

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Взаимодействие между процессами

Группа: М80-210Б-22

Студент: Бонокин Д.С.

Вариант: 11

Преподаватель: Соколов А.А.

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

Постановка задачи

Цель работы:

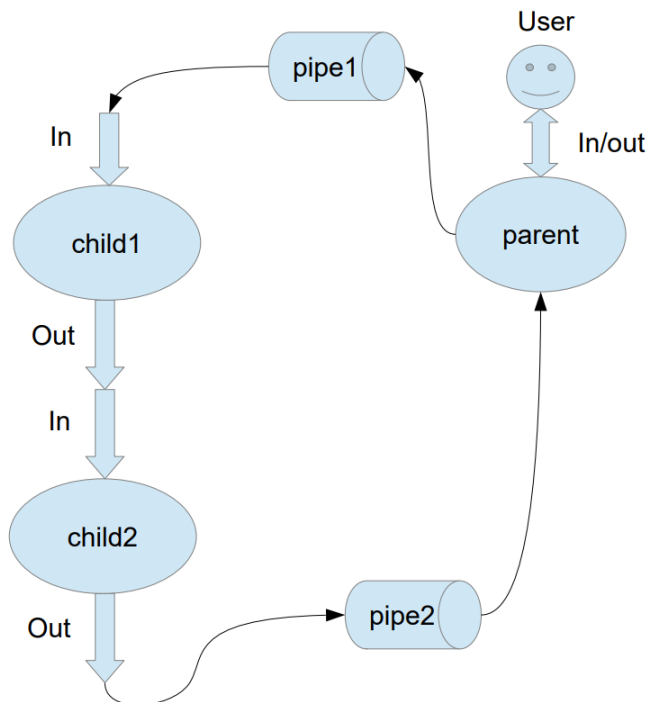
Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данными 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.

Вариант 11.



Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «_».

Общие сведения о программе

Программа компилируется при помощи утилиты Make и запускается путем запуска ./parent. Также используются заголовочные файлы: iostream,unistd.h, ctype. В программе используются следующие системные вызовы:

- `int pipe(int *fd);` – создаёт канал (пайп).
- `pid_t fork(void);` – создаёт дочерний процесс.
- `int dup2(int oldfd, int newfd);` – делает newfd копией дескриптора oldfd, закрывая newfd, если требуется.
- `int execl(const char *path, const char *arg, ...);` – заменяет текущий образ процесса новым образом процесса.
- `int close(int fd);` – закрывает файловый дескриптор.
- `size_t write(int fd, const void *buf, size_t count);` – записывает до count байтов из буфера buf в файл, на который ссылается файловый дескриптор fd.
- `size_t read(int fd, void *buf, size_t count);` – пытается записать count байтов файлового дескриптора fd в буфер, адрес которого начинается с buf.

Общий метод и алгоритм решения

Создал три канала для связи дочерних процессов и родительского с дочерними с помощью pipe(). Далее создал два дочерних процесса с помощью fork() и вызвал скомпилированные child1.cpp и child2.cpp с помощью execl(). В родительском процессе читал символы, которые пишет пользователь и сначала посылал в child1. Первый дочерний процесс с помощью toupper() переводил символы в верхний регистр

и посылал их child2 через канал. Второй дочерний процесс заменял пробел на «_» и посылал обратно родительскому процессу, который уже выводил их на стандартный Вывод.

Код программы

main.cpp

```
#include <unistd.h>
#include <iostream>
#include <cctype>

pid_t create_process() {
    pid_t pid = fork();
    if (pid == -1) {
        perror("fork error!\n");
        exit(-1);
    }
    return pid;
}

void create_pipe(int* pipe_fd) {
    if (pipe(pipe_fd) == -1) {
        perror("pipe error!\n");
        exit(-1);
    }
}

void dup_fd(int oldfd, int newfd) {
    if (dup2(oldfd, newfd) == -1) {
        perror("dup2 error!\n");
        exit(-1);
    }
}

void new_program(const char *name){
    if (execl(name, name, NULL) == -1){
        perror("execl error!\n");
        exit(-1);
    }
}

int main() {
    int pipe1_fd[2], pipe2_fd[2];

    create_pipe(pipe1_fd);
    create_pipe(pipe2_fd);

    pid_t child1 = create_process();

    if (child1 == 0) {
        close(pipe1_fd[1]);
        close(pipe2_fd[0]);

        int pipe3_fd[2];
        create_pipe(pipe3_fd);

        pid_t child2 = create_process();

        if (child2 == 0) {
            close(pipe3_fd[0]);
            close(pipe2_fd[1]);
```

```

    dup_fd(pipe1_fd[0], STDIN_FILENO);
    dup_fd(pipe3_fd[1], STDOUT_FILENO);

    new_program("child2");

    close(pipe3_fd[1]);
    close(pipe1_fd[0]);
} else {
    close(pipe1_fd[0]);
    close(pipe3_fd[1]);

    dup_fd(pipe3_fd[0], STDIN_FILENO);
    dup_fd(pipe2_fd[1], STDOUT_FILENO);

    new_program("child1");

    close(pipe1_fd[0]);
    close(pipe2_fd[1]);
}
} else {
    close(pipe1_fd[0]);
    close(pipe2_fd[1]);

    char c = getchar();
    char new_c;
    while (c != EOF) {
        write(pipe1_fd[1], &c, sizeof(c));
        read(pipe2_fd[0], &new_c, sizeof(c));
        putchar(new_c);
        c = getchar();
    }

    close(pipe1_fd[1]);
    close(pipe2_fd[0]);
}

return 0;
}

```

child1.cpp

```

#include <iostream>
#include <unistd.h>

int main() {
    char c;
    while (read(STDIN_FILENO, &c, sizeof(c)) != -1) {
        c = toupper(c);
        write(STDOUT_FILENO, &c, sizeof(c));
    }
    close(STDIN_FILENO);
    close(STDOUT_FILENO);
}

```

child2.cpp

```

#include <iostream>
#include <unistd.h>

int main() {
    char c;
    while (read(STDIN_FILENO, &c, sizeof(c)) != -1) {
        if (c == ' ') {
            c = '_';
        }
        write(STDOUT_FILENO, &c, sizeof(c));
    }
}

```

```
close(STDIN_FILENO);  
close(STDOUT_FILENO);  
}
```

Протокол работы программы

Тестирование:

danil@danil-1-2:~/lab/lab1/build\$./main

cvbs ddsq

CVBS__DDSQ

derftv DDc fvt

DERFTV_DDC_FVT

Вывод

В ходе лабораторной работы я написал программу, которая делает системные вызовы. Я научился работать с каналами и процессами.