Московский авиационный институт

(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Операционные Системы»

Лабораторная работа № 4

Тема: Освоение принципов работы с файловыми системами и обеспечение обмена данных между процессами посредством технологии «File mapping»

Студент: Семин Александр Витальевич

Группа: М8О-206Б-20

Преподаватель: Соколов Андрей Алексеевич

Дата:

Оценка:

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 решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант 9

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида: «число число число». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. 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 float.

2. **Набор тестов**

Первый тестовый набор:

```
…:~/Desktop/OS_labs/l4$ ./a.out file2.txt
Enter number of lines in your text file:
3
Child's process was created. Id is 5247
In child
In parent
line 1: res1 = 7.7311, res2 =  3.3836
line 2: res1 = 3.9178, res2 =  5.5895
line 3: res1 = 8.9991, res2 =  2.9834
```

file2.txt:
3
110.123 14.244 32.546
321.348 82.021 57.491
31.452 3.495 10.542

Второй тестовый набор: (программа завершается при ошибочном задании данных в исходном файле в процессе выполнения)
…:~/Desktop/OS_labs/l4$ ./a.out file2.txt
Enter number of lines in your text file:
2
Child's process was created. Id is 5057
In child

file2.txt:
2
110.123 14.244 32.546
321.348 82.021

3. **Листинг программы**

child.c

```
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <stdbool.h>
#include "iout.h"



void print_num_in_data(char* data, int a, int DATA_COUNT) {
   if (a > 0 && a < 10) {
         data[DATA_COUNT] = '0' + a;
   }
}

void write_in_data(char* data,int* DATA_COUNT, float val) {
   int counter = *DATA_COUNT;
   if (val < 0) {
```

```
        data[counter] = '-';
        counter++;
        val *= -1;
    }
    float left;
    float right = modff(val, &left);

    //вывод целой части
    int l = left;
    int arr[100] = {'\0'};

    int arr_size = reverse(l, arr);
    if (l == 0) {
        data[counter] = '0';
        counter++;
    } else {
        for (int i = arr_size -1; i >= 0; i--) {
            print_num_in_data(data, arr[i], counter);
            counter++;
        }
    }
    data[counter] = '.';
    counter++;

    //вывод дробной часи

    int i = 0;
    while (right > 0) { //0,456
        float new = right*10; //4,56
        right = right*10;
        while (right > 1) right--; // по итогу right = 0,56
        float for_print = new - right; // 4,56 - 0,56
        print_num_in_data(data,for_print,counter); // вывод цифры 4
        counter++;
        //повтор цикла для 0,56 и тд
        i++;
        if (i == 4) {
            break;
        }
    }
    *DATA_COUNT = counter;
}


int main() {
    int N;
    my_read_int(STDIN_FILENO, &N);
    int DATA_COUNT = 0;
// my_read_int(STDIN_FILENO, &N);
    float cur = 0.0, sec = 0.0, third = 0.0;
    int status_trunc = ftruncate(STDOUT_FILENO, sizeof(char)*100*N);
    if (status_trunc < 0) {
    //    my_print("FTRUNCATE ERROR\n");
```

```
            return -1;
      }
      char* data = mmap(0, sizeof(char)*100*N, PROT_WRITE, MAP_SHARED,
STDOUT_FILENO, 0);

      int status = read_float(STDIN_FILENO, &cur);

      while (status == 0 && N > 0) {
            status = read_float(STDIN_FILENO, &sec);
            if (status == -1) //неверное значение
                  return -1;
            if (status == -3) { //перенос строки после второго числа
      //          my_print("Incorrect type of commands in file\n");
                  return -3;
            }

            status = read_float(STDIN_FILENO, &third);

            if (status == -1) // неверное значение
                  return -1;
            if (sec == 0 || third == 0) { // проверка деления на 0
      //          my_print("Error: division by 0 is forbidden!\n");
                  return -4;
            }
            if (status == 0) { //отсутствует перенос строки после 3-го числа
      //          my_print("Wrong commadns! Line should looks like <number
number number<endline>>\n");
                  return -5;
            }
            float res1 = cur / sec;
            float res2 = cur / third;


            write_in_data(data, &DATA_COUNT, res1);
            data[DATA_COUNT] = ' ';
            DATA_COUNT++;
            write_in_data(data, &DATA_COUNT, res2);
            data[DATA_COUNT] = '\n';
            DATA_COUNT++;
            N--;
            status = read_float(STDIN_FILENO, &cur);
      }

      msync(data, sizeof(char)*100*N, MS_SYNC);
      int err = munmap(data, sizeof(char)*100*N);
/* if (err < 0) {
         my_print("MUNMAP ERROR\n");
            return -1;
      }*/

      return 0;
```

parent.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdlib.h>
#include "iout.h"

int main(int argc, char *argv[]) {
   int lines;
   my_print("Enter number of lines in your text file:\n");
   int stat = my_read_int(0, &lines);
   if (stat < 0) {
         return -1;
   }

   if (argc != 2) {
         my_print("USAGE: ./a.out <filename>\n");
         return -1;
   }
   int N = lines;

   //открытие входного файла

   int input = open(argv[1], O_RDWR);
   if (input == -1) {
         my_print("Can't open file\n");
         return -2;
   }


   //временный файл
   char tmp_name[] = "tmpXXXXXX";
   if (mkstemp(tmp_name) < 0) {
         my_print("Can't create temporary file\n");
         return -3;
   }

   int fd_tmp = open(tmp_name, O_RDWR);
   if (fd_tmp < 0) {
         my_print("Can't open temporary file\n");
         return -2;
   }

   //разделение процессов
   pid_t pid = fork();
   if (pid == -1) {
         perror("Fork error");
         return -1;
```

```
    }
    if (pid != 0) {
          my_print("Child's process was created. Id is ");
          print_int(pid);
          my_print("\n");
    }

    //ожидание выполнения дочернего процесса
    int status = 0;
    wait(&status);
    if (WEXITSTATUS(status)) {
          return -5;
    }

    if (pid == 0) {                    // child process
          my_print("In child\n");
          dup2(input, STDIN_FILENO);
          dup2(fd_tmp, STDOUT_FILENO);
          execl("child", "", NULL);
    } else {                                  // parent process
          int line_in_file = 1;
          my_print("In parent\n");

          char* data = mmap(0, sizeof(char)*100*N,PROT_READ, MAP_SHARED,
fd_tmp, 0);
          int DATA_COUNT = 0;
          while (lines > 0) {
                my_print("line "); print_int(line_in_file);
                //write int numb of line
                my_print(": res1 = ");
                if (data[DATA_COUNT] == '\n') DATA_COUNT++;
                for (DATA_COUNT; data[DATA_COUNT] != ' '; DATA_COUNT++) {
                      write(1, &data[DATA_COUNT], sizeof(char));
                }

                //write float res1
                my_print(", res2 = ");
                for    (DATA_COUNT;    data[DATA_COUNT]    !=    '\n'    &&
data[DATA_COUNT] != '\0'; DATA_COUNT++) {
                      write(1, &data[DATA_COUNT], sizeof(char));
                }
                //write float res2
                my_print("\n");
                line_in_file++;
                lines--;
          }
          int err = munmap(data, sizeof(char)*100*N);
          if (err < 0) {
                my_print("MUNMAP ERROR\n");
          }
    }
    close(input);
    unlink(tmp_name);
```

```
      return 0;
   }
```

iout.h

```
#ifndef IOUT_H
#define IOUT_H

#include <math.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdbool.h>

void my_print(char str[1024]) {
      write(1, str, strlen(str));
}

int my_read_int(int fd, int* a) {
      int res = 0;
      char c;
      read(fd, &c, sizeof(char));
      while(c != '\n' && c != EOF && c != ' ') {
            if (c < '0' || c > '9') {
//                      my_print("Incorrect input\n");
                        return -1;
                  }
            res = res*10 + (c -'0');
            read(fd, &c, sizeof(char));
      }
      *a = res;
      return 0;
}

int read_2_floats(char* data,int* i, int data_size, float* fst, float* sec)
{
      bool read_fst = true, read_sec = false;
      int count = *i;
      while(read_fst || read_sec) {
            bool dot_fnd = false;
            bool mines = false;
            char c;
            float cur = 0;
            double i = 0.1;
            c = data[count];
            while(data[count] != ' ' && data[count] != '\n' && data[count]
!= EOF) {
                  if (c == '-') {
                        mines = true;
                        count++;
                        c = data[count];
                        continue;
                  }
```

```
                  if (((c < '0') || (c > '9')) && c != '.'){
                        return -1;
                  }
                  if (!dot_fnd) {
                        if(c == '.')
                              dot_fnd = true;
                        else {
                              cur = cur * 10 + c - '0';
                        }
                  } else {
                        if(c == '.')
                              return -1;

                        cur = cur + i * (c - '0');
                        i /= 10;
                  }
                  if (count == data_size - 1) {
                        printf("konec data\n");
                  }
                  count++;
                  c = data[count];
            }
            count++;
            if (mines) {
                  cur *= -1;
                  mines = false;
            }
            if (read_fst) {
                  *fst = cur;
                  read_fst = false;
                  read_sec = true;
                  continue;
            }
            if (read_sec) {
                  *sec = cur;
                  read_sec = false;
                  continue;
            }
      }

      *i = count;
      return 0;
}

int read_float(int fd, float* cur){
      bool dot_fnd = false;
      bool mines = false;
      char c;
      *cur = 0;
      double i = 0.1;
      int res = read(fd, &c, sizeof(char));
      while(res > 0){
            if (c == '-') {
```

```
                mines = true;
                res = read(fd, &c, sizeof(char));
                continue;
            }
            if(c == '\n') return -3;
            if(c == ' ')
                break;
            if(((c < '0') || (c > '9')) && c != '.'){
                return -1;
            }
            if (!dot_fnd) {
                if(c == '.')
                    dot_fnd = true;
                else {
                    *cur = *cur * 10 + c - '0';
                }
            } else {
                if(c == '.')
                    return -1;

                *cur = *cur + i * (c - '0');
                i /= 10;
            }
            res = read(fd, &c, sizeof(char));
        }
        if (mines) {
            *cur *= -1;
        }
        if(res == 0)
            return 0;
        return 0;
}

int reverse(int x, int* l) {
        int i = 0;
        while (x > 0) {
            int a = x % 10;
            l[i] = a;
            i++;
            x /= 10;
        }
        return i;
}

void print_num(int a) {
        char* num;
        if (a == 0) num = "0";
        if (a == 1) num = "1";
        if (a == 2) num = "2";
        if (a == 3) num = "3";
        if (a == 4) num = "4";
        if (a == 5) num = "5";
        if (a == 6) num = "6";
```

```
	if (a == 7) num = "7";
	if (a == 8) num = "8";
	if (a == 9) num = "9";
	write(1, num, sizeof(char));
}

void print_int(int x) {
	int data_int[100];
	int data_sz = reverse(x, data_int);
	if (x == 0) {
		print_num(x);
	}
	for (int i = 0; i < data_sz; i++) {
		print_num(data_int[i]);
	}
}

void print_float(float x) {
	if (x < 0) {
		write(1, "-", sizeof(char));
		x *= -1;
	}
	float left;
	float right = modff(x, &left);

	//вывод целой части
	int q = left;
	print_int(q);
	write(1, ".", sizeof(char));

	//вывод дробной часи
	int i = 0;
	while (right > 0) { //0,456
		float new = right*10; //4,56
		right = right*10;
		while (right > 1) right--; // по итогу right = 0,56
		float for_print = new - right; // 4,56 - 0,56
		print_num(for_print); // вывод цифры 4
		//повтор цикла для 0,56 и тд
		i++;
		if (i == 5) {
			break;
		}
	}
}

#endif
```

4. **Выводы**

Данная лабораторная работа знакомит с механизмом межпроцессорного взаимодействия при помощи отображаемых файлов (File Mapping). Данный механизм позволяет отобразить необходимое

количество информации из файла в оперативную память, чтобы несколько процессов могли иметь доступ к ней.

По сравнению с пайпами, данный метод более быстрых за счет отсутствия запросов на чтение и запись, а также благодаря тому, что доступ к отображенным данным происходит за O(1). Как недостаток можно отметить, что дочерние процессы должны знать имя отображаемого файла (или дескриптор), что вызывает некоторые неудобства, а также этот метод требует больше памяти, чем пайпы.