Кафедра прикладной математики и кибернетики

Выполнили:

Студенты 3 курса группы ИП-111  
Корнилов А.А.,  
Попов М.И.,

Толкач А.А.

Проверил:

Профессор кафедры ПМиК  
Малков Е.А.

ЛАБОРАТОРНАЯ РАБОТА №3

По дисциплине: «Операционные системы»

Сибирский государственный университет  
телекоммуникаций и информатики

Новосибирск, 2023

Министерство цифрового развития, связи  
и массовых коммуникаций Российской Федерации

**Цель:** знакомство с процессами Linux.

**Упражнение 1.** Протестируйте программы, рассмотренные на Лекции 3.

**Упражнение 2.** Создайте процесс с помощью вызова fork, с помощью команд ps и grep получите информацию о созданных вами родительском и дочернем процессах. Используя команду kill убейте родительский процесс, продолжил ли выполняться дочерний процесс?

**Упражнение 3.** Создайте дерево процессов с помощью вызова fork. С помощью команды pstree найдите поддерево созданных процессов. В каталоге /proc виртуальной файловой системы найдите папки с именами, совпадающими с идентификаторами созданных процессов, и просмотрите содержимое папок task/children.

**Выполнение работы:**

1. Скомпилируем и выведем результаты работы программ показанных на лекции (рис.1) (рис.2).

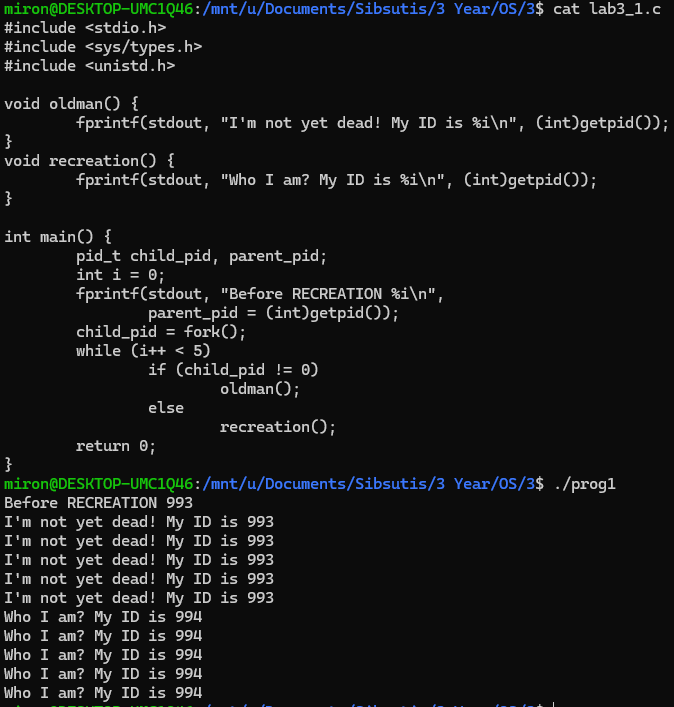


Рисунок 1 – Программа примера создания процесса

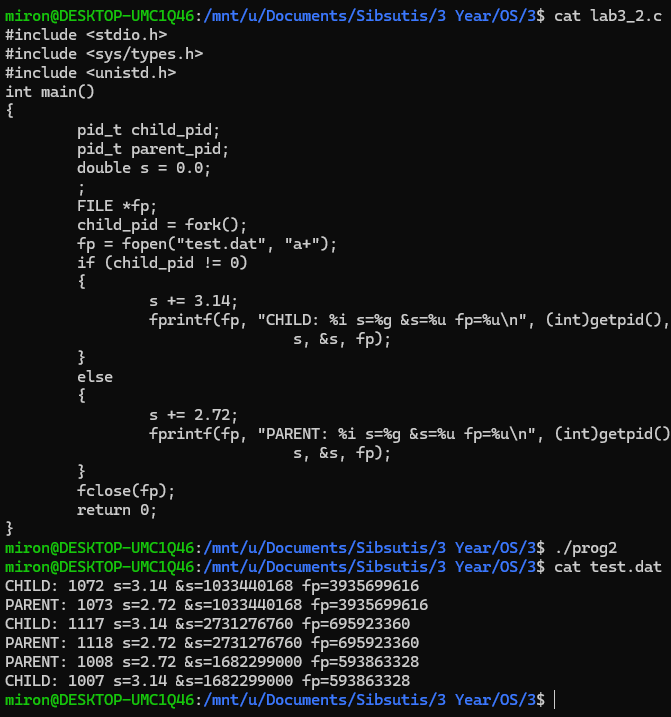


Рисунок 2 – Создание и запись файла PID-ов родительского и дочернего процесса

2.Создана программа создания дочернего и родительского процесса (рис.3) с остановкой после создания

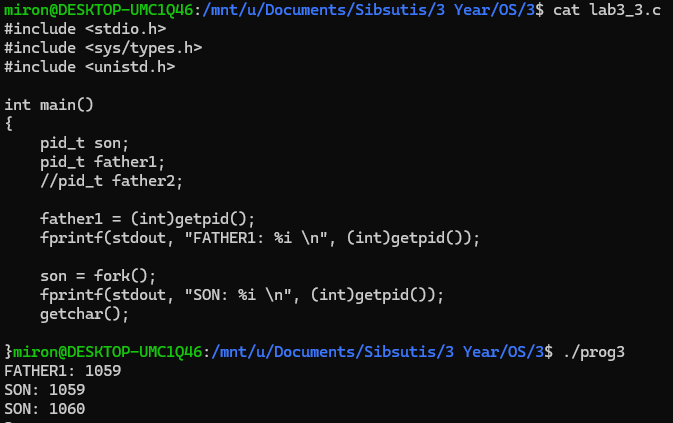


Рисунок 3 – Код программы и её запуск

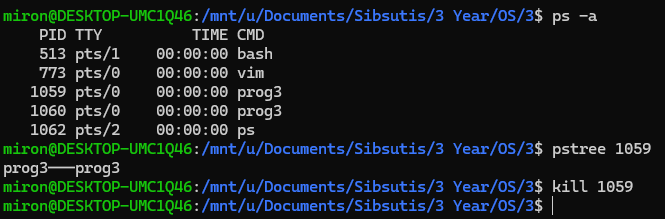


Рисунок 4 – Просмотр дерева процессов и снятия задачи с родительского процесса

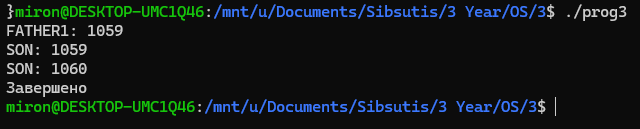


Рисунок 5 – Результат снятия задачи с родительского процесса

3. Написана программа создания дерева процессов

|  |
| --- |
| #include <stdlib.h>  #include <stdio.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/wait.h>  int level = 1;  char const offsets[] = "\t\t\t\t\t\t\t\t";  pid\_t create\_child\_process(int(\*child\_fn)()) {      fflush(stdout);      fflush(stderr);      pid\_t child\_pid = fork();      switch(child\_pid) {          case 0:              ++level;              exit(child\_fn());          case -1:              abort();          default:              printf("%.\*s %u spawned %u\n", level, offsets, (unsigned)getpid(), (unsigned)child\_pid);              return child\_pid;      }  }  void wait\_for\_any\_child() {      int wstatus;      pid\_t child\_pid = wait(&wstatus);      if(child\_pid == -1)          abort();      printf("%.\*s %u terminated\n", level, offsets, (unsigned)child\_pid);  }  int p2() { return 0; }  int p5() { return 0; }  int p6() { return 0; }  int p7() { return 0; }  int p4() {      create\_child\_process(p5);      create\_child\_process(p6);      create\_child\_process(p7);      getchar();      wait\_for\_any\_child();      wait\_for\_any\_child();      wait\_for\_any\_child();      return 0;  }  int p3() {      create\_child\_process(p4);      wait\_for\_any\_child();      return 0;  }  int p1() {      printf("%u started\n", (unsigned)getpid());      create\_child\_process(p2);      create\_child\_process(p3);      wait\_for\_any\_child();      wait\_for\_any\_child();      printf("%u terminated\n", (unsigned)getpid());      return 0;  }  int main() {      p1();      getchar();  } |

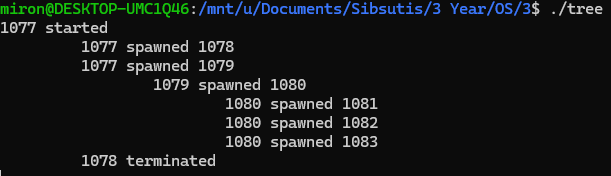


Рисунок 6 - Результат работы программы

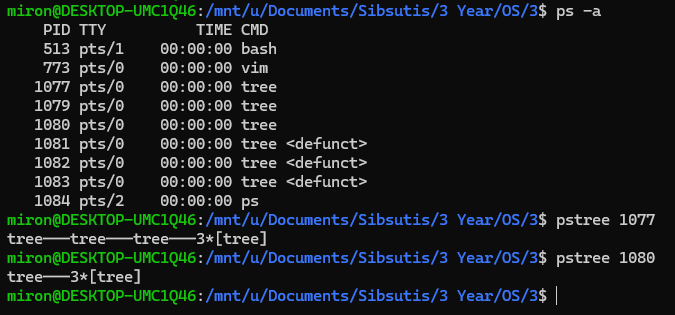


Рисунок 7 – дерево процессов программы



Рисунок 8 – Содержимое файла children для PID1080