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:
ст. пр. Островский Алексей Мичеславович

Белгород 2024 г.

Лабораторная работа №1

Системные вызовы. Базовая работа с процессами в ОС Linux.

Вариант 8

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

Выполнить индивидуальное задание, закрепляющее на практике полученные знания.

Задание:

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).

1. `pid_t getpid()` - возвращает pid текущего процесса.
2. `pid_t fork()` - создаёт процесс, возвращает 0 в дочернем процессе и pid дочернего процесса в родительском потоке. Возвращает -1, если создать процесс не удалось.
3. `pid_t getppid()` - возвращает pid родительского процесса.
4. `pid_t waitpid(pit_t process, int& status, int options)` - ожидает окончание выполнения процесса с PID process, сохраняет статус выполнения в переменную по адресу status, настройки передаются в options.

Напишем программу для создания бинарного дерева с семью вершинами:

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

int main()
{
    int status = 0;
    /*
    Получаем PID текущего процесса, корня дерева и выводим его. Дерево:

    0

    */
    printf("Мы начали в корне 0! Тут pid = %d.\n", getpid());

    /*
    Создаём левый элемент для корня 0, создаём процесс.
    Если node_1 == 0, значит мы находимся в дочернем листе.
    Иначе смотреть ниже для создания правого элемента для корня
    дерева.
```

```
*/
pid_t node_1 = fork();
if (node_1 == 0)
{
    /*
    Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

      0
     / \
    1  ...

    */
    printf("Мы в поддереве 1! Тут pid = %d, ppid = %d.\n", getpid(), getppid());

    // Аналогично создзаём левый элемент для поддерева 1.
    // если это дочерний процесс - это лист, выполняем работу.
    pid_t node_3 = fork();
    if (node_3 == 0)
    {
        /*
        Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

            0
           / \
          1  ...
         / \
        3  ...
        */
        printf("Мы в листе 3! Тут pid = %d, ppid = %d.\n", getpid(), getppid());
        // Выполняем работу
        sleep(2);
        // И выходим из процесса
        exit(0);
    }

    // Аналогично создзаём правый элемент для поддерева 1.
    // если это дочерний процесс - это лист, выполняем работу.
    pid_t node_4 = fork();
    if (node_4 == 0)
    {
        /*
        Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

             0
            / \
           1  ...
          / \
        ...  4
        */
        printf("Мы в листе 4! Тут pid = %d, ppid = %d.\n", getpid(), getppid());
        // Выполняем работу
        sleep(3);
        // И выходим из процесса
        exit(0);
```

```
    }

    // Ожидаем, когда закончат выполнение листы поддерева и если в них возникает ошибка
    // возвращаем ошибку
    printf("Ожидаем окончания листа 3 с pid = %d.\n", node_3);
    waitpid(node_3, &status, 0);
    if (status)
    {
        printf("Лист 3 завершился с ошибкой!\n");
        exit(status);
    }

    printf("Ожидаем окончания листа 4 с pid = %d.\n", node_4);
    waitpid(node_4, &status, 0);
    if (status)
    {
        printf("Лист 4 завершился с ошибкой!\n");
        exit(status);
    }

    // Листы выполнились корректно, возвращаем 0.
    exit(0);
}

/*
Создаём правый элемент для корня 0, создаём процесс.
Если node_2 == 0, значит мы находимся в дочернем листе.
Иначе смотреть ниже для ожидания окончания выполнения элементов
дерева
*/
pid_t node_2 = fork();
if (node_2 == 0)
{
    /*
    Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

       0
      / \
    ...  2

    */
    printf("Мы в поддереве 2! Тут pid = %d, ppid = %d.\n", getpid(), getppid());

    // Аналогично создзаём левый элемент для поддерева 2.
    // если это дочерний процесс - это лист, выполняем работу.
    pid_t node_5 = fork();
    if (node_5 == 0)
    {
        /*
        Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

           0
          / \
        ...  2
```

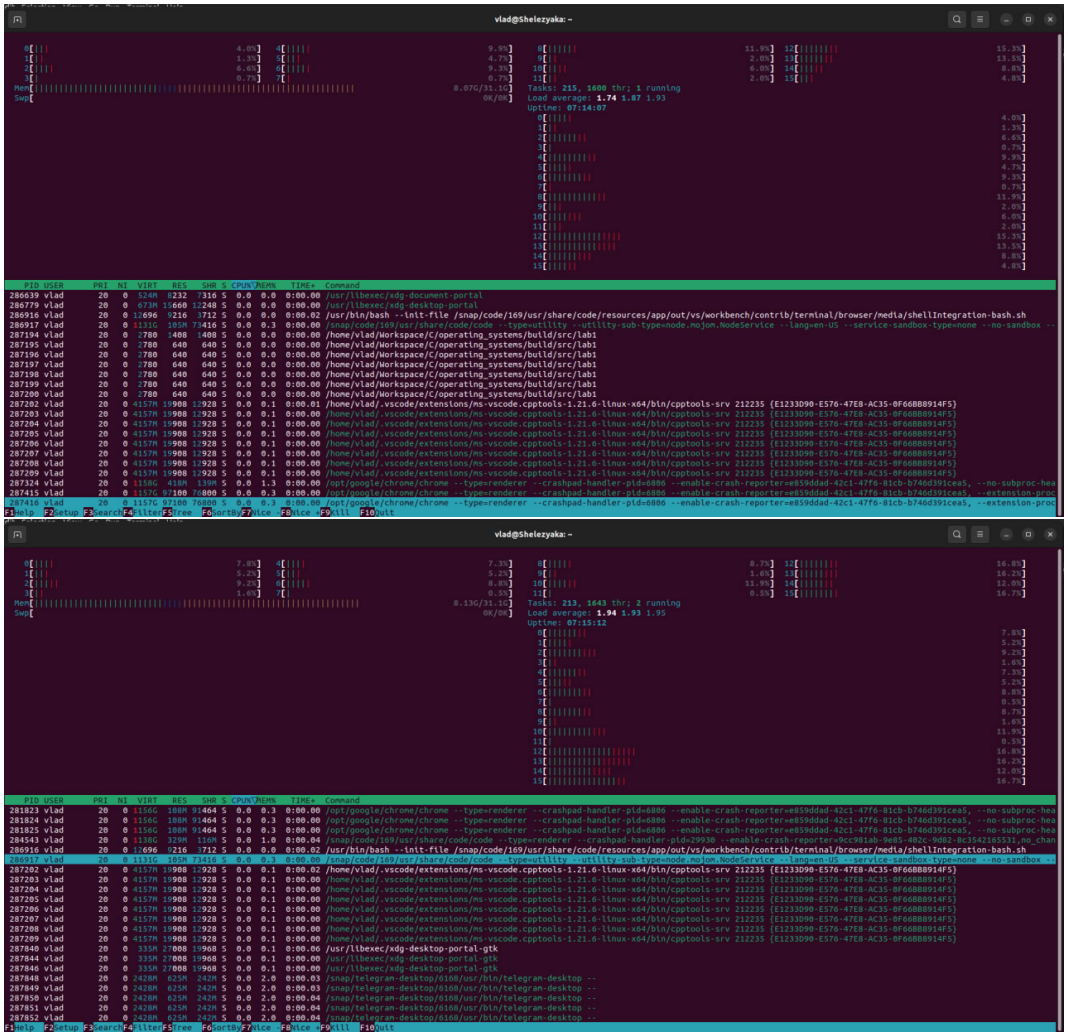
```
            / \
           5  ...
        */
        printf("Мы в листе 5! Тут pid = %d, ppid = %d.\n", getpid(), getppid());
        // Выполняем работу
        sleep(4);
        // И выходим из процесса
        exit(0);
    }

    // Иначе - создаём новый лист. Если это дочерний процесс - это лист
    // и выполняем работу
    pid_t node_6 = fork();
    if (node_6 == 0)
    {
        /*
        Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

           0
          / \
        ...  2
            / \
          ...  6
        */
        printf("Мы в листе 6! Тут pid = %d, ppid = %d.\n", getpid(), getppid());
        // Выполняем работу
        sleep(5);
        // И выходим из процесса
        exit(0);
    }

    // Ожидаем, когда закончат выполнение листы поддерева и если в них возникает ошибка
    // возвращаем ошибку
    printf("Ожидаем окончания листа 5 с pid = %d.\n", node_5);
    waitpid(node_5, &status, 0);
    if (status)
    {
        printf("Лист 5 завершился с ошибкой!\n");
        exit(status);
    }

    printf("Ожидаем окончания листа 6 с pid = %d.\n", node_6);
    waitpid(node_6, &status, 0);
    if (status)
    {
        printf("Лист 6 завершился с ошибкой!\n");
        exit(status);
    }

    // Листы выполнились корректно, возвращаем 0.
    exit(0);
}

/*
```

```
    Ожидаем, пока левый элемент node_1 выполняет свою работу. Если
    он выполняется с ошибкой, возвращаем эту ошибку.
    */
    printf("Ожидаем окончания поддерева 1 с pid = %d.\n", node_1);
    waitpid(node_1, &status, 0);
    if (status)
    {
        printf("Поддерево 1 завершилось с ошибкой!\n");
        exit(status);
    }

    /*
    Ожидаем, пока правый элемент node_2 выполняет свою работу. Если
    он выполняется с ошибкой, возвращаем эту ошибку.
    */
    printf("Ожидаем окончания поддерева 2 с pid = %d.\n", node_2);
    waitpid(node_2, &status, 0);
    if (status)
    {
        printf("Поддерево 2 завершилось с ошибкой!\n");
        exit(status);
    }

    /*
    Все процессы-поддеревья выполнены корректно, возвращаем 0.
    */
    return 0;
}
```

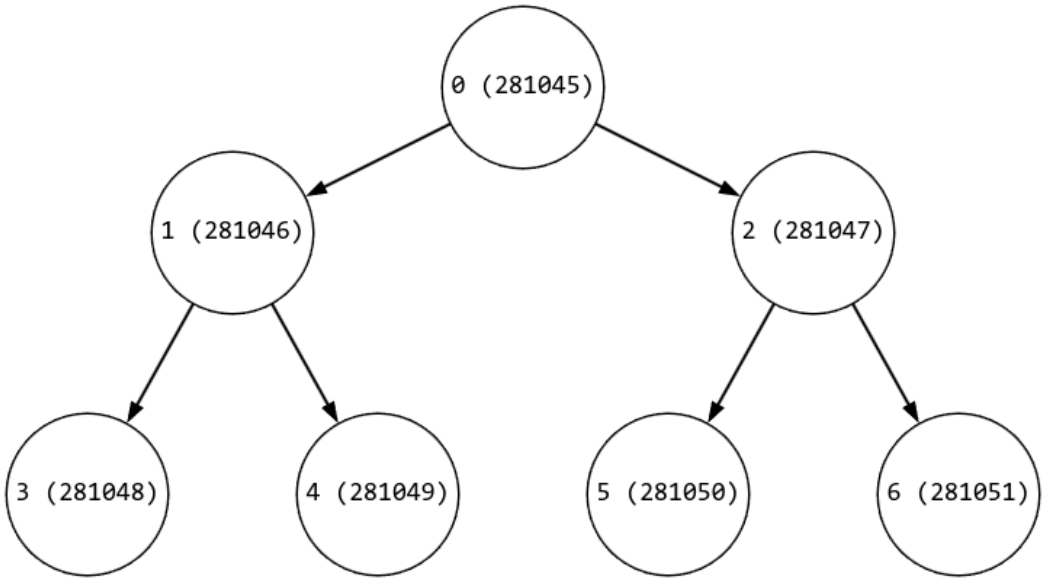
Вывод программы:

```
Мы начали в корне 0! Тут pid = 281045.
Мы в поддереве 1! Тут pid = 281046, ppid = 281045.
Ожидаем окончания поддерева 1 с pid = 281046.
Мы в поддереве 2! Тут pid = 281047, ppid = 281045.
Мы в листе 3! Тут pid = 281048, ppid = 281046.
Ожидаем окончания листа 3 с pid = 281048.
Мы в листе 4! Тут pid = 281049, ppid = 281046.
Мы в листе 5! Тут pid = 281050, ppid = 281047.
Ожидаем окончания листа 5 с pid = 281050.
Мы в листе 6! Тут pid = 281051, ppid = 281047.
Ожидаем окончания листа 4 с pid = 281049.
Ожидаем окончания поддерева 2 с pid = 281047.
Ожидаем окончания листа 6 с pid = 281051.
```

Вывод htop когда листья ”работают”и после:



Полученное дерево:



Альтернативный вариант с использованием for-loop:

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

#define MAX_LEVEL 3
```

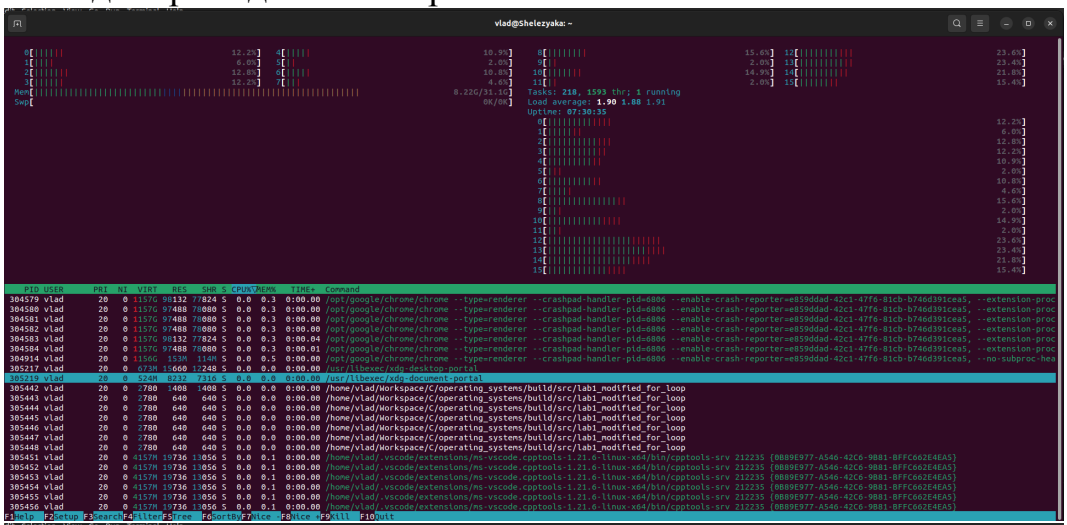
```c
int main() {
    for (int current_level = 0; current_level < MAX_LEVEL - 1; current_level++) {
        /*
        Выводим текущую глубину дерева, PID и PPID.
        */
        printf("Мы находимся на глубине %d\nТекущий pid = %d и ppid = %d\n", current_level, getpid(), getppid());

        /*
        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(3);
    exit(0);

    return 0;
}
```

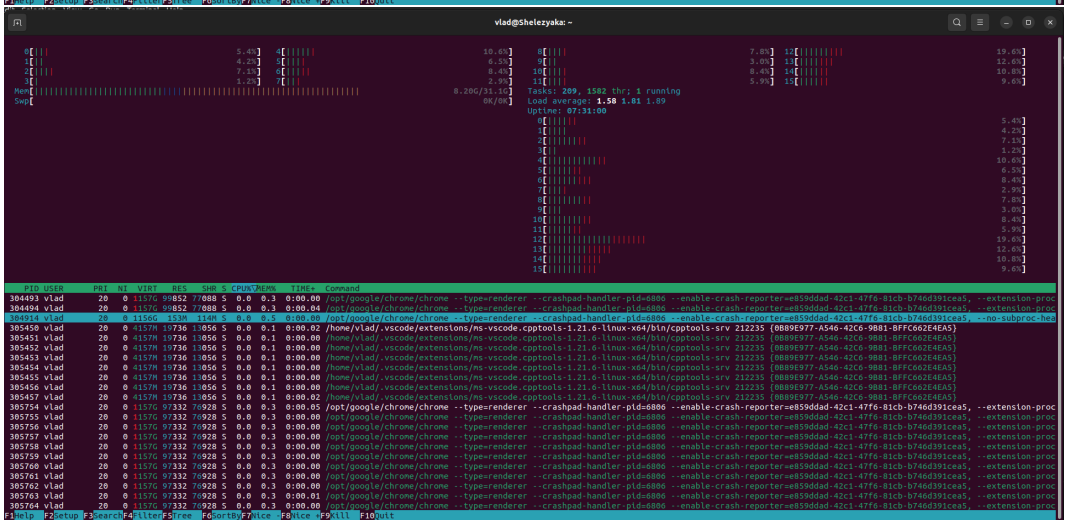
Вывод программы:

```
Мы находимся на глубине 0
Текущий pid = 284852 и ppid = 284565
Мы находимся на глубине 1
```

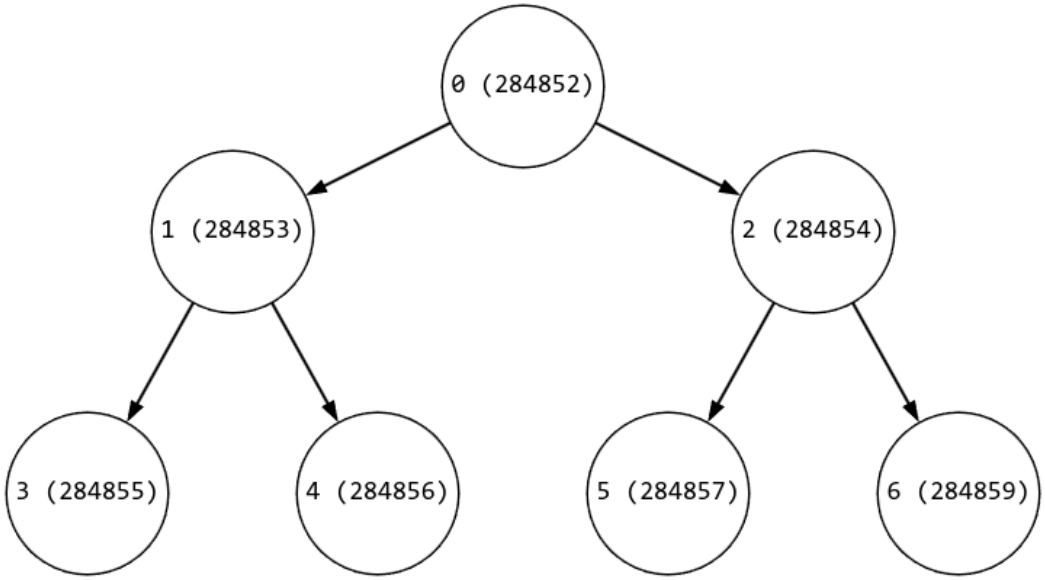
```
Текущий pid = 284853 и ppid = 284852
Мы находимся на глубине 1
Текущий pid = 284854 и ppid = 284852
Мы находимся на глубине 2
Текущий pid = 284855 и ppid = 284853
Мы находимся на глубине 2
Текущий pid = 284856 и ppid = 284853
Мы находимся на глубине 2
Текущий pid = 284857 и ppid = 284854
Мы находимся на глубине 2
Текущий pid = 284859 и ppid = 284854
```

Вывод htop когда листья ”работают”и после:





Полученное дерево:



Превратим дерево в граф, заменив лист 3 на корень дерева. Кол-во процессов должно увеличиться на 2.

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

int main()
{
    int status = 0;
    /*
    Получаем PID текущего процесса, корня дерева и выводим его. Дерево:

    0

    */
    printf("Мы начали в корне 0! Тут pid = %d.\n", getpid());

    /*
    Создаём левый элемент для корня 0, создаём процесс.
    Если node_1 == 0, значит мы находимся в дочернем листе.
    Иначе смотреть ниже для создания правого элемента для корня
    дерева.
    */
    pid_t node_1 = fork();
    if (node_1 == 0)
    {
        /*
        Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

          0
         / \
        1  ...

        */
```

```
        printf("Мы в поддереве 1! Тут pid = %d, ppid = %d.\n", getpid(), getppid());

        // Аналогично создзаём левый элемент для поддерева 1.
        // если это дочерний процесс - это лист, выполняем работу.
        pid_t node_3 = fork();
        if (node_3 == 0)
        {
            /*
            Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

                0
               / \
              1  ...
             / \
            3  ...
            */
            printf("Мы в листе 3! Тут pid = %d, ppid = %d.\nКажется, этот лист собирается расти!\n", getpid(),
↪   getppid());

            // Аналогично создаём дочерние листья для поддерева 3
            pid_t node_7 = fork();
            if (node_7 == 0)
            {
                /*
                Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

                     0
                    / \
                   1  ...
                  / \
                 3  ...
                / \
               7  ...

                Выполняем работу и выходим с кодом 0
                */
                printf("Мы в свежем листе 7! Тут pid = %d, ppid = %d.\n", getpid(), getppid());
                sleep(30);
                exit(0);
            }
            pid_t node_8 = fork();
            if (node_8 == 0)
            {
                /*
                Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

                      0
                     / \
                    1  ...
                   / \
                  3  ...
                 / \
                ...  8
```

```
            Выполняем работу и выходим с кодом 0
            */
            printf("Мы в свежем листе 8! Тут pid = %d, ppid = %d.\n", getpid(), getppid());
            sleep(30);
            exit(0);
        }

        // Мы находимся в родительском поддереве и ожидаем, когда закончат работу листья.
        // если листья закончились с ошибкой, выходим из процесса с ошибкой
        printf("Ожидаем окончания листа 7 с pid = %d.\n", node_7);
        waitpid(node_7, &status, 0);
        if (status)
        {
            printf("Лист 7 завершился с ошибкой!\n");
            exit(status);
        }

        printf("Ожидаем окончания листа 8 с pid = %d.\n", node_8);
        waitpid(node_8, &status, 0);
        if (status)
        {
            printf("Лист 8 завершился с ошибкой!\n");
            exit(status);
        }

        // Листы выполнились корректно, возвращаем 0.
        exit(0);
    }

    // Аналогично создзаём правый элемент для поддерева 1.
    // если это дочерний процесс - это лист, выполняем работу.
    pid_t node_4 = fork();
    if (node_4 == 0)
    {
        /*
        Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

             0
            / \
           1  ...
          / \
        ...  4
        */
        printf("Мы в листе 4! Тут pid = %d, ppid = %d.\n", getpid(), getppid());
        // Выполняем работу
        sleep(30);
        // И выходим из процесса
        exit(0);
    }

    // Ожидаем, когда закончат выполнение листы поддерева и если в них возникает ошибка
    // возвращаем ошибку
    printf("Ожидаем окончания листа 3 с pid = %d.\n", node_3);
    waitpid(node_3, &status, 0);
```

```
    if (status)
    {
        printf("Лист 3 завершился с ошибкой!\n");
        exit(status);
    }

    printf("Ожидаем окончания листа 4 с pid = %d.\n", node_4);
    waitpid(node_4, &status, 0);
    if (status)
    {
        printf("Лист 4 завершился с ошибкой!\n");
        exit(status);
    }

    // Листы выполнились корректно, возвращаем 0.
    exit(0);
}

/*
Создаём правый элемент для корня 0, создаём процесс.
Если node_2 == 0, значит мы находимся в дочернем листе.
Иначе смотреть ниже для ожидания окончания выполнения элементов
дерева
*/
pid_t node_2 = fork();
if (node_2 == 0)
{
    /*
    Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

       0
      / \
    ...  2

    */
    printf("Мы в поддереве 2! Тут pid = %d, ppid = %d.\n", getpid(), getppid());

    // Аналогично создзаём левый элемент для поддерева 2.
    // если это дочерний процесс - это лист, выполняем работу.
    pid_t node_5 = fork();
    if (node_5 == 0)
    {
        /*
        Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

           0
          / \
        ...  2
            / \
           5  ...
        */
        printf("Мы в листе 5! Тут pid = %d, ppid = %d.\n", getpid(), getppid());
        // Выполняем работу
        sleep(30);
```

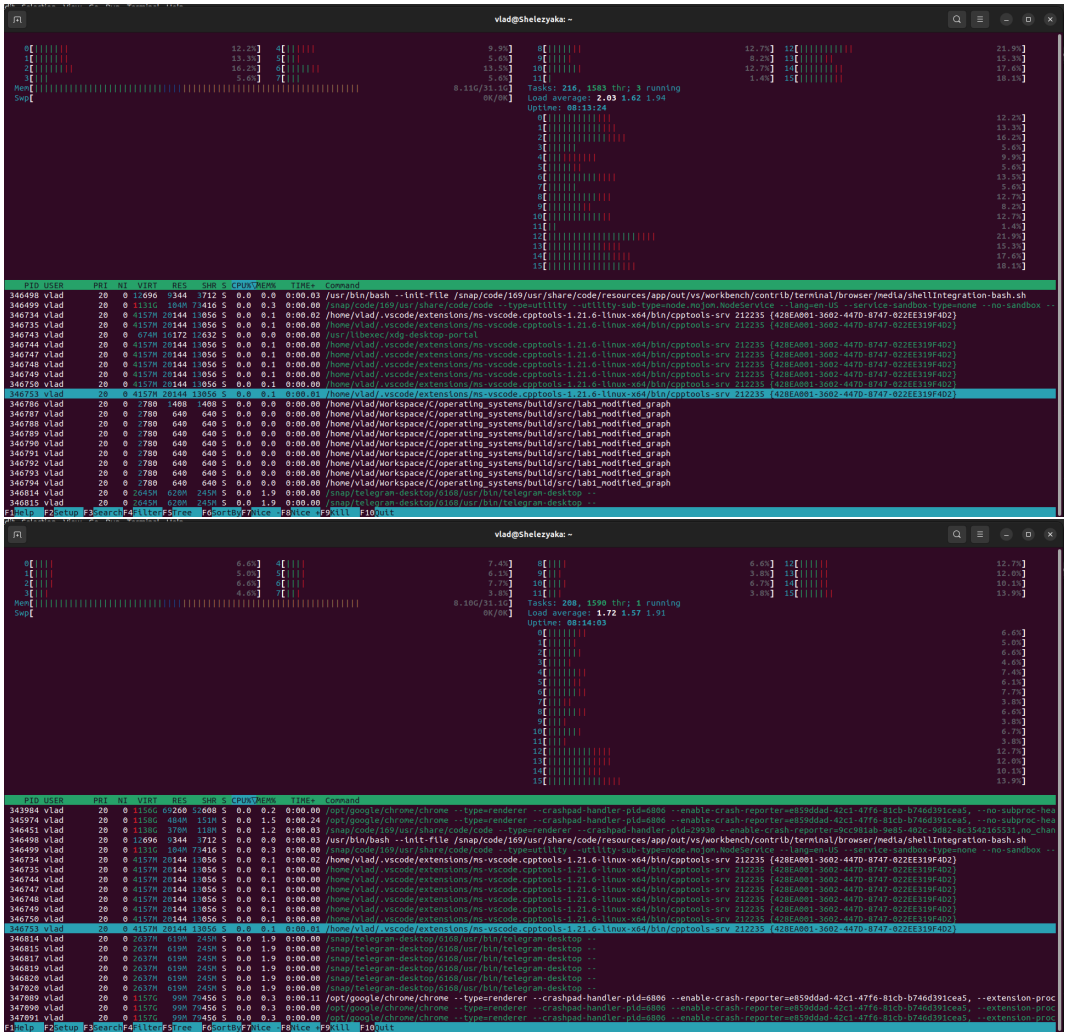
```
        // И выходим из процесса
        exit(0);
    }

    // Иначе - создаём новый лист. Если это дочерний процесс - это лист
    // и выполняем работу
    pid_t node_6 = fork();
    if (node_6 == 0)
    {
        /*
        Получаем PID текущего процесса-поддерева и PID родительского элемента PPID. Дерево:

           0
          / \
        ...  2
            / \
          ...  6
        */
        printf("Мы в листе 6! Тут pid = %d, ppid = %d.\n", getpid(), getppid());
        // Выполняем работу
        sleep(30);
        // И выходим из процесса
        exit(0);
    }

    // Ожидаем, когда закончат выполнение листы поддерева и если в них возникает ошибка
    // возвращаем ошибку
    printf("Ожидаем окончания листа 5 с pid = %d.\n", node_5);
    waitpid(node_5, &status, 0);
    if (status)
    {
        printf("Лист 5 завершился с ошибкой!\n");
        exit(status);
    }

    printf("Ожидаем окончания листа 6 с pid = %d.\n", node_6);
    waitpid(node_6, &status, 0);
    if (status)
    {
        printf("Лист 6 завершился с ошибкой!\n");
        exit(status);
    }

    // Листы выполнились корректно, возвращаем 0.
    exit(0);
}

/*
Ожидаем, пока левый элемент node_1 выполняет свою работу. Если
он выполняется с ошибкой, возвращаем эту ошибку.
*/
printf("Ожидаем окончания поддерева 1 с pid = %d.\n", node_1);
waitpid(node_1, &status, 0);
if (status)
```

```
    {
        printf("Поддерево 1 завершилось с ошибкой!\n");
        exit(status);
    }

    /*
    Ожидаем, пока правый элемент node_2 выполняет свою работу. Если
    он выполняется с ошибкой, возвращаем эту ошибку.
    */
    printf("Ожидаем окончания поддерева 2 с pid = %d.\n", node_2);
    waitpid(node_2, &status, 0);
    if (status)
    {
        printf("Поддерево 2 завершилось с ошибкой!\n");
        exit(status);
    }

    /*
    Все процессы-поддеревья выполнены корректно, возвращаем 0.
    */
    return 0;
}
```

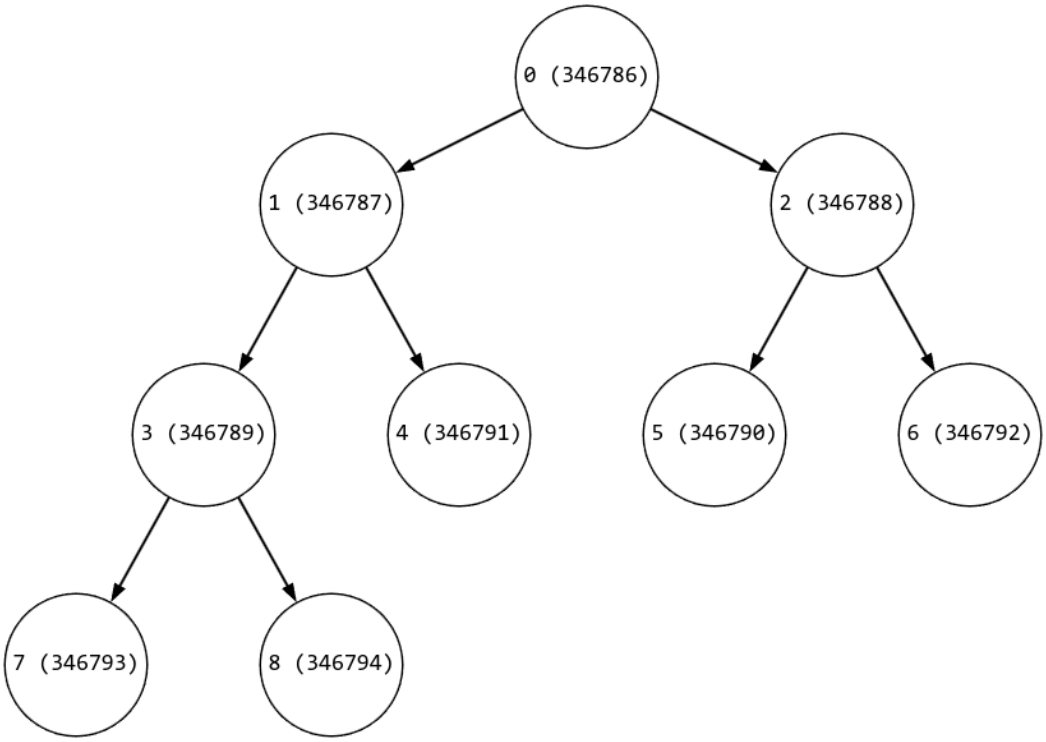
Вывод программы:

```
Мы начали в корне 0! Тут pid = 346786.
Мы в поддереве 1! Тут pid = 346787, ppid = 346786.
Ожидаем окончания поддерева 1 с pid = 346787.
Мы в поддереве 2! Тут pid = 346788, ppid = 346786.
Мы в листе 3! Тут pid = 346789, ppid = 346787.
Кажется, этот лист собирается расти!
Ожидаем окончания листа 5 с pid = 346790.
Мы в листе 5! Тут pid = 346790, ppid = 346788.
Ожидаем окончания листа 3 с pid = 346789.
Мы в листе 4! Тут pid = 346791, ppid = 346787.
Мы в листе 6! Тут pid = 346792, ppid = 346788.
Мы в свежем листе 7! Тут pid = 346793, ppid = 346789.
Ожидаем окончания листа 7 с pid = 346793.
Мы в свежем листе 8! Тут pid = 346794, ppid = 346789.
Ожидаем окончания листа 6 с pid = 346792.
Ожидаем окончания листа 8 с pid = 346794.
Ожидаем окончания листа 4 с pid = 346791.
Ожидаем окончания поддерева 2 с pid = 346788.
```

Вывод htop когда листья ”работают”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.