Московский авиационный институт

(Национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 2 по курсу «Операционные системы»

Студент: Саженов Константин Станиславович

Группа: 8О-208

Преподаватель: Е. С. Миронов

Вариант: 7

Дата:

Оценка:

Москва, 2020

Лабораторная работа №2

1 Описание

Данная лабораторная работа будет выполняться в ОС Unix.

Процесс – абстракция, описывающая выполняющуюся программу. В моем случае, имеется два процесса: процесс-родитель и процесс-ребенок. Взаимодействие между процессами будет осуществляться с помощью неименнованного канала(pipe1). Процесс-родитель осуществляет чтение названия файла из стандартного ввода, из которого процесс-ребенок будет осуществлять чтение и выводит на экран все, что придет ему через pipe1 от процесса-ребенка. Процесс-ребенок перенаправляет поток ввода на файл и поток вывода на pipe1 и считает сумму чисел, введенных в файл, а затем выводит результат на стандартный вывод.
Для корректной работы программы, использующей больше одного процесса, необходимо решить три проблемы: передача информации между процессами, обеспечение совместной работы без создания взаимных помех, определение правильной последовательности работы процессов. Как сказано выше, информация передается по каналам. Совместная работа обеспечивается благодаря системным вызовам read, write, close, которые используют файловые дескрипторы в определенной последовательности для корректной передачи полученных данных по каналу. Правильная последовательность работы процессов осуществлется при помощи wait(что является сокращением к waitpid). Этот системный вызов приостанавливает выполнение текущего процесса до тех пор, пока не завершится дочерний процесс.
Использованные системные вызовы:
pid_t fork(void); – создает дочерний процесс. Если возвращает 0, то созданный текущий процесс – ребенок, если >0, то – родитель, если <0, то – ошибка(и текущий процесс – родитель).
exit(int status); – выходит из процесса с заданным статусом.
pid_t wait(int *status); – он же **pid_t waitpid(pid_t pid, int *status, int options)**, где pid=0, options=0. Приостаналивает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик.
int pipe(int *fd); – предоставляет средства передачи данных между двумя процессами(неименованный канал).
int close(int fd); – закрывает файловый дескриптор.
int read(int fd, void *buffer, int nbyte); – читает nbyte байтов из файлового дескриптора fd в буффер buffer.
int putchar(char c); – записывает символ с в стандартный поток вывода, а возаращает записанный символ или EOF, в случае ошибки.

2 Исходный код

```
//parent.c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

#include <sys/wait.h>
#include <fcntl.h>


int main(int argc, char* argv[]) {

    char *filename;
    size_t len;

    printf("Enter a filename with tests: ");
    if(getline(&filename, &len, stdin) == -1){
        perror("getline");
    }

    filename[strlen(filename) - 1] = '\0';

    int fds[2];
    if(pipe(fds) != 0){
        perror("pipe");
    }

    int rfd = fds[0];
    int wfd = fds[1];

    if(access(filename, R_OK) == -1){
        fprintf(stderr, "File not exists\n");
        exit(EXIT_FAILURE);
    }
    int file = open(filename, O_RDONLY);
    if (file == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();

    if(pid == 0){
```

```
        if (dup2(wfd, fileno(stdout)) == -1) {
            exit(1);
        }
        close(wfd);

        if (dup2(file, fileno(stdin)) == -1) {
            exit(1);
        }
        execl("2_lab_child", "2_lab_child", NULL);
        perror("execl");
        exit(EXIT_FAILURE);
    }
    else if(pid == -1){
        perror("fork");
    }

    close(fds[1]);
    close(file);

    char c;
    while(read(rfd, &c, sizeof(char)) > 0){
        putchar(c);
    }

    int status;
    if(wait(&status) == -1){
        perror("wait");
    }
    if(!WIFEXITED(status) || (WIFEXITED(status) &&
(WEXITSTATUS(status)) != 0)){
        fprintf(stderr, "Some error occurred in child process\n");
        return 1;
    }
    return 0;
}

//child.c
//
// Created by sakost on 23.09.2020.
//
#include <stdio.h>
#include <unistd.h>

int main(){
    double a;
    char c;
```

```
    double res = 0;

    while(scanf("%lf%c", &a, &c) != EOF) {
        res += a;
        if(c == '\n') {
            printf("%lf\n", res);
            res = 0.;
            continue;
        }
    }

    printf("%lf\n", res);
    close(STDOUT_FILENO);
    return 0;
}
```

3 Консоль

```
sakost at sakost-pc ▶ 21:30 ▶ 🐧 os/2 lab
 ▶ ./2_lab
Enter a filename with tests: test.txt
3
7
100
200

sakost at sakost-pc ▶ 21:31 ▶ 🐧 os/2 lab
 ▶ cat test.txt
1 2
3 4
56 44
101 99

sakost at sakost-pc ▶ 21:31 ▶ 🐧 os/2 lab
 ▶ ./2_lab
Enter a filename with tests: esrse
Some error occurred in child process

sakost at sakost-pc ▶ 21:31 ▶ 🐧 os/2 lab
1 ↵ ▶
```

4 Выводы

Данная лабораторная работа помогла мне разобраться с тем, как следует работать с неименованными каналами для межпроцессорного взаимодействия, как переопределять потоки ввода/вывода, а также как в общем случае следует работать с файловыми дескрипторами и системными вызовами для управления ими.

Хотя программа, с точки зрения практичности, является бесполезной, она позволяет заглянуть в то, каким образом устроено межпроцессорное взаимодействие.

Редко встретишь программу, которая работает в однопроцессорном режиме. Поэтому нам, как программистам, необходимо уметь работать с ними. Очевидно, что в сложных программах используется большее количество неименованных каналов, чем один. Находясь в ситуации нравственного выбора(писать хороший код или не очень хороший), важно сделать правильное архитектурное решение.