Министерство образования Республики Беларусь
учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа № 4

«BASH: ПОТОКИ ДАННЫХ.
ПРОГРАММИРОВАНИЕ»»

Выполнил: студент 2-го курса
группы АС-65
Осовец М.М
Проверил: Степанчук В.И.

Брест 2024

Ход работы

Цель работы: изучить работу компилятора GCC, системные вызовы и процессы в ОС Linux.

Задание для выполнения

Написать программу, которая будет реализовывать следующие функции:

• сразу после запуска получает и сообщает свой ID и ID родительского процесса;

• перед каждым выводом сообщения об ID процесса и родительского процесса эта информация получается заново;

• порождает процессы, формируя генеалогическое дерево согласно варианту, сообщая, что "процесс с ID таким-то породил процесс с таким-то ID";

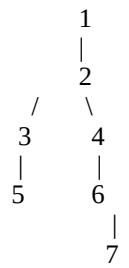
• перед завершением процесса сообщить, что "процесс с таким-то ID и таким-то ID родителя завершает работу";

• один из процессов должен вместо себя запустить программу, указанную в варианте задания.

На основании выходной информации программы предыдущего пункта изобразить генеалогическое дерево процессов (с указанием идентификаторов процессов). Объяснить каждое выведенное сообщение и их порядок в предыдущем пункте.

14	0 1 2 2 3 4 6	7	who

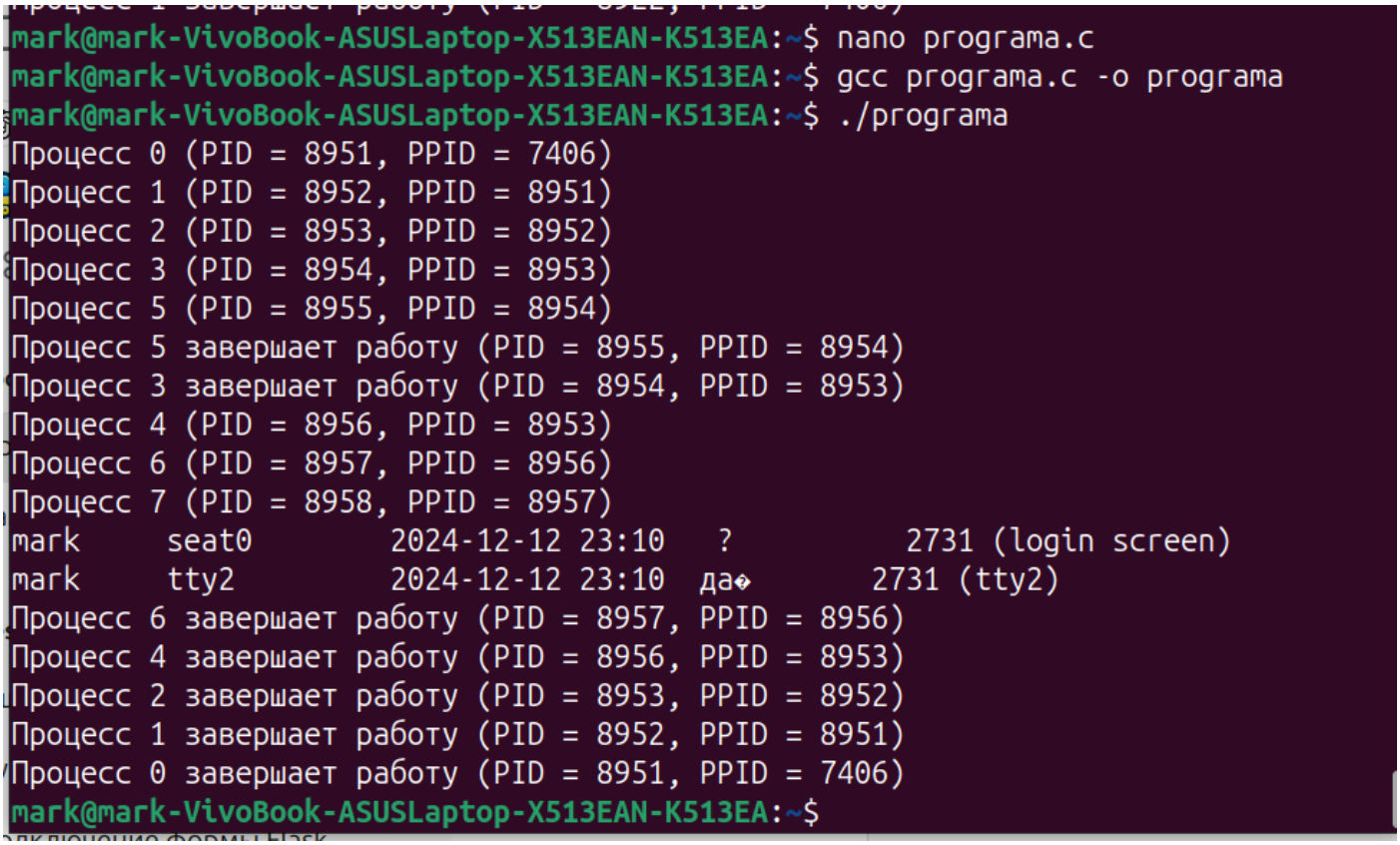
1	2	3	4	5	6	7
0	1	2	2	3	4	6




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

int main() {
    pid_t pid1, pid2, pid3, pid4, pid5, pid6, pid7;

    printf("Процесс 0 (PID = %d, PPID = %d)\n", getpid(), getppid());

    // Создаём процесс 1
    if ((pid1 = fork()) == -1) {
        perror("Ошибка создания процесса 1");
        exit(1);
    } else if (pid1 == 0) {
        printf("Процесс 1 (PID = %d, PPID = %d)\n", getpid(), getppid());

        // Создаём процесс 2
        if ((pid2 = fork()) == -1) {
            perror("Ошибка создания процесса 2");
            exit(1);
        } else if (pid2 == 0) {
            printf("Процесс 2 (PID = %d, PPID = %d)\n", getpid(), getppid());

            // Создаём процесс 3
            if ((pid3 = fork()) == -1) {
                perror("Ошибка создания процесса 3");
                exit(1);
            } else if (pid3 == 0) {
                printf("Процесс 3 (PID = %d, PPID = %d)\n", getpid(), getppid());

                // Создаём процесс 5
```

```
                if ((pid5 = fork()) == -1) {
                    perror("Ошибка создания процесса 5");
                    exit(1);
                } else if (pid5 == 0) {
                    printf("Процесс 5 (PID = %d, PPID = %d)\n", getpid(),
getppid());
                    printf("Процесс 5 завершает работу (PID = %d, PPID = %d)\n",
getpid(), getppid());
                    exit(0);
                }
                wait(NULL); // Ожидание завершения процесса 5
                printf("Процесс 3 завершает работу (PID = %d, PPID = %d)\n",
getpid(), getppid());
                exit(0);
            }

            wait(NULL); // Ожидание завершения процесса 3

            // Создаём процесс 4
            if ((pid4 = fork()) == -1) {
                perror("Ошибка создания процесса 4");
                exit(1);
            } else if (pid4 == 0) {
                printf("Процесс 4 (PID = %d, PPID = %d)\n", getpid(), getppid());

                // Создаём процесс 6
                if ((pid6 = fork()) == -1) {
                    perror("Ошибка создания процесса 6");
                    exit(1);
                } else if (pid6 == 0) {
                    printf("Процесс 6 (PID = %d, PPID = %d)\n", getpid(),
getppid());

                    // Создаём процесс 7
                    if ((pid7 = fork()) == -1) {
                        perror("Ошибка создания процесса 7");
                        exit(1);
                    } else if (pid7 == 0) {
                        printf("Процесс 7 (PID = %d, PPID = %d)\n", getpid(),
getppid());

                        // Выполняем команду who
                        execlp("who", "who", "-u", NULL);
                        perror("Ошибка exec в процессе 7");
                        exit(1);
                    }

                    wait(NULL); // Ожидание завершения процесса 7
                    printf("Процесс 6 завершает работу (PID = %d, PPID = %d)\n",
getpid(), getppid());
                    exit(0);
                }

                wait(NULL); // Ожидание завершения процесса 6
                printf("Процесс 4 завершает работу (PID = %d, PPID = %d)\n",
getpid(), getppid());
                exit(0);
            }

            wait(NULL); // Ожидание завершения процесса 4
            printf("Процесс 2 завершает работу (PID = %d, PPID = %d)\n", getpid(),
getppid());
            exit(0);
        }

        wait(NULL); // Ожидание завершения процесса 2
```

```
        printf("Процесс 1 завершает работу (PID = %d, PPID = %d)\n", getpid(),
getppid());
        exit(0);
    }

    wait(NULL); // Ожидание завершения процесса 1
    printf("Процесс 0 завершает работу (PID = %d, PPID = %d)\n", getpid(),
getppid());
    return 0;
}
```



Вывод: мы ознакомились с компилятором GCC. Изучили процессы, их идентификацию и иерархию. Изучили системные вызовы, в частности вызов fork().