Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу

«Операционные системы»

Процессы операционных систем

Студент: Яруллин А.Р.
Группа: М8О–201Б–21
Вариант: 13
Преподаватель: Миронов Е.С.
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022.

Постановка задачи

Цель работы

Целью является приобретение практических навыков в:

- Управление процессами в ОС
- 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.

Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: stdio.h, stdbool.h, stdlib.h, string.h, unistd.h, errorlib.h. В программе используются следующие системные вызовы:

1. **pid_t fork()** - создание дочернего процесса, возвращает -1 при ошибке создания дочернего процесса, 0 если процесс является дочерним, и pid если процесс является родительским.
2. **int execlp(const char *file, const char *arg, ...)** – заменяет текущий образ процесса новым образом процесса, с аргументами arg.
3. **pid_t waitpid(pid_t pid, int *status, int options)** - Ожидание завершения дочернего процесса
4. **void exit(int status)** - завершения выполнения процесса и возвращение статуса
5. **int pipe(int pipefd[2])** - создание неименованного канала для передачи данных между процессами
6. **int dup2(int oldfd, int newfd)** - переназначение файлового дескриптора
7. **int close(int fd)** - закрыть файл

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, pipe, execlp, dup2, waitpid.
2. Написать функцию oerror, для вывода сообщений об ошибках.
3. Написать программу child1 для перевода текста в нижний регистр
4. Написать программу child2 для замены пробельных символов на нижнее подчеркивание
5. Написать программу создающую два дочерних процесса заменяемые child1 и child2 и соединить каналами pipe1 и pipe2

Основные файлы программы

main.c:

```
#include "parent.h"
#include <stdio.h>
#include <stdlib.h>
#include <vector>

int main(void) {
    std::vector <std::string> input;

    std::string s;
    while (getline(std::cin, s)) {
        input.push_back(std::move(s));
    }
    std::vector <std::string> output = ParentRoutine(input);

    for (const auto &res : output){
        std::cout << res << std::endl;
    }
    return 0;
}
```

child1.c

```
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include <iostream>
#include"errorlib.h"

int main() {
    char c;
    while(read(0, &c, 1) > 0) {
        if(c >= 'A' && c <= 'Z') {
            c = std::tolower(c);
        }
        if(write(1, &c, 1) == -1) {

            Oerror("can't write the pipe 3 by child1", -1);
        }
    }
    return 0;
}
```

child2.c

```
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include"errorlib.h"

int main() {
    char c;
    while(read(0, &c, 1) > 0) {
        if(c == ' ') {
            c = '_';
        }
        if(write(1, &c, 1) == -1) {
            Oerror("can't write the pipe 2 by child2", -1);
```

```
        }
    }
    return 0;
}
```

errorlib.c

```
#include "errorlib.h"
int Oerror(const char * error, int id) {
    write(STDERR_FILENO, error, strlen(error));
    exit(id);
}
```

parent.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <string.h>
#include "parent.h"
#include "errorlib.h"
#include <vector>
#include <iostream>
#include <fstream>
#include <sys/wait.h>

std::vector<std::string> ParentRoutine(const std::vector<std::string> &input)
{
    const char endLine = '\n';
    char readChar;
    std::string str;
    std::vector<std::string> output;
    int fd1[2], fd2[2], fd3[2], errc;
    if (pipe(fd1) == -1 ||
        pipe(fd2) == -1 ||
```

```
    pipe(fd3) == -1)
{
    Oerror("can't create a pipe\n",-1);
}
int pid1, pid2;
if ((pid1 = fork()) == 0)
{
    if (dup2(fd1[0], STDIN_FILENO) == -1 ||
        dup2(fd3[1], STDOUT_FILENO) == -1 ||
        close(fd1[1]) == -1 ||
        close(fd3[0]) == -1 ||
        close(fd2[1]) == -1 ||
        close(fd2[0]))
    {
        Oerror("Error init a pipe\n",-1);
    }

    if (execlp("./child1", "child1", NULL) == -1)
    {
        Oerror("can't open file child1:\n",-1);
    }
}
if (pid1 > 0 && (pid2 = fork()) == 0)
{
    if (dup2(fd2[1], STDOUT_FILENO) == -1 ||
        dup2(fd3[0], STDIN_FILENO) == -1 ||
        close(fd2[0]) == -1 ||
        close(fd3[1]) == -1 ||
        close(fd1[1]) == -1 ||
        close(fd1[0]) == -1)
    {
        Oerror("Error init a pipe\n",-1);
```

```
    }
    if (execlp("./child2", "child2", NULL) == -1)
    {
        Oerror("can't open file child2:\n",-1);
    }
}
if (pid1 == -1 || pid2 == -1)
{
    Oerror("can't create processes child:\n",-1);
}
if (pid1 != 0 && pid2 != 0)
{
    if (close(fd1[0]) == -1 ||
        close(fd2[1]) == -1)
    {
        Oerror("can't close pipe read:\n",-1);
    }
    for (const auto &s : input)
    {
        if (write(fd1[1], s.c_str(), s.size()) == -1)
        {
            Oerror("can't write in pipe 1:\n",-1);
        }
        if (write(fd1[1], &endLine, 1) == -1)
        {
            Oerror("can't write in pipe 1:\n",-1);
        }
        str.clear();
        while ((errc = read(fd2[0], &readChar, 1)))
        {
            if (errc == -1)
            {
```

```
                Oerror("can't read from pipe 2:\n",-1);
            }
            if (readChar == '\n')
            {
                break;
            }

            str += readChar;
        }

        output.push_back(std::move(str));
    }
    close(fd1[1]);
    close(fd2[0]);
    close(fd3[1]);
    close(fd3[0]);
    wait(NULL);
  }
  return output;
}
```

errorlib.h

```
#ifndef ERRORLIB_H
#define ERRORLIB_H
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
int Oerror(const char * error, int id);
#endif
```

parent.h

```
#ifndef PARENT_H
#define PARENT_H
```

```
#include <sys/wait.h>
#include <iostream>
#include <fstream>
#include <vector>

std::vector<std::string> ParentRoutine(const std::vector<std::string> &input);

#endif //PARENT_H
```

Пример работы

Input

```
TThE quick browNNN
f0x jumps .OvEr. the l@zy dog
```

Output

```
tthe_quick_brownnn________
__________
f0x_jumps_.over._the_l@zy_dog
```

Вывод

В результате данной лабораторной работы были изучены основные методы работы с процессами в ОС linux. В данной работе я научился создавать процессы, работать с родительскими и дочерними процессами, передавать между ними данные с помощью каналов. Полученные знания пригодятся не только при выполнении лабораторных работ, но и в дальнейшей промышленной разработке