МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №2

по курсу операционные системы I семестр, 2021/22 уч. год

Студент *Сикорский Александр Александрович, группа М8О-208Б-20*
Преподаватель *Миронов Евгений Сергеевич*
Вариант *21*
Оценка
Дата
Подпись

Содержание

Репозиторий

`https://github.com/sikorskii/os/tree/master/lab2`

Постановка задачи

Цель работы:

Приобретение практических навыков в:

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

Задание: Вариант 21:

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 два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

Общие сведения о программе

Исходный код лежит в 2 файлах:

1. src/main.cpp: код процесса - родителя.
2. src/child1.cpp: код процессов - детей. У обоих детей идентичный функционал.

Несмотря на то, что файлы имеют расширение .cpp, в них по сути написан только "сишный" код, программа собирается с помощью CMakeList и сразу же с санитайзером. Нужно получить два исполняемых файла: один для родителя и один для ребенка.

Программа использует следующие системные вызовы:

- fork - создает новый дочерний процесс, получающий свою область памяти и работающий в ней. Изначально является копией родителя.

- pipe - создает канал для обмена данными между объектами. Как я понимаю, предоставляет некую общую область в памяти, куда можно писать и откуда можно читать. Вызов открывает 2 дескриптора на чтение и запись соответственно. Ненужные можно закрыть.
- fflush - когда программа пишет в файл или поток, она на самом деле пишет в буфер. Это нужно в том числе для оптимизации. Чтобы явно записать что-то в физический файл, нужно выполнить этот вызов. Этот же вызов приводит к очистке буфера потока.
- close - закрывает открытый файловый дескриптор, например, один или оба конца pipe'a.
- read - считывает заданное количество байт из заданного дескриптором файла в заданную область памяти.
- write - аналогично read, но пишет в файл, заданный дескриптором.

Общий метод и алгоритм решения

Сначала родитель открывает два pipe'a для первого потомка и для второго, затем считывает имена файлов, которые будут открывать потомки. После этого ему нужно как-то передать дескрипторы в дочерние процессы. Так как они запускаются с помощью execl, они просто так не видят их. Для этого в родителе работает некий механизм а-ля сериализация, при котором число приводится к строке, которую уже можно передать аргументом при запуске дочернего процесса. Это происходит для двух дескрипторов для каждого потомка. Соответственно на другой стороне происходит обратное преобразование строки к числу.

Родитель освобождает память на указателях имен файлов и полученных для дескрипторов строк и начинает выполнять задание варианта. Он считывает строки, проверяет их длину на четность-нечетность и пишет их в соответствующие первому и второму потомкам файловые дескрипторы. Пустые строки никуда не пишутся. EOF на входе прекращает работу программы.

Ребенок собирает свой массив дескрипторов из пришедших к нему строк. Также он уже получил аргументом имя файла, куда ему положено писать. У ребенка и у родителя есть буфер в 50 знаков, куда они сохраняют строки. Функция ребенка разворачивает пришедшую строку и записывает её в указанный файл, который, само собой, после закрывается.

Исходный код

main.cpp

```cpp
#include "unistd.h"
#include <cstdio>
#include <cstring>
#include <cstdlib>

// 0 - reading
// 1 - writing
//MAX STRING LENGTH IS 50

int main() {

    int pipe1[2];
    int pipe2[2];

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
        perror("Pipe error!");
    }

    char buf[50];

    char *filename1;
    char *filename2;

    printf("Enter file1 name\n");
    //scanf("%s", buf);
    filename1 = fgets (buf, sizeof(buf) - 1, stdin);
    asprintf(&filename1, "%s", buf);

    printf("Enter file2 name\n");
    //scanf("%s", buf);
    filename2 = fgets (buf, sizeof(buf) - 1, stdin);
    asprintf(&filename2, "%s", buf);

    pid_t child1_pid, child2_pid;

    if (pipe(pipe1) == -1) {
        perror("Pipe error!");
    }

    char *toc1;
    char *toc2;
    asprintf(&toc1, "%d", pipe1[0]);
```

```
asprintf(&toc2, "%d", pipe1[1]);


child1_pid = fork();

if (child1_pid == -1) {
    printf("fork error!\n");
}

else if (child1_pid == 0) { //child1

    printf("[%d] It's child1\n", getpid());
    fflush(stdout);
    execl("child1.out", "child1", filename1, toc1, toc2, NULL);
    //execution of child1's program begins here

}

else { //parent

    printf("[%d] It's parent. Child id: %d\n", getpid(), child1_pid);
    fflush(stdout);

    if (pipe(pipe2) == -1) {
        perror("Pipe error!");
    }

    char *toc11;
    char *toc22;
    asprintf(&toc11, "%d", pipe2[0]);
    asprintf(&toc22, "%d", pipe2[1]);

    child2_pid = fork();

    if (child2_pid == -1) { //error
        printf("fork error!\n");
    }

    else if (child2_pid == 0) { //child2

        printf("[%d] It's child2\n", getpid());
        fflush(stdout);
```

```
            execl("child1.out", "child2", filename2, toc11, toc22, NULL);
            //execution of child2's program begins here

        }

        //parent code below

        free(filename1);
        free(filename2);
        free(toc1);
        free(toc2);
        free(toc11);
        free(toc22);

        char *str;

        while (true) {
            char c[50];
            str = fgets (c, sizeof(c), stdin);

            if(strlen(c) == 1)
                continue;

            if (str == nullptr)
                break;

            close(pipe1[0]);
            close(pipe2[0]);

            if ((strlen(c) - 1) % 2 == 0) {
                write(pipe2[1], c, strlen(c) + 1);
            }
            else {
                write(pipe1[1], c, strlen(c) + 1);
            }

        }

        return 0;
    }
}
```

child1.cpp

```cpp
//
// Created by aldes on 19.09.2021.
//

#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
// 0 - reading
// 1 - writing
void reverse(char *str) {
    int i = 0;
    int j = (int)strlen(str) - 1;

    while(i < j) {
        char temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++;
        j--;
    }
}

int main(int argc, char* argv[]) {
    printf("\ni am child and i will write in file %s\n", argv[1]);

    int fd[2];
    fd[0] = (int) strtol(argv[2], nullptr, 10);
    fd[1] = (int) strtol(argv[3], nullptr, 10);

    FILE *fp = fopen(argv[1], "w");
    fprintf(fp, "child been here\n");

    close(fd[1]);
    char buf[50];

    while(read(fd[0], buf, sizeof(buf)) != 0) {
        reverse(buf);
        //printf("Recieved : %s\n", buf);
        fprintf(fp, "Received string: %s\n", buf);
        fflush(fp);
```

```
    }

    fclose(fp);
    return 0;
}
```

Демонстрация работы программы

Ниже приведен пример работы программы.
>"string" обозначает пользовательский ввод.

output.txt

```
Enter file1 name
>aboba
Enter file2 name
>cringe
[3247] Its parent. Child id: 3249
[3249] Its child1
[3250] Its child2

i am child and i will write in file aboba


i am child and i will write in file cringe

>anfd
>dfg
>xczv
>sfakglfdjg
>wqreq
>234
>24524624
>dgfdsg
>dcxzv
```

файл с выводом aboba

```
child been here
Received string:
gfd
Received string:
qerqw
Received string:
432
Received string:
vzxcd
```

файл с выводом cringe

```
child been here
Received string:
dfna
Received string:
vzcx
Received string:
gjdflgkafs
Received string:
42642542
Received string:
gsdfgd
```

Выводы

В ходе работы я изучил несколько системных вызовов, научился создавать новые процессы и начал изучать межпроцессное взаимодействие. В работе с пайпами есть некоторое удобство, что программисту не надо задумываться над синхронизацией процессов. Вызовы read и write блокируют процессы до того, как в трубу что-то напишут или что-то оттуда считают. Новый процесс я создавал с помощью fork, а его образ заменял на свою программу с помощью execl. Можно было просто написать полный код дочернего процесса в if'е после fork'а, но так неинтересно. Да, такая работа с помощью execl немного сложнее, но интереснее и для первой лабораторной кажется мне некоторым вызовом.