Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу

«Операционные системы»

Освоение принципов работы с файловыми системами. Обеспечение обмена данных между процессами посредством технологии «File Mapping»

Студент: Аминов Степан Сеегеевич
Группа: М80 – 308Б
Вариант: 3
Преподаватель: Миронов Евгений Сергеевич
Дата:
Оценка:
Подпись: _____

Москва, 2021

1 Постановка задачи

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

3 вариант) Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int.

2 Общий метод и алгоритм решения

При запуске программы пользователю предлагается ввести имя файла. В этот файл будет записываться вывод соответствующих процессов. Если пользователь ввёл имя несуществующего файла, он будет создан.

После запуска программы создаётся дочерний процесс и канал fd. Родительский процесс считывает числа с консольного ввода. Передача чисел дочернему процессу осуществляется через pipe. Дочерний процесс производит необходимые вычисления и выводит результат в созданный файл. При делении на ноль дочерний процесс выдаст ошибку.

3 Файлы программы

main.c

```
#include "unistd.h"
#include "stdio.h"

int main()
{
   char filename[100];
   char buf[100];
   int nums[3];
   int krya = 0;
   scanf("%s", filename);
   FILE *file = fopen(filename,"w");

   int fd[2];
         pipe(fd);
   int id = fork();
         if (id == -1){
                  perror("fork error");
                  return -1;
         }if(id > 0){
      printf("[%d] It's parent. Child id: %d\n", getpid(), id);
      fflush(stdout);
      int x, y, z;
      scanf("%d %d %d", &x, &y,&z);
      read(krya,nums,3*sizeof(int));

      write(fd[1], nums[0], sizeof(int));
      write(fd[1], &y, sizeof(int));
      write(fd[1], &z, sizeof(int));

   }if (id == 0){
      int x, y, z;
      float res1, res2;
      printf("[%d] It's child\n", getpid());
      fflush(stdout);
      read(fd[0], &x, sizeof(int));
      read(fd[0], &y, sizeof(int));
      read(fd[0], &z, sizeof(int));

      if((y == 0)||(z == 0)){
         return -1;
      }

      res1 = x / y;
      res2 = res1 / z;
```

```
        fprintf(file,"%f", res2);

        fclose(file);

        close(fd[0]);
        close(fd[1]);

        return 0;
            }
}
```

4 Демонстрация работы программы

```
magic@magical:~/labs/os$ ./a.out
test
[3666] It's parent. Child id: 3667
[3667] It's child
140
4
2
[140] It's child
[140] It's child
[4] It's child
[2] It's child
[17.500000] It's child
magic@magical:~/labs/os$ cat test
17.500000
magic@magical:~/labs/os$ strace -f -e trace="%process,read,write,dup2,mmap" -o
strace_log.txt ./a.out
test2
[3747] It's parent. Child id: 3748
[3748] It's child
140
4
2
[140] It's child
[140] It's child
[4] It's child
[2] It's child
[17.500000] It's child
magic@magical:~/labs/os$ cat strace_log.txt
3747  execve("./a.out", ["./a.out"], 0x7ffc9ed22c18 /* 66 vars */) = 0
3747  arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe51e42a00) = -1 EINVAL (Invalid argument)
3747  mmap(NULL, 102741, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7feccecbe000
3747  read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0\0"...,
832) = 832
3747  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7feccecbc000
3747  mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fecceaca000
3747  mmap(0x7fecceaef000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x25000) = 0x7fecceaef000
3747  mmap(0x7feccec67000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x19d000) = 0x7feccec67000
3747  mmap(0x7feccecb2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1e7000) = 0x7feccecb2000
3747  mmap(0x7feccecb8000, 13528, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7feccecb8000
3747  arch_prctl(ARCH_SET_FS, 0x7feccecbd540) = 0
3747  read(0, "test2\n", 1024)          = 6
3747  clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7feccecbd810) = 3748
3747  write(1, "[3747] It's parent. Child id: 37"..., 35) = 35
3748  write(1, "[3748] It's child\n", 18 <unfinished ...>
3747  read(0,  <unfinished ...>
3748  <... write resumed>)              = 18
3748  read(4,  <unfinished ...>
3747  <... read resumed>"140\n", 1024)  = 4
3747  read(0, "4\n", 1024)              = 2
3747  read(0, "2\n", 1024)              = 2
3747  write(5, "\214\0\0\0", 4)         = 4
3748  <... read resumed>"\214\0\0\0", 4) = 4
3747  write(5, "\4\0\0\0", 4 <unfinished ...>
3748  write(1, "[140] It's child\n", 17 <unfinished ...>
3747  <... write resumed>)              = 4
3748  <... write resumed>)              = 17
3748  read(4,  <unfinished ...>
```

```
3747  write(5, "\2\0\0\0", 4 <unfinished ...>
3748  <... read resumed>"\4\0\0\0", 4)  = 4
3748  write(1, "[140] It's child\n", 17 <unfinished ...>
3747  <... write resumed>)              = 4
3748  <... write resumed>)              = 17
3748  write(1, "[4] It's child\n", 15)  = 15
3747  exit_group(0 <unfinished ...>
3748  read(4,  <unfinished ...>
3747  <... exit_group resumed>)         = ?
3748  <... read resumed>"\2\0\0\0", 4)  = 4
3748  write(1, "[2] It's child\n", 15)  = 15
3747  +++ exited with 0 +++
3748  write(1, "[17.500000] It's child\n", 23) = 23
3748  write(3, "17.500000", 9)          = 9
3748  exit_group(0)                     = ?
3748  +++ exited with 0 +++
```

5 Выводы

В ходе решения данной лабораторной работы я научился работать с каналами pipe и отработал их применение на практике. Также я ещё раз попрактиовался работе с файлами, использовал функции fork для создания дочернего процесса внутри программы. В целом я считаю полезными полученные знания, так как в этой лабораторной мы изучили основы межпроцессорного взаимодействия, которое является основой для множества больших программ.