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>
#include <sys/ptrace.h>
#include <sys/wait.h>
#include <sys/user.h> // Для структуры user_regs_struct

#define MAX_LEVEL (3)
#define ARRAYS_AMOUNT ((1 << MAX_LEVEL) - 1)
#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;
}

void sortFifo(int id)
{
    pid_t currentPid = getpid();
    printf("Поддерево %d: выполняется сортировка задачи id = %d.\n", currentPid, id);
    // Создаём буффер для элементов массива
    int *buffer = malloc(sizeof(int *) * 1000);
    // Читаем числа из массива
    int bufSize = retreiveFifo(id, buffer, 1000);

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

typedef struct Delegation
{
    // На сколько уровней необходимо опустить делегацию вниз по дереву
    int level;
    // id задачи
    int id;
} Delegation;

void process_child(int pid, Delegation *delegations, int *delegationsSize)
```

```
{
    int currentPid = getpid();
    int status;
    struct user_regs_struct regs;
    if (pid != -1)
    {
        printf("Поддерево %d: ожидаем окончания поддерева с pid = %d.\n", currentPid, pid);
        // Ожидаем, пока поддерево не приостановит своё выполнение
        waitpid(pid, &status, 0);
        // Если мы не вышли из процесса
        while (!WIFEXITED(status))
        {
            // Считаем регистры
            if (ptrace(PTRACE_GETREGS, pid, NULL, &regs) == -1)
            {
                perror("ptrace GETREGS");
                exit(1);
            }

            // r14 = уровень, r15 = какая задача
            Delegation del = {regs.r14, regs.r15};

            printf("Поддерево %d: получена перенаправленная задачи на %d уровней вверх, id = %d.\n", currentPid,
↪   del.level, del.id);

            // Если необходимый уровень достиг, выполняем сортировку.

            if (del.level == 0)
                sortFifo(del.id);
            else
            {
                // Иначе - добавляем в массив передачи задач.
                // Мы передадим этот массив родителю.
                del.level--;
                delegations[(*delegationsSize)++] = del;
            }

            // Продолжить выполнение процесса до следующей остановки
            if (ptrace(PTRACE_CONT, pid, 0, 0) == -1)
            {
                perror("ptrace PTRACE_CONT");
                exit(1);
            }

            // Ожидаем следующей остановки дочернего процесса
            waitpid(pid, &status, 0);
        }

        if (status)
        {
            printf("Поддерево %d: поддерево %d завешилось с ошибкой\n", currentPid, pid);
            exit(status);
        }
    }
```

```
}

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

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

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

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

    // Выполняем саму задачу
    sortFifo(id);

    // Ожидаем, когда свою работу закончат поддеревья. Если происходит ошибка, выходим с ошибкой
    printf("Поддерево %d: завершило свои задачи и ожидает дочерние поддеревья.\n", currentPid);

    process_child(left, delegations, &delegationsSize);
    process_child(right, delegations, &delegationsSize);

    if (ptrace(PTRACE_TRACEME, 0, NULL, NULL) == -1)
    {
        // Объявление, что текущий процесс желает быть
        // отслеживаемым родительским процессом
        perror("PTRACE_TRACEME");
        exit(1);
    }

    // Для каждой делегации из массива делегаций
    for (int i = 0; i < delegationsSize; i++)
    {
        Delegation del = delegations[i];
        int64_t delLevel = del.level;
        int64_t delId = del.id;
        printf("Поддерево %d: выполняется перенаправление задачи на %d уровней вверх, id = %d.\n", currentPid, del.level,
↪   del.id);
```

```
        // Сохраним делегацию в регистры
        __asm__ volatile(
            "movq %0, %%r14\n"
            "movq %1, %%r15\n"
            :
            : "r"(delLevel), "r"(delId)
            : "r15", "r14");
        // Выкинем статус из процесса, означающий, что в
        // нём что-то произошло. В данном случае -
        // делегация.
        raise(SIGCHLD);
        printf("Поддерево %d: задача перенаправлена.\n", currentPid);
    }

    exit(0);
}

int tree()
{
    // Создаём корень
    pid_t root = fork();
    if (root == 0)
    {
        // Запускаем рекурентную сортировку
        tree_rec(0);
        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 **sortArray = malloc(sizeof(int *) * ARRAYS_AMOUNT);
    for (int i = 0; i < ARRAYS_AMOUNT; i++)
    {
        sortArray[i] = malloc(sizeof(int) * ARRAY_SIZE);
        for (int j = 0; j < ARRAY_SIZE; j++)
        {
            sortArray[i][j] = rand() % 1000;
        }
    }

    // Запишем в файлы массивы для сортировки
    for (int i = 0; i < ARRAYS_AMOUNT; i++)
    {
        saveFifo(i, sortArray[i], ARRAY_SIZE);
```

```
    }

    printf("********************\n");
    printf("\nМассивы:\n");
    for (int i = 0; i < ARRAYS_AMOUNT; 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();
    printf("Сортировка выполнена\n");
    printf("********************\n");
    printf("Массивы:\n");
    for (int i = 0; i < ARRAYS_AMOUNT; i++)
    {
        printf("%d: ", i);
        int bufSize = retreiveFifo(i, 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>
#include <sys/ptrace.h>
#include <sys/wait.h>
#include <sys/user.h> // Для структуры user_regs_struct

#define MAX_LEVEL (3)
```

```c
#define ARRAYS_AMOUNT ((1 << MAX_LEVEL) - 1)
#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;
}

void sortFifo(int id)
```

```
{
    pid_t currentPid = getpid();
    printf("Поддерево %d: выполняется сортировка задачи id = %d.\n", currentPid, id);
    // Создаём буффер для элементов массива
    int *buffer = malloc(sizeof(int *) * 1000);
    // Читаем числа из массива
    int bufSize = retreiveFifo(id, buffer, 1000);

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

typedef struct Delegation
{
    // На сколько уровней необходимо опустить делегацию вниз по дереву
    int level;
    // id задачи
    int id;
} Delegation;

void process_child(int pid, Delegation *delegations, int *delegationsSize)
{
    int currentPid = getpid();
    int status;
    struct user_regs_struct regs;
    if (pid != -1)
    {
        printf("Поддерево %d: ожидаем окончания поддерева с pid = %d.\n", currentPid, pid);
        // Ожидаем, пока поддерево не приостановит своё выполнение
        waitpid(pid, &status, 0);
        // Если мы не вышли из процесса
        while (!WIFEXITED(status))
        {
            // Считаем регистры
            if (ptrace(PTRACE_GETREGS, pid, NULL, &regs) == -1)
            {
                perror("ptrace GETREGS");
                exit(1);
            }

            // r14 = уровень, r15 = какая задача
            Delegation del = {regs.r14, regs.r15};

            printf("Поддерево %d: получена перенаправленная задачи на %d уровней вверх, id = %d.\n", currentPid,
↪  del.level, del.id);

            // Если необходимый уровень достиг, выполняем сортировку.

            if (del.level == 0)
                sortFifo(del.id);
            else
```

```
            {
                // Иначе - добавляем в массив передачи задач.
                // Мы передадим этот массив родителю.
                del.level--;
                delegations[(*delegationsSize)++] = del;
            }

            // Продолжить выполнение процесса до следующей остановки
            if (ptrace(PTRACE_CONT, pid, 0, 0) == -1)
            {
                perror("ptrace PTRACE_CONT");
                exit(1);
            }

            // Ожидаем следующей остановки дочернего процесса
            waitpid(pid, &status, 0);
        }

        if (status)
        {
            printf("Поддерево %d: поддерево %d завешилось с ошибкой\n", currentPid, pid);
            exit(status);
        }
    }
}

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

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

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

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

```
    if (id == 6)
    {
        // Заносим перенаправление в список
        delegationsSize++;
        delegations[0] = (Delegation){1, id};
    }
    else
    {
        // Выполняем саму задачу
        sortFifo(id);
    }

    // Ожидаем, когда свою работу закончат поддеревья. Если происходит ошибка, выходим с ошибкой
    printf("Поддерево %d: завершило свои задачи и ожидает дочерние поддеревья.\n", currentPid);

    process_child(left, delegations, &delegationsSize);
    process_child(right, delegations, &delegationsSize);

    if (ptrace(PTRACE_TRACEME, 0, NULL, NULL) == -1)
    {
        // Объявление, что текущий процесс желает быть
        // отслеживаемым родительским процессом
        perror("PTRACE_TRACEME");
        exit(1);
    }

    // Для каждой делегации из массива делегаций
    for (int i = 0; i < delegationsSize; i++)
    {
        Delegation del = delegations[i];
        int64_t delLevel = del.level;
        int64_t delId = del.id;
        printf("Поддерево %d: выполняется перенаправление задачи на %d уровней вверх, id = %d.\n", currentPid, del.level,
↪   del.id);
        // Сохраним делегацию в регистры
        __asm__ volatile(
            "movq %0, %%r14\n"
            "movq %1, %%r15\n"
            :
            : "r"(delLevel), "r"(delId)
            : "r15", "r14");
        // Выкинем сигнал из процесса, означающий, что в
        // нём что-то произошло. В данном случае -
        // делегация.
        raise(SIGCHLD);
        printf("Поддерево %d: задача перенаправлена.\n", currentPid);
    }

    exit(0);
}

int tree()
{
    // Создаём корень
```

```
    pid_t root = fork();
    if (root == 0)
    {
        // Запускаем рекурентную сортировку
        tree_rec(0);
        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 **sortArray = malloc(sizeof(int *) * ARRAYS_AMOUNT);
    for (int i = 0; i < ARRAYS_AMOUNT; i++)
    {
        sortArray[i] = malloc(sizeof(int) * ARRAY_SIZE);
        for (int j = 0; j < ARRAY_SIZE; j++)
        {
            sortArray[i][j] = rand() % 1000;
        }
    }

    // Запишем в файлы массивы для сортировки
    for (int i = 0; i < ARRAYS_AMOUNT; i++)
    {
        saveFifo(i, sortArray[i], ARRAY_SIZE);
    }

    printf("********************\n");
    printf("\nМассивы:\n");
    for (int i = 0; i < ARRAYS_AMOUNT; 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();
    printf("Сортировка выполнена\n");
    printf("********************\n");
```

```
    printf("Массивы:\n");
    for (int i = 0; i < ARRAYS_AMOUNT; i++)
    {
        printf("%d: ", i);
        int bufSize = retreiveFifo(i, 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:

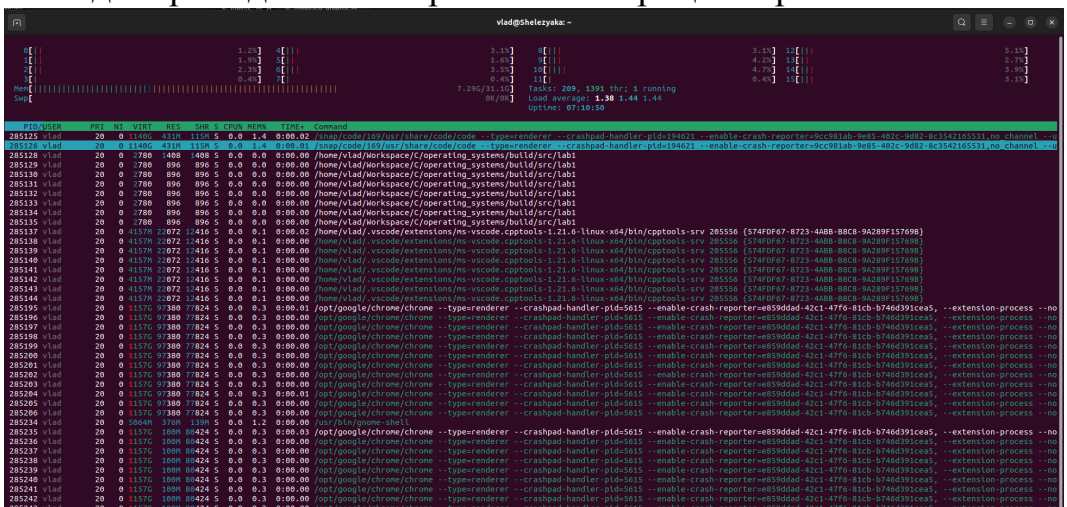
Первая программа:

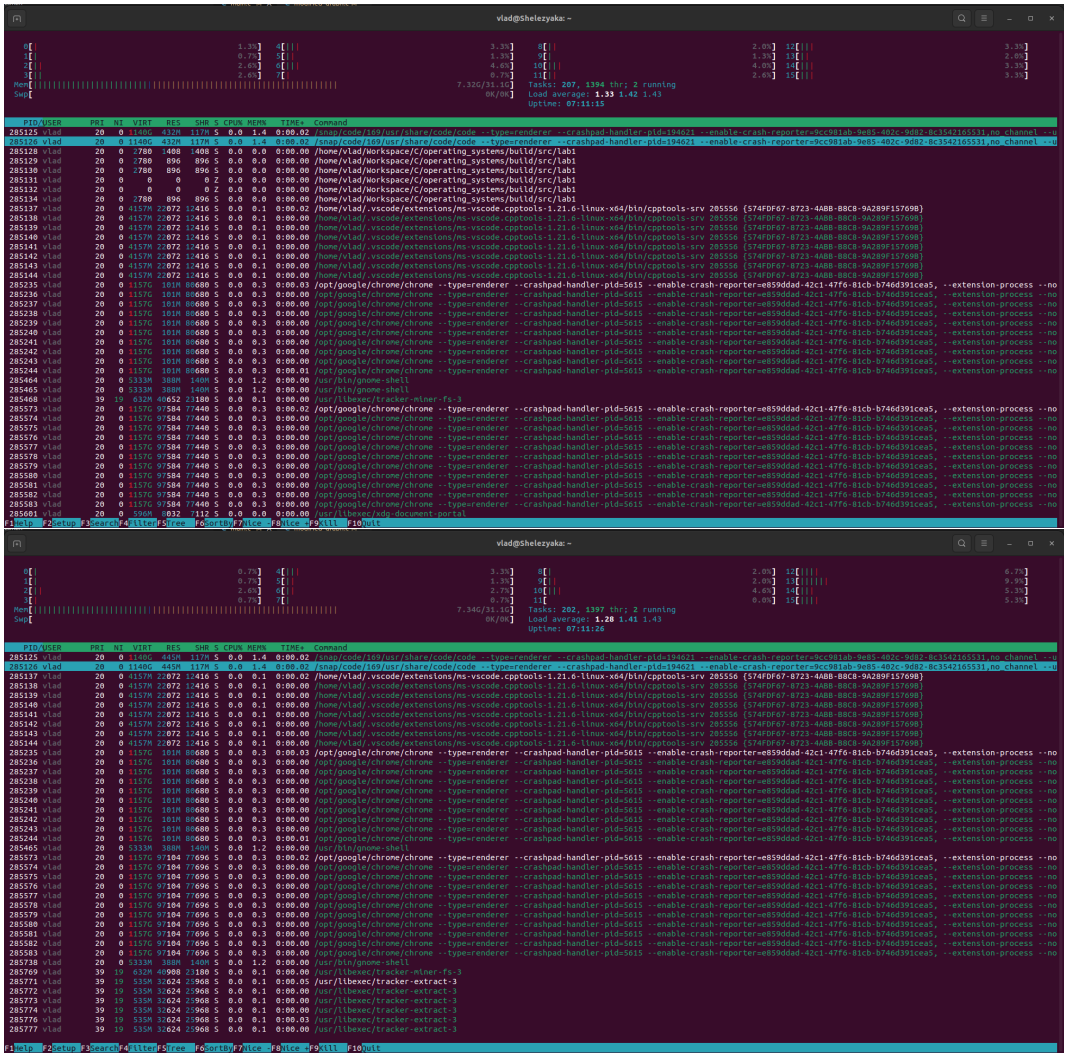
Вывод программы:

```
vlad@Shelezyaka:~/Workspace/C/operating_systems/build/src$ /home/vlad/Workspace/C/operating_systems/build/src/lab1
********************

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 285129: поддерево с id = 0 начало свою работу.
Поддерево 285129: инициализация левого поддерева с pid = 285130.
Поддерево 285130: поддерево с id = 1 начало свою работу.
Поддерево 285129: инициализация правого поддерева с pid = 285131.
Поддерево 285129: выполняется сортировка задачи id = 0.
Поддерево 285130: инициализация левого поддерева с pid = 285132.
Поддерево 285131: поддерево с id = 2 начало свою работу.
Поддерево 285132: поддерево с id = 3 начало свою работу.
Поддерево 285132: выполняется сортировка задачи id = 3.
Поддерево 285131: инициализация левого поддерева с pid = 285133.
Поддерево 285130: инициализация правого поддерева с pid = 285134.
Поддерево 285130: выполняется сортировка задачи id = 1.
Поддерево 285133: поддерево с id = 5 начало свою работу.
Поддерево 285133: выполняется сортировка задачи id = 5.
Поддерево 285134: поддерево с id = 4 начало свою работу.
Поддерево 285131: инициализация правого поддерева с pid = 285135.
Поддерево 285131: выполняется сортировка задачи id = 2.
Поддерево 285134: выполняется сортировка задачи id = 4.
Поддерево 285135: поддерево с id = 6 начало свою работу.
```

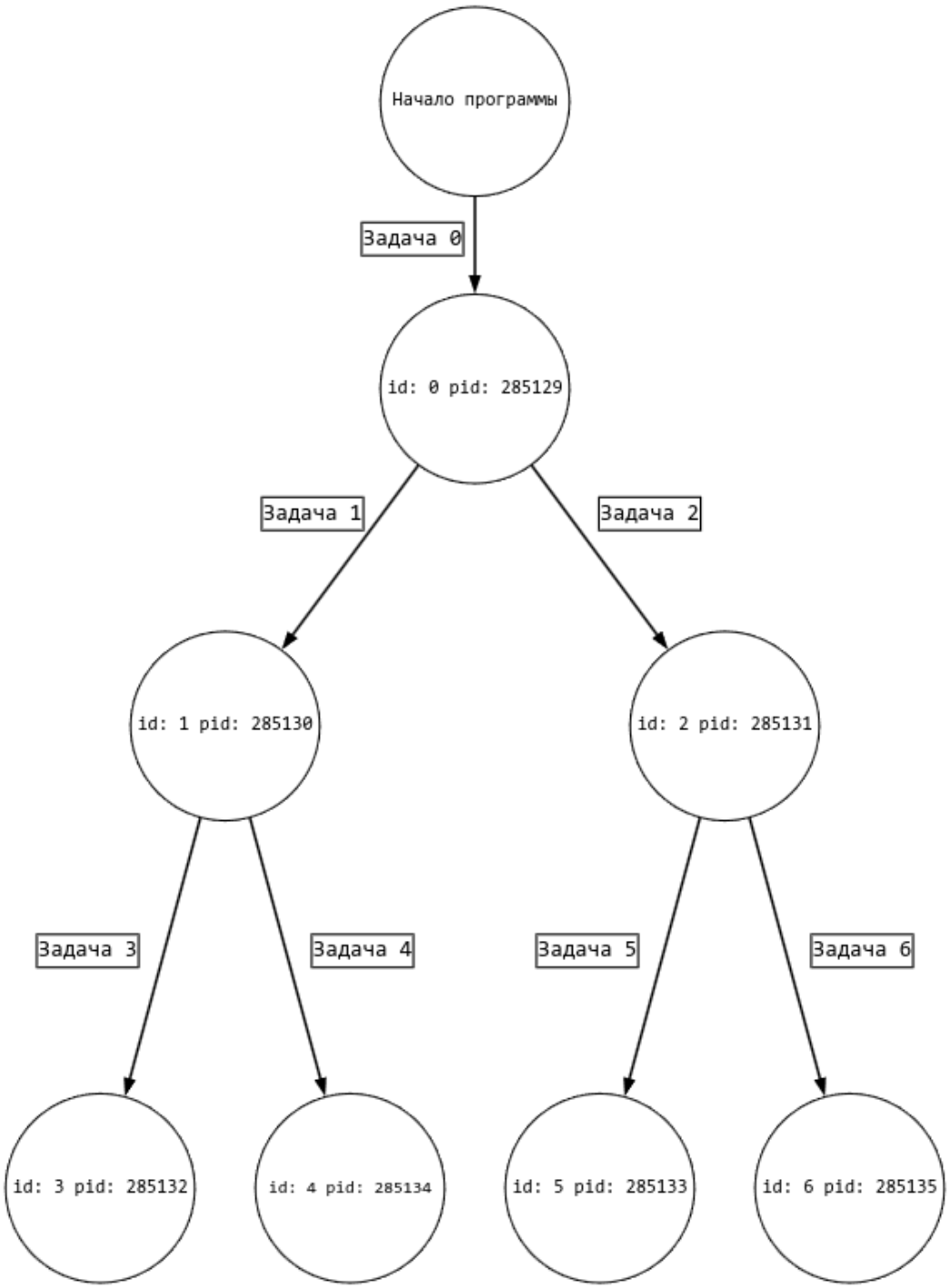
```
Поддерево 285135: выполняется сортировка задачи id = 6.
Поддерево 285131: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 285131: ожидаем окончания поддерева с pid = 285133.
Поддерево 285135: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 285132: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 285133: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 285131: ожидаем окончания поддерева с pid = 285135.
Поддерево 285134: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 285129: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 285129: ожидаем окончания поддерева с pid = 285130.
Поддерево 285130: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 285130: ожидаем окончания поддерева с pid = 285132.
Поддерево 285130: ожидаем окончания поддерева с pid = 285134.
Поддерево 285129: ожидаем окончания поддерева с pid = 285131.
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:



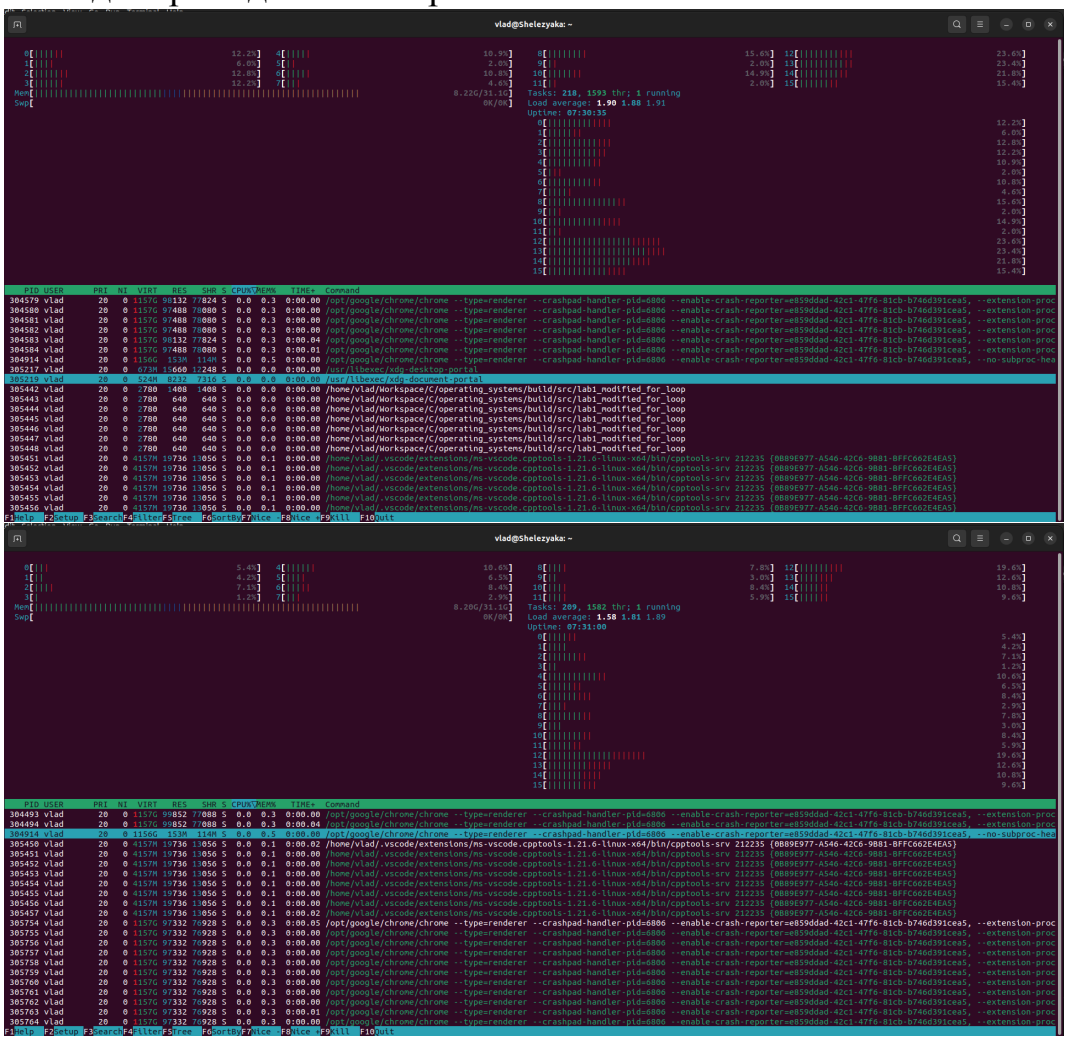
Вторая программа:

Вывод программы:

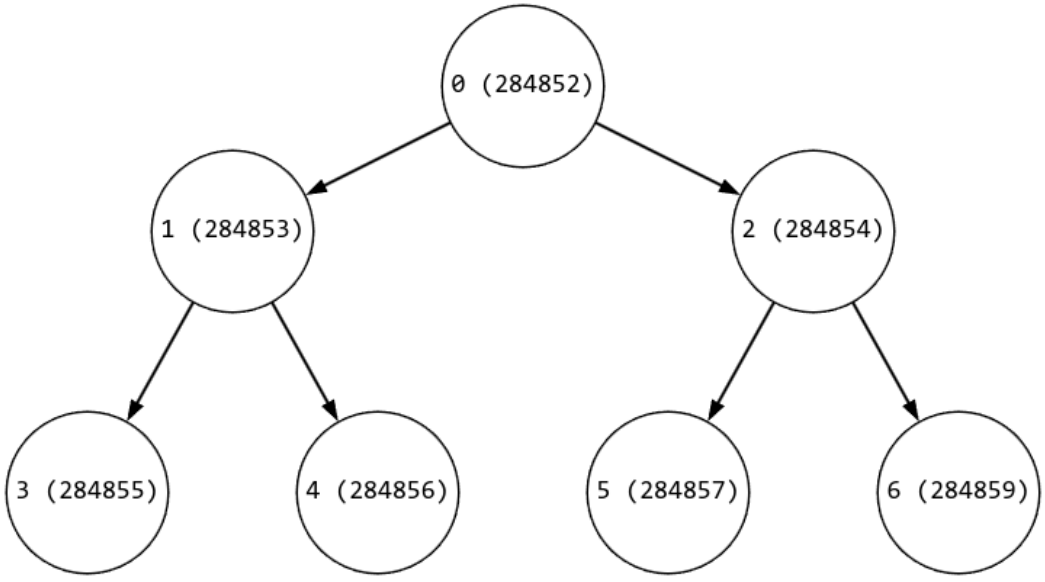
```
Мы находимся на глубине 0
Текущий pid = 284852 и ppid = 284565
Мы находимся на глубине 1
Текущий pid = 284853 и ppid = 284852
Мы находимся на глубине 1
Текущий pid = 284854 и ppid = 284852
Мы находимся на глубине 2
Текущий pid = 284855 и ppid = 284853
Мы находимся на глубине 2
Текущий pid = 284856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



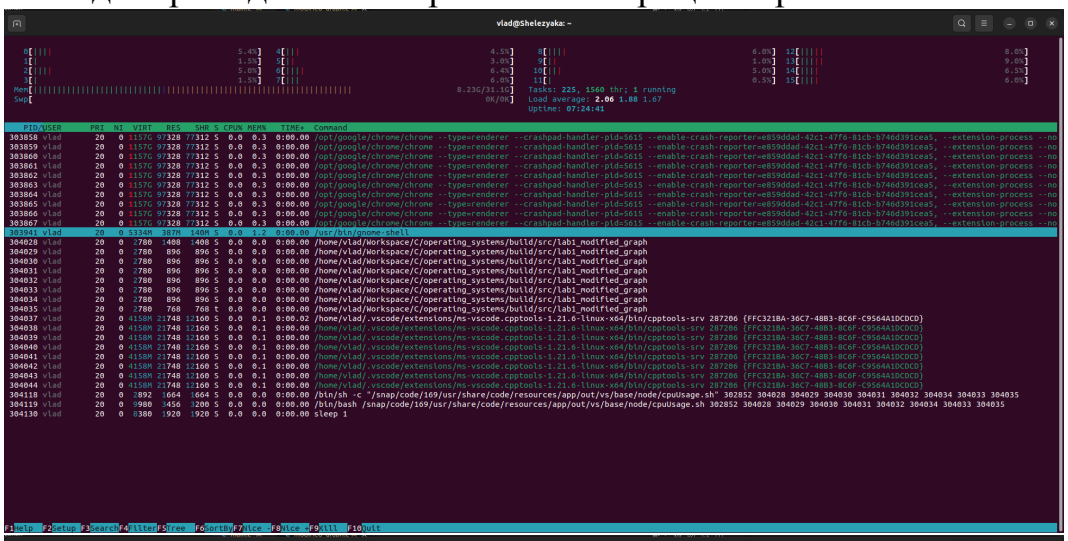
Третья программа:

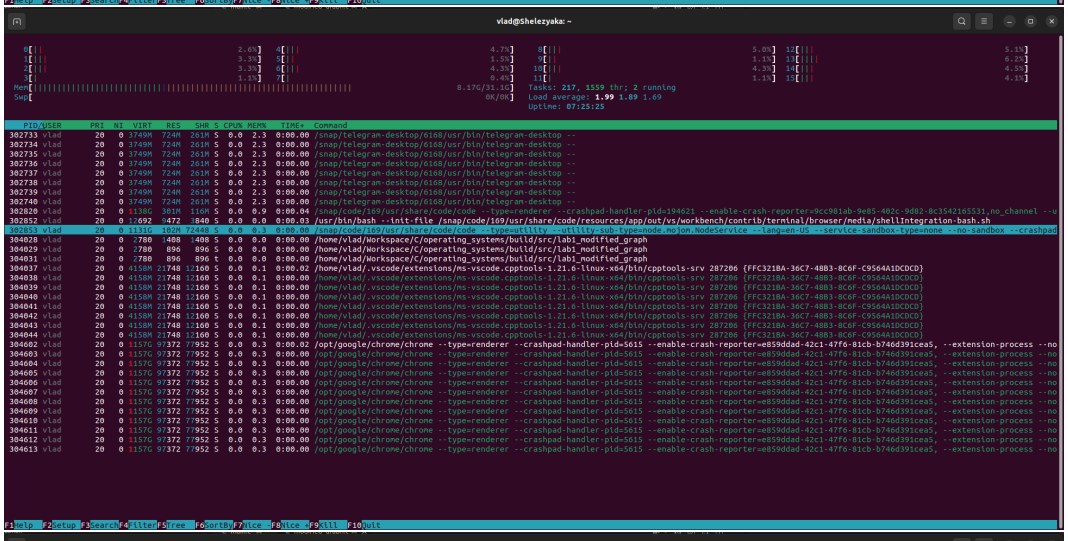
Вывод программы:

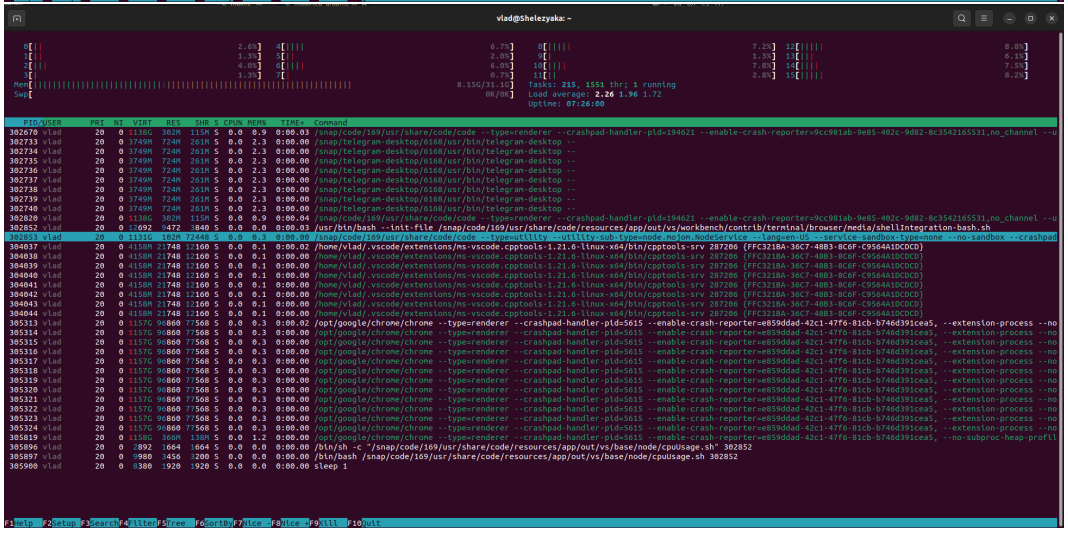
```
vlad@Shelezyaka:~/Workspace/C/operating_systems/build/src$
↪  /home/vlad/Workspace/C/operating_systems/build/src/lab1_modified_graph
********************

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 304029: поддерево с id = 0 начало свою работу.
Поддерево 304029: инициализация левого поддерева с pid = 304030.
Поддерево 304030: поддерево с id = 1 начало свою работу.
Поддерево 304029: инициализация правого поддерева с pid = 304031.
Поддерево 304029: выполняется сортировка задачи id = 0.
Поддерево 304031: поддерево с id = 2 начало свою работу.
Поддерево 304030: инициализация левого поддерева с pid = 304032.
Поддерево 304032: поддерево с id = 3 начало свою работу.
Поддерево 304032: выполняется сортировка задачи id = 3.
Поддерево 304031: инициализация левого поддерева с pid = 304033.
Поддерево 304030: инициализация правого поддерева с pid = 304034.
Поддерево 304030: выполняется сортировка задачи id = 1.
Поддерево 304033: поддерево с id = 5 начало свою работу.
Поддерево 304033: выполняется сортировка задачи id = 5.
Поддерево 304034: поддерево с id = 4 начало свою работу.
Поддерево 304034: выполняется сортировка задачи id = 4.
Поддерево 304031: инициализация правого поддерева с pid = 304035.
Поддерево 304031: выполняется сортировка задачи id = 2.
Поддерево 304035: поддерево с id = 6 начало свою работу.
Поддерево 304035: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 304035: выполняется перенаправление задачи на 1 уровней вверх, id = 6.
Поддерево 304031: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 304031: ожидаем окончания поддерева с pid = 304033.
Поддерево 304032: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 304033: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 304031: ожидаем окончания поддерева с pid = 304035.
Поддерево 304031: получена перенаправленная задачи на 1 уровней вверх, id = 6.
Поддерево 304035: задача перенаправлена.
Поддерево 304031: выполняется перенаправление задачи на 0 уровней вверх, id = 6.
Поддерево 304034: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 304029: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 304029: ожидаем окончания поддерева с pid = 304030.
Поддерево 304030: завершило свои задачи и ожидает дочерние поддеревья.
Поддерево 304030: ожидаем окончания поддерева с pid = 304032.
Поддерево 304030: ожидаем окончания поддерева с pid = 304034.
Поддерево 304029: ожидаем окончания поддерева с pid = 304031.
Поддерево 304029: получена перенаправленная задачи на 0 уровней вверх, id = 6.
Поддерево 304029: выполняется сортировка задачи id = 6.
Поддерево 304031: задача перенаправлена.
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
********************
```

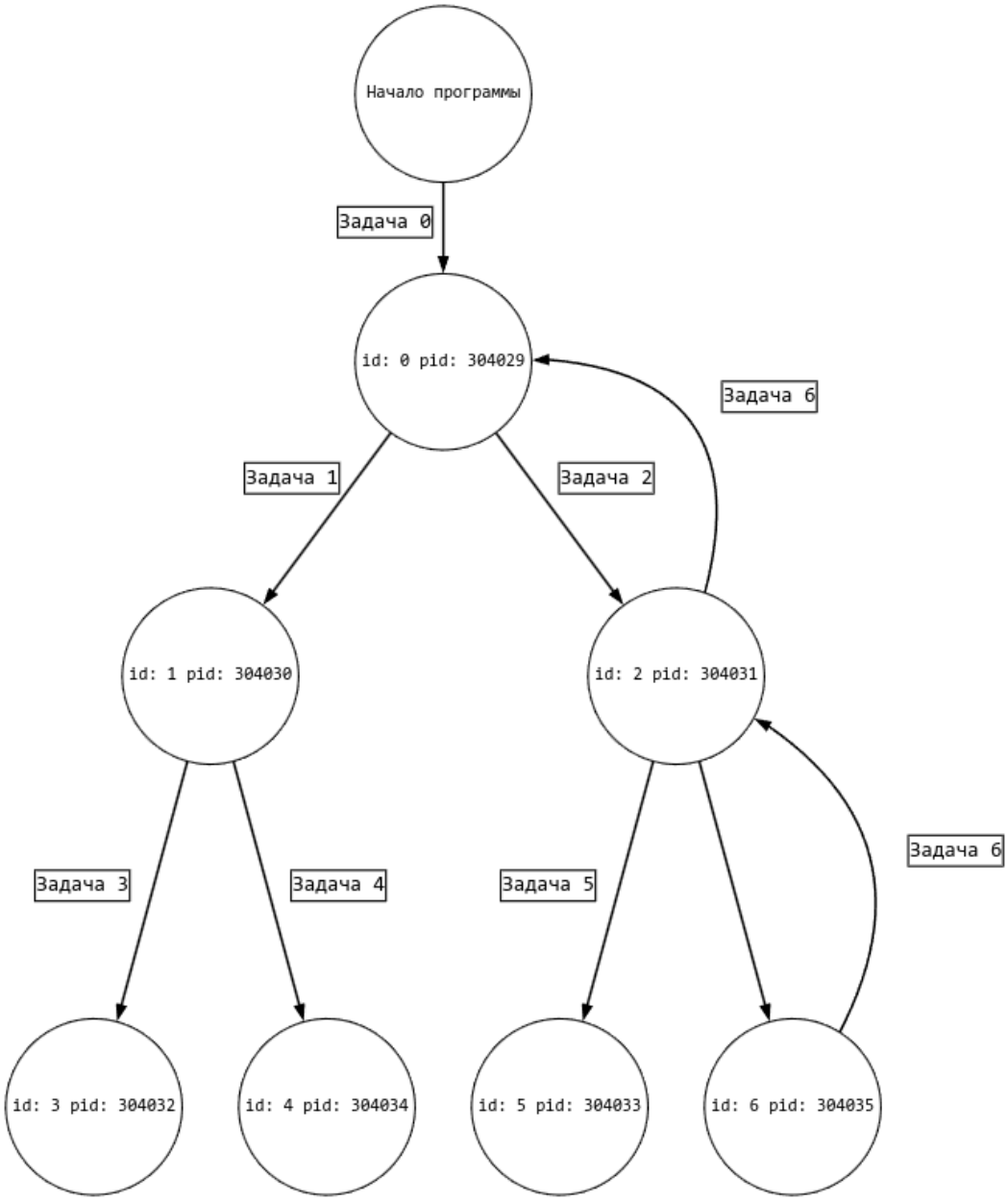
Вывод htop когда листья ”работают в процессе работы и после:







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.