Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу
«Операционные системы»

Межпроцессное взаимодействия

Студент: Недосеков Иван Дмитриевич
Группа: М8О-206Б-19
Вариант: 9
Преподаватель: Соколов Андрей Алексеевич
Дата: _____
Оценка: _____
Подпись: _____

Москва, 2020

Постановка Задачи

Цель работы

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.
9 вариант: в файле записаны команды вида:«число число число <endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float.

Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: *unistd.h, fcntl.h, sys/types.h, sys/stat.h, stdio.h, string.h, stdlib.h*. В программе используются следующие системные вызовы:

1. **pipe** — создает неименованный конвеер, в него необходимо передать массив как минимум из двух int. В массив записываются дескрипторы на 0-ую позицию на получения информации, на 1-ую для отправки.
2. **read** — читает по дескриптору заданное количество байт в буфер.
3. **write** — записывает по дескриптору указанное количество байт из буфера.
4. **fork** — создает процесс копию , ребенку (копии) возвращается 0, а отцу PID ребенка.
5. **open** — открывает файл по имени с определенными модификаторами доступа, и возвращает файловый дескриптор.
6. **close** — закрывает файловый дескриптор.
7. **dup2** — заменяет один файловый дескриптор другим.
8. **exit** — завершение программы с указанным кодом.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

- Изучить принципы работы pipe, read, write, fork, open, close, dup2, exit.
- Написать парсер стандартного ввода.
- Написать логику работы отца и ребенка.
- Проверить программу на тестах.

Основные файлы программы

```c
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#define SUCCESS 0
#define FAIL -1
#define BUF_SIZE 256
```

```c
#define MAX_STRING  256
#define NUMERAL_SYSTEM 10


int IsSep( const char ch ){
    return ch == ' ' || ch == '\n' || ch == '\t';
}
int IsEndStr( const char ch ){
    return ch == '\0' || ch == EOF;
}
int IsNum( const char ch ){
    return '0'<= ch && ch <= '9';
}

int  StrToFloatTok( const char* str, float* floatTok, int countTok ){
↪  //разбиение строки на flot'ы
    int i = 0;
    for ( int tok = 0; tok < countTok; ++tok ){
        int sign = 1;
        int intPart = 0;
        while( IsSep(str[i]) ){
            ++i;
        }
        if ( str[i] == '-' ){   //берем знак если минус
            sign = -1;
            ++i;
        }
        while( IsSep(str[i]) ){
            ++i;
        }
        while( IsNum( str[i] ) ){      //дошли до целой части
            intPart = ( intPart * NUMERAL_SYSTEM ) + ( str[i] - '0' );
            ++i;
        }
        if ( IsSep(str[i]) ){       // проверяем что прервало число запятая или
        ↪  пробел и тд.
            floatTok[tok] = sign * intPart;
            continue;
        }
        if ( IsEndStr(str[i]) ){
            floatTok[tok] = sign * intPart;
            return tok + 1;
        }
        ++i;
        float floatPart = 0;    //считываем число после запятой
```

```
        int floatPartSize = 1;
        while( IsNum(str[i]) ){
            floatPart = floatPart + ( ( str[i] - '0' ) * pow( NUMERAL_SYSTEM ,
            ↪  -1 *  floatPartSize ) );
            ++floatPartSize;
            ++i;
        }
        floatTok[tok] = sign * (intPart + floatPart);
        if ( IsEndStr(str[i]) ){
            return tok + 1;
        }
    }
    return countTok;
}

int ReadString( int fd, char* str ){    // чтение строки с потока
    char ch;
    int readCh = 0;

    if ( read( fd, &ch, sizeof( char ) ) < 1 ){
        return EOF;
    }
    while ( IsSep(ch) ){
        if ( read( fd, &ch, sizeof( char ) ) < 1){
            return EOF;
        }
    }

    while( readCh < MAX_STRING ){
        str[readCh] = ch;
        if ( read( fd, &ch, sizeof( char ) ) < 1 ){
            break;
        }
        if ( ch == '\n' ){
            break;
        }
        ++readCh;
    }

    ++readCh;
    if ( readCh >= MAX_STRING ){
        perror("Oversize string");
        exit( FAIL );
    }
    str[readCh] = '\0';
```

```
    return readCh;
}

int ChildWork(){
    float floats[3];
    char command[BUF_SIZE];
    while( ReadString( STDIN_FILENO, command ) > 0 ){
        if ( StrToFloatTok( command, floats, 3 ) < 3 ){
            perror("Nonvalid command");
            perror(command);
            return FAIL;
        }
        for( int i = 1; i < 3; ++i ){
            float res = floats[0] / floats[i];
            if ( isinff( res ) ){
                perror("Division by zero");
                return FAIL;
            }
            write( STDOUT_FILENO, &res, sizeof( float )  );
        }
    }
    return SUCCESS;
}

void ParentWork( int childFD ){
    float toPrint;
    char pr[BUF_SIZE];
    while( read( childFD, &toPrint, sizeof( float ) ) > 0 ){
        sprintf( pr,"Received from child %4.4f \n", toPrint );
        write( STDOUT_FILENO, pr , strlen( pr ) * sizeof( char ) );
    }
}

int main(){
    int fd[2], pipe1[2];

    if ( pipe(fd) != 0 ){
        return FAIL;
    }
    if ( pipe( pipe1 ) != 0 ){
        return FAIL;
    }

    int id = fork();
```

```
if ( id < 0 ){
    perror("Fork error");
  return FAIL;

}else if( id == 0){ // Программа ребенка

    close( fd[1] );
    close( pipe1[0] );
    fflush(stdout);

    char fileName[BUF_SIZE];
    ReadString( fd[0], fileName );

    int  file = open( fileName, O_RDONLY );
    if ( file < 0 ){
        perror("Child can't open file");
        exit( FAIL );
    }

    close( fd[0] );
    close( STDIN_FILENO );
    dup2( file, STDIN_FILENO ); // перенаправляем стандартный поток
    ↪ ввода-вывода
    close( STDOUT_FILENO );
    dup2( pipe1[1], STDOUT_FILENO );

    if ( ChildWork() < 0 ){
        close( pipe1[1] );
        exit( FAIL );
    }

    close( pipe1[1] );
    exit( SUCCESS );

}else{ //Программа родителя

    close( pipe1[1] ); // закрываем неиспользуемые дескрипторы конвееров
    close( fd[0] );
    fflush(stdout);    // и сбрасываем буффер вывода

    char fileName[BUF_SIZE];
    if ( ReadString( STDIN_FILENO, fileName ) < 1 ){
        perror("Need file with commands");
        exit( FAIL );
    }
```

```
        int lenName = strlen( fileName );
        write( fd[1], fileName, sizeof( char ) * lenName );
        close( fd[1] );

        ParentWork( pipe1[0] );

        close( pipe1[0] );
        exit( SUCCESS );
    }
}
```

Пример работы

```
mx$ cat test.txt
91 1 4.0
-78 2 3
-0.2 1 0
mx$ ./build/os_lab1
test.txt
Received from child 91.0000
Received from child 22.7500
Received from child -39.0000
Received from child -26.0000
Received from child -0.2000
Division by zero: Success
```

GitHub

Ссылка на репозиторий GitHub с проектом:
https://github.com/GrozniyToaster/os_lab_02

Вывод

В лабараторной работе мы изучили основы межпроцессорного взаимодействия через конвейеры, это полезный инструмент, так как в современном мире делается упор на разработку многопроцессорных и многопоточных программ. Также изучили основы разработки много процессорных программ, что в перспективе поможет разрабатывать большие проекты на работе.