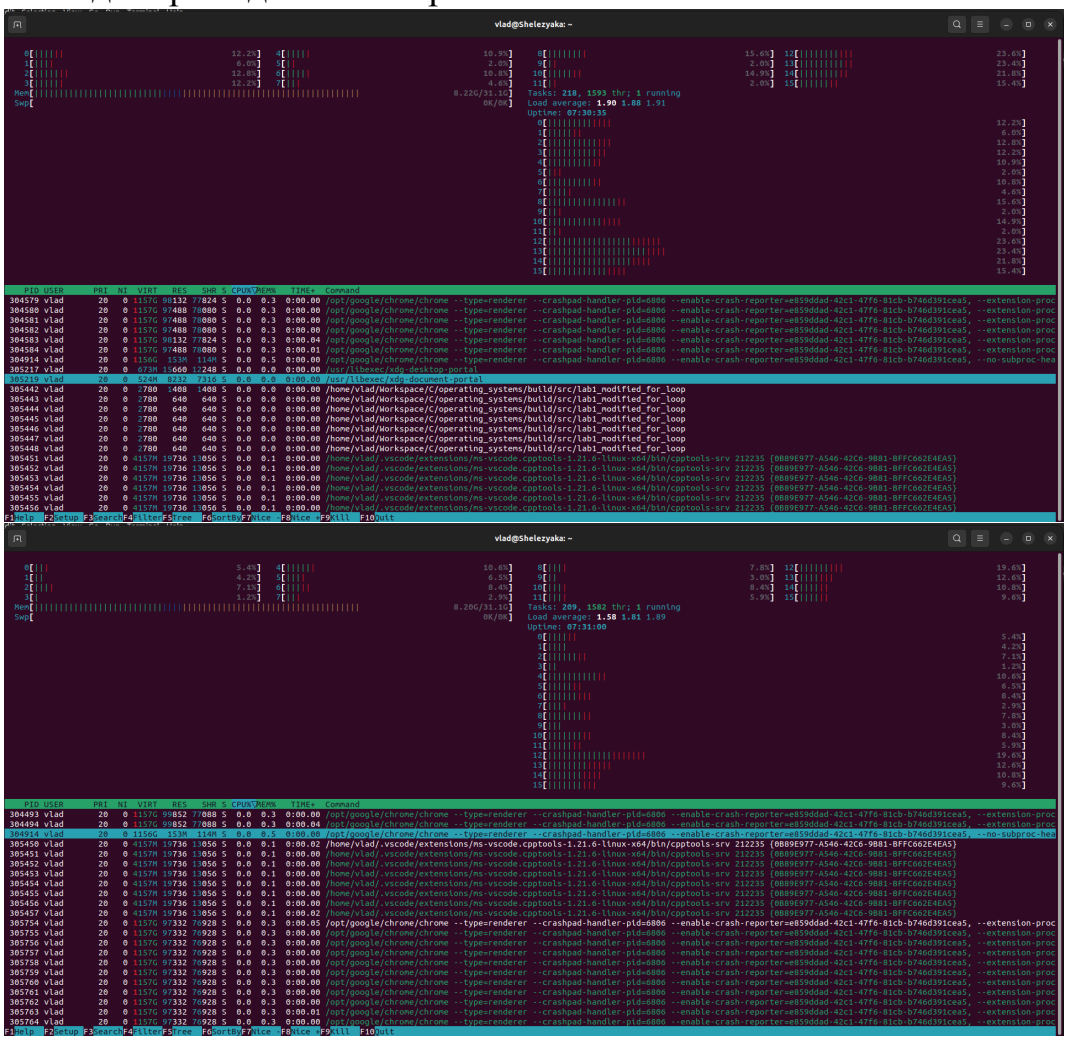
Вторая программа:

Вывод программы:

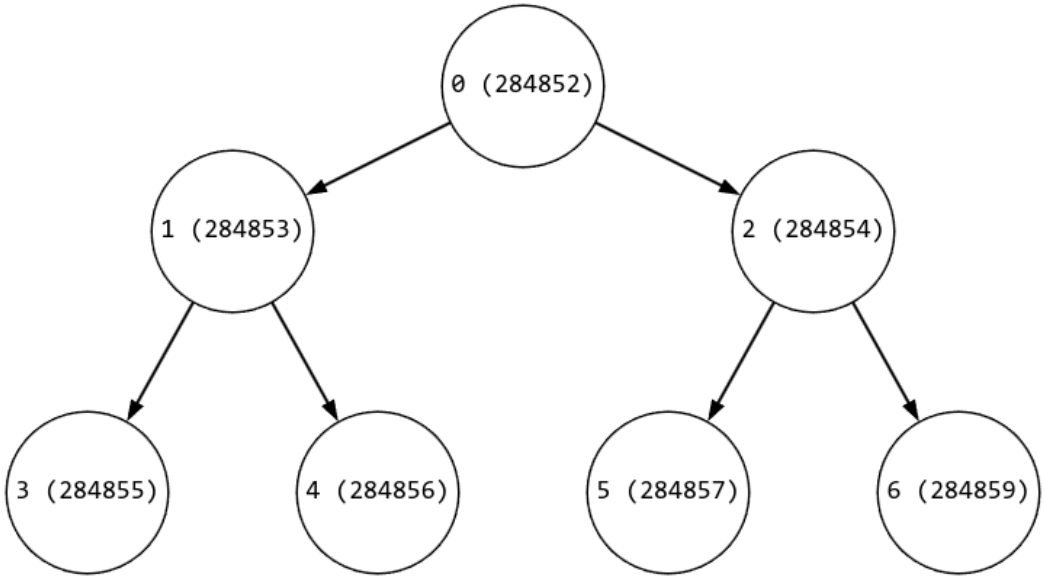
```
Мы находимся на глубине 0
Текущий pid = 284852 и ppid = 284565
Мы находимся на глубине 1
Текущий pid = 284853 и ppid = 284852
Мы находимся на глубине 1
Текущий pid = 284854 и ppid = 284852
Мы находимся на глубине 2
Текущий pid = 284855 и ppid = 284853
Мы находимся на глубине 2
Текущий pid = 284856 и ppid = 284853
Мы находимся на глубине 2
```

```
Текущий pid = 284857 и ppid = 284854
Мы находимся на глубине 2
Текущий pid = 284859 и ppid = 284854
```

Вывод htop когда листья ”работают”и после:



Полученное дерево:



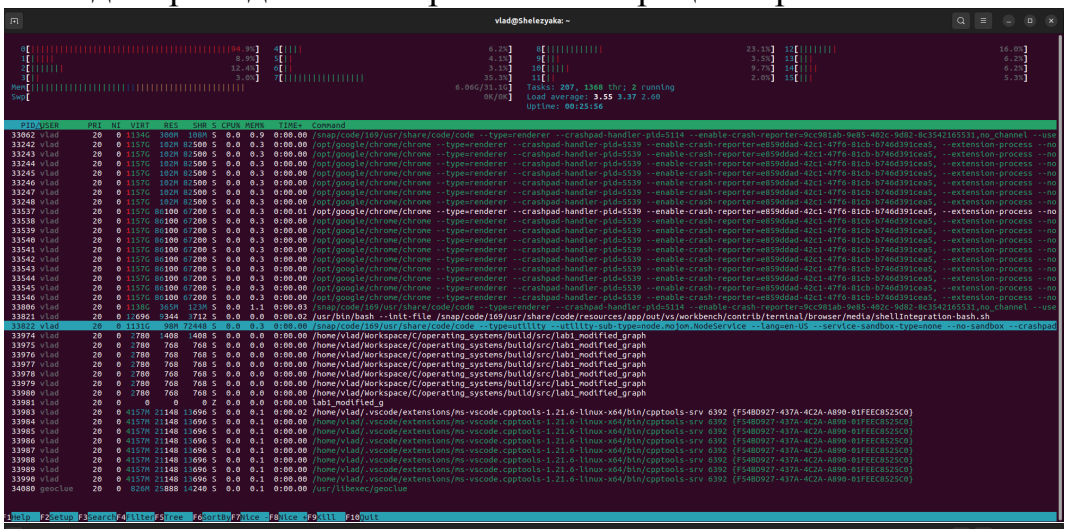
Третья программа:

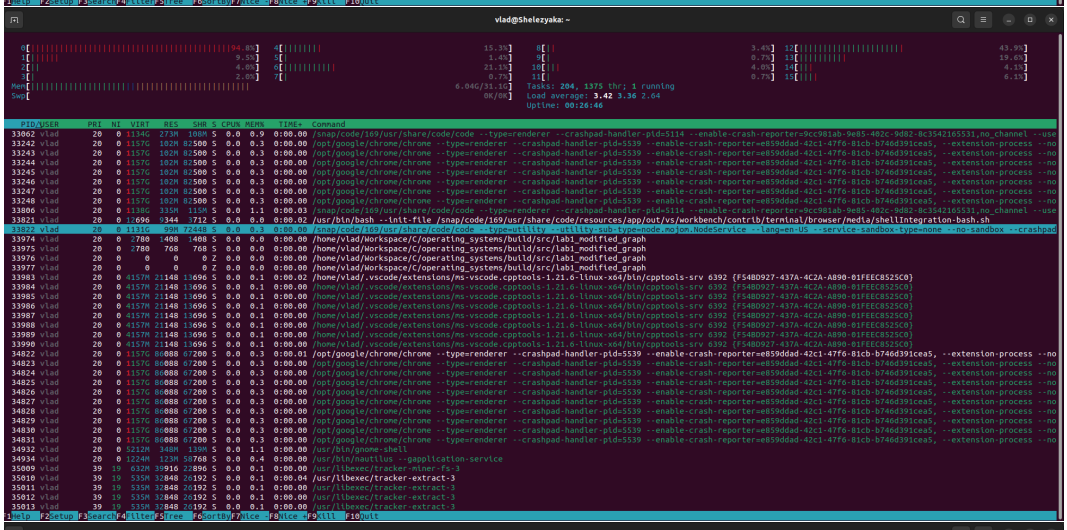
Вывод программы:

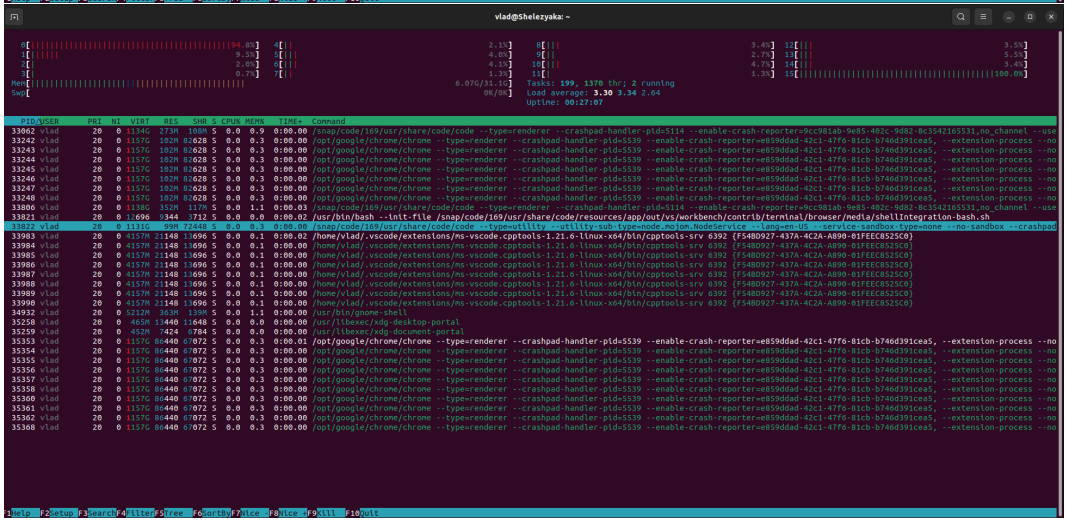
```
vlad@Shelezyaka:~/Workspace/C/operating_systems/build/src$
↪  /home/vlad/Workspace/C/operating_systems/build/src/lab1_modified_graph
********************
Пейлоад задач:
0 1 2 3 4 5 0
Массивы:
0: 383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426 172 736
1: 211 368 567 429 782 530 862 123 67 135 929 802 22 58 69 167 393 456 11 42
2: 229 373 421 919 784 537 198 324 315 370 413 526 91 980 956 873 862 170 996 281
3: 305 925 84 327 336 505 846 729 313 857 124 895 582 545 814 367 434 364 43 750
4: 87 808 276 178 788 584 403 651 754 399 932 60 676 368 739 12 226 586 94 539
5: 795 570 434 378 467 601 97 902 317 492 652 756 301 280 286 441 865 689 444 619
6: 440 729 31 117 97 771 481 675 709 927 567 856 497 353 586 965 306 683 219 624
********************
Выполняется сортировка...
Поддерево 33975: поддерево с id = 0 начало свою работу.
Поддерево 33975: инициализация левого поддерева с pid = 33976.
Поддерево 33976: поддерево с id = 1 начало свою работу.
Поддерево 33976: инициализация левого поддерева с pid = 33978.
Поддерево 33975: инициализация правого поддерева с pid = 33977.
Поддерево 33975: обнаружено задание с id = 1.
Поддерево 33977: поддерево с id = 2 начало свою работу.
Поддерево 33978: поддерево с id = 3 начало свою работу.
Поддерево 33978: обнаружено задание с id = 4.
Поддерево 33976: инициализация правого поддерева с pid = 33979.
Поддерево 33976: обнаружено задание с id = 2.
Поддерево 33979: поддерево с id = 4 начало свою работу.
Поддерево 33977: инициализация левого поддерева с pid = 33980.
Поддерево 33979: обнаружено задание с id = 5.
Поддерево 33977: инициализация правого поддерева с pid = 33981.
Поддерево 33980: поддерево с id = 5 начало свою работу.
Поддерево 33977: обнаружено задание с id = 3.
Поддерево 33980: обнаружено задание с id = 6.
Поддерево 33981: поддерево с id = 6 начало свою работу.
Поддерево 33981: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 33977: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 33977: ожидаем окончания поддерева с pid = 33980.
Поддерево 33978: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 33980: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 33977: ожидаем окончания поддерева с pid = 33981.
Поддерево 33979: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 33975: обнаружено задание с id = 7.
Поддерево 33976: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 33976: ожидаем окончания поддерева с pid = 33978.
Поддерево 33976: ожидаем окончания поддерева с pid = 33979.
Поддерево 33975: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 33975: ожидаем окончания поддерева с pid = 33976.
Поддерево 33975: ожидаем окончания поддерева с pid = 33977.
Сортировка выполнена
********************
Массивы:
0: 27 59 172 335 362 383 386 421 426 492 540 649 690 736 763 777 793 886 915 926
1: 11 22 42 58 67 69 123 135 167 211 368 393 429 456 530 567 782 802 862 929
2: 91 170 198 229 281 315 324 370 373 413 421 526 537 784 862 873 919 956 980 996
```

```
3: 43 84 124 305 313 327 336 364 367 434 505 545 582 729 750 814 846 857 895 925
4: 12 60 87 94 178 226 276 368 399 403 539 584 586 651 676 739 754 788 808 932
5: 97 280 286 301 317 378 434 441 444 467 492 570 601 619 652 689 756 795 865 902
6: 31 97 117 219 306 353 440 481 497 567 586 624 675 683 709 729 771 856 927 965
********************
```

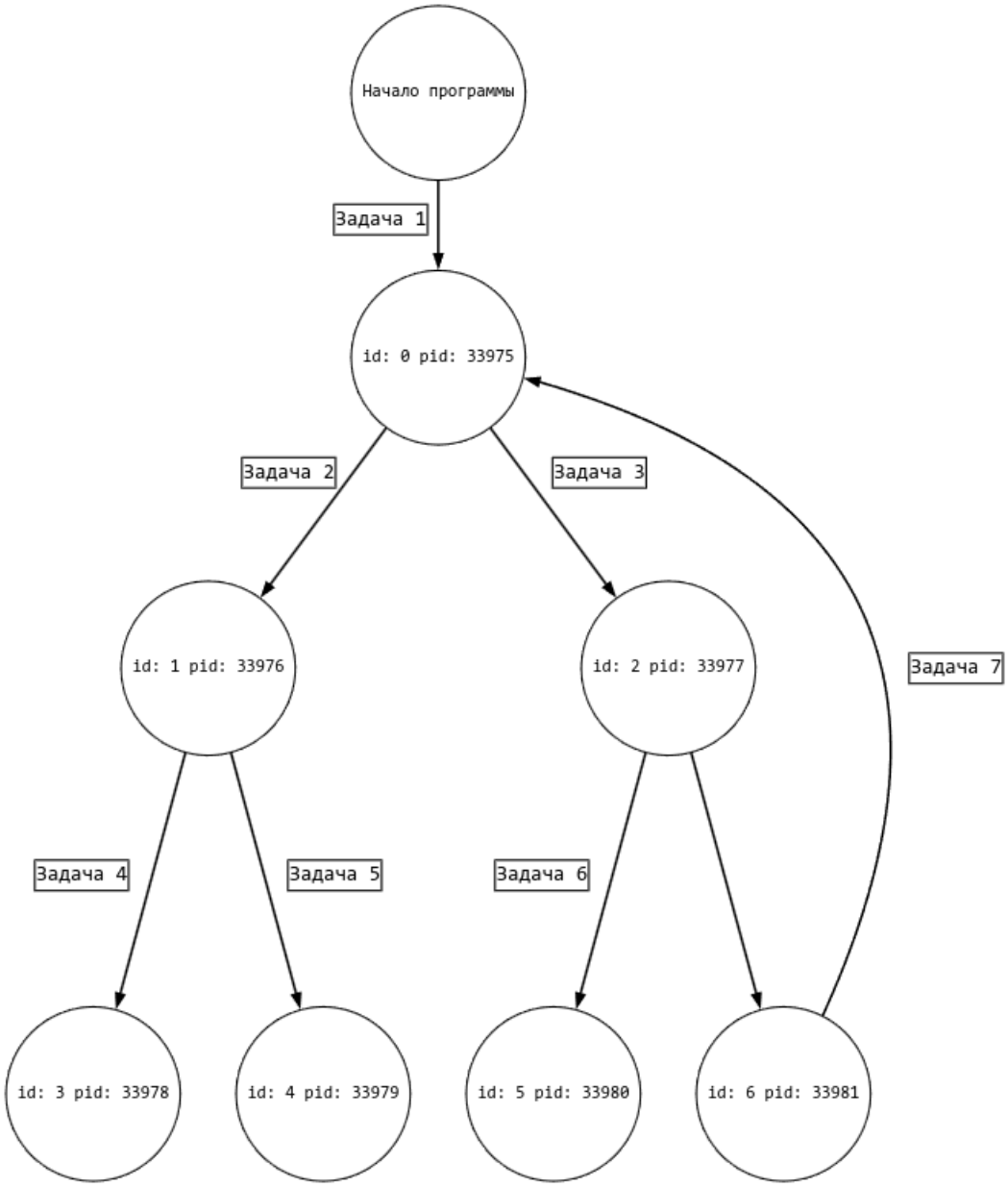
Вывод htop когда листья ”работают в процессе работы и после:







Полученное дерево:



Вывод: в ходе лабораторной работы изучили основы работы с системными вызовами и процессами в операционной системе Linux (Ubuntu). Научились создавать отдельный процесс, отслеживать статус выполнения дочернего процесса и обрабатывать коды после окончания работы дочернего процесса. Научились получать текущий PID и родительский PID. Наиболее интересным вариантом реализации оказалось задание по генерации дерева с применением for-loop. Цикл работает потому что при создании дочернего процесса в него копируются все родительские переменные, следовательно мы можем получить доступ к глубине дерева в текущем поддереве/листе. Первая версия программы создавала дерево глубиной на 1 лист больше. Также в процессе исследования fork были допущены ошибки, например так создавалось раньше левое и правое поддерево: `pid_t left = fork(), right = fork();` что приводило к ошибкам и поддеревьям с двумя или тремя вершинами.