Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу
«Операционные системы»

Студент: Черноус Алексей Тимофеевич
Группа: М8О-209Б-20
Вариант: 1
Преподаватель: Ядров Артем Леонидович
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/HGRaicer/MAI_OS/tree/main/lab1

Постановка задачи

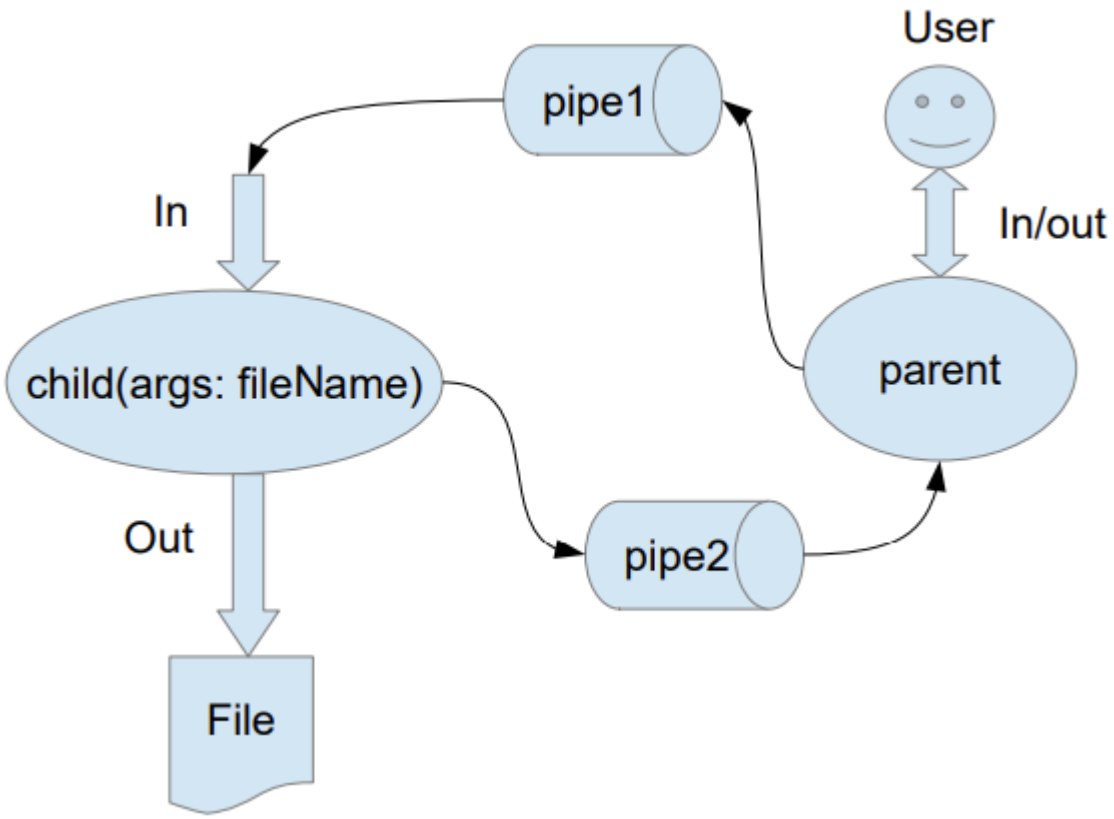
Цель работы

Приобретение практических навыков в:

1. Управление процессами в ОС
2. Обеспечение обмена данных между процессами посредством каналов

Задание

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.

1 вариант) Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип int. Количество чисел может быть произвольным.

Общие сведения о программе

Запускаем программу через Makefile. Также используется заголовочные файлы: unistd.h, stdio.h , stdlib.h, ctype.h. В программе используются следующие системные вызовы:

1. **fork** - создает копию текущего процесса, который является дочерним процессом для текущего процесса
2. **pipe** - создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.
3. **close** - закрывает файл.
4. **read** - читает количество байт(третий аргумент) из файла с файловым дескриптором(первый аргумент) в область памяти(второй агрумент).
5. **write -** записывает в файл с файловым дескриптором(первый аргумент) из области памяти(второй аргумент) количество байт(третий аргумент).
6. **perror –** вывод сообщения об ошибке.
7. **dup2 –** создание копии файлового дескриптора.

Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, pipe, dup2, close, read, write.
2. Написать программу, которая будет работать с 2-мя процессами: родительским и дочерним, процессы связываются между собой при помощи pipe-ов.

   Для передачи данных между процессорами использовать pipe.

Исходный код

child.c

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#define MAX_INPUT 512
#define MAX_NUMBERS 100

int check(char ch){
  if ((ch-'0' >= 0) && (ch-'0'<=9)){
    return 1;
  }
  return 0;
}

int main() {

  char filename[MAX_INPUT];
  char data[MAX_INPUT];
  char buf;
  int j = 0;
```

```
int flag = 0;
int i = 0;
int result = 0;

while (read(STDIN_FILENO, &buf, 1) > 0){
  if (buf != '|'){
    if (flag != 1){
    filename[j++] = buf;}
    else{
      data[i++] = buf;
    }
  } else {
    flag = 1;
  }
}

data[i] = '\0';

char *buff = malloc(sizeof(char) * 10);

int jj = 0;
int tmp = 0;
for (int k =0; k<i; ++k){
  if (data[k] != '\0'){
    while (check(data[k])==1){
      buff[jj++] = data[k];
      ++k;
    }
    sscanf(buff, "%d", &tmp);
```

```
      result += tmp;
      jj = 0;
      while(buff[jj] != '\0'){
        buff[jj++] = ' ';
      }
      jj = 0;
    }
    else{
      break;
    }
  }
  free(buff);
  filename[j]='\0';
  FILE *file = fopen(filename, "w");
  if (file == NULL) {
    perror("Ошибка при открытии файла");
    return 1;
  }

  fprintf(file, "%d\n", result);

  fclose(file);

  printf("Данные успешно записаны в файл: %s\n", filename);

  return 0;
}
```

parent.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define MAX_INPUT 512

int main(void) {
  char filepath[MAX_INPUT];
  char line[MAX_INPUT];

  printf("Путь для файла вывода:\n");
  scanf("%s", filepath);

  printf("Введите числа, разделенные пробелами:\n");

  //Очищаем буфер ввода перед считыванием новой строки
  int c;
  while ((c = getchar()) != '\n' && c != EOF);

  if (fgets(line, sizeof(line), stdin) == NULL) {
    fprintf(stderr, "Ошибка при чтении ввода\n");
    return 1;
  }

  int pipefd[2];
  if (pipe(pipefd) == -1) {
    perror("pipe");
```

```
    exit(1);
  }

  switch (fork()) {
    case -1:perror("fork");
      exit(1);
    case 0:close(pipefd[1]);
      dup2(pipefd[0], STDIN_FILENO);
      close(pipefd[0]);
      char *args[] = {"./build/child", NULL};
      execv(args[0], args);

      perror("execv");
      exit(1);

    default:close(pipefd[0]);

      write(pipefd[1], filepath, MAX_INPUT);
      write(pipefd[1], "|", 1);
      write(pipefd[1], line, MAX_INPUT);

      close(pipefd[1]);

      int status = 0;

      wait(&status);
  }

  return 0;
```

```
}
```

Демонстрация работы программы

```
rai@rai-laptop:~/cods/os/lab1$ make
mkdir build
gcc -o build/parent src/parent.c
gcc -o build/child src/child.c
./build/parent
Путь для файла вывода:
gg
Введите числа, разделенные пробелами:
22 33 44 55
Данные успешно записаны в файл: gg
```

Выводы

Существуют специальные системные вызовы(fork) для создания процессов, также существуют специальные каналы pipe, которые позволяют связать процессы и обмениваться данными при помощи этих pipe-ов. При использовании fork важно помнить, что фактически создается копию вашего текущего процесса и неправильная работа может привести к неожиданным результатам и последствиям, однако создание процессов очень удобно, когда вам нужно выполнять несколько действий параллельно. Также у каждого процесса есть свой id, по которому его можно определить. Также важно работать с чтением и записью из канала, помня что read, write возвращает количество успешно считанных/записанных байт и оно не обязательно равно тому значению, которое вы указали. Также важно не забывать закрывать pipe после завершения работы.