Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 2**

по курсу

«Операционные системы»

|  |  |
| --- | --- |
| Студент: | Пшеницын А. А. |
| Группа: | М8О-206Б-18 |
| Вариант: | 12 |
| Преподаватель: | Соколов Андрей Алексеевич |
| Оценка: |  |
| Дата: |  |
| Подпись: |  |

Москва, 2019

**Цель работы**

Приобретение практических навыков в:

* Управление процессами в ОС
* Обеспечение обмена данных между процессами посредством каналов

**Задание (Вариант № 12)**

Написание собственного простого целочисленного калькулятора с операцией «\*», «/». В

дочернем процессе должны происходить вычисления выражений. В родительском процессе

должны происходить вывод/ввод и передача их дочернему процессу.

**Общие сведения о программе**

В программе используются следующие функции:

* pid\_t fork(void) - при вызове fork() порождается новый процесс (процесс-потомок), который почти идентичен порождающему процессу-родителю.
* int pipe(int \*fd) — Параметр fd является указателем на массив из двух целых переменных. При нормальном завершении вызова в первый элемент массива – fd[0] – будет занесен файловый дескриптор, соответствующий выходному потоку данных pip’а и позволяющий выполнять только операцию чтения, а во второй элемент массива – fd[1] – будет занесен файловый дескриптор, соответствующий входному потоку данных и позволяющий выполнять только операцию записи.

**Код программы**

#include<stdio.h>

#include<stdbool.h>

#include<stdlib.h>

#include<string.h>

#include<sys/types.h>

#include<sys/wait.h>

#include<unistd.h>

#include<math.h>

#define SIZE 2

void WritePipe(int \*lhs, int \*rhs, char \*op, int fd) {

write(fd, lhs, sizeof(int));

write(fd, rhs, sizeof(int));

write(fd, op, sizeof(char));

}

void ReadPipe(int \*res, int fd) {

read(fd, res, sizeof(int));

}

void write\_str(int res){

char c[100];

sprintf(c, "%d", res);

int i = 0;

while(c[i] != '\0'){

i++;

}

write(1, c, i);

}

typedef enum{

T\_VAL,

T\_NULL,

T\_OPER,

T\_MINUS,

}T\_TYPE;

typedef struct{

int arg1;

int arg2;

char oper;

bool out;

}ARG;

ARG token\_read(){

ARG res;

T\_TYPE prev = T\_NULL;

T\_TYPE prev\_tmp = T\_NULL;

while(1){

char c = getchar();

if(c == EOF || c == '\n'){

res.out = false;

return res;

}

else if(c == ' '){

continue;

}

else if(c >= '0' && c <= '9'){

if(prev == T\_VAL){

res.out = false;

return res;

}

int tmp = c - '0';

while(1){

c = getchar();

if(c >= '0' && c <= '9'){

tmp = tmp \* 10 + (c - '0');

}

else{

ungetc(c, stdin);

break;

}

}

if(prev == T\_NULL){

if(prev\_tmp == T\_MINUS){

res.arg1 = -tmp;

prev\_tmp = T\_NULL;

}

else{

res.arg1 = tmp;

}

prev = T\_VAL;

}

else{

if(prev\_tmp == T\_MINUS){

res.arg2 = -tmp;

prev\_tmp = T\_NULL;

}

else{

res.arg2 = tmp;

}

res.out = true;

return res;

}

}

else if(c == '\*' || c == '/'){

if(prev != T\_VAL){

res.out = false;

return res;

}

res.oper = c;

prev = T\_OPER;

}

else if(c == '-'){

if(prev == T\_NULL || prev == T\_OPER){

prev\_tmp = T\_MINUS;

}

else{

res.out = false;

return res;

}

}

else{

res.out = false;

return res;

}

}

}

int main(){

int fd1[SIZE];

int fd2[SIZE];

char buf[100];

pid\_t pid;

ARG tmp;

if (pipe(fd1) == -1){

printf("Pipe не создался\n");

exit(0);

}

if(pipe(fd2) == -1){

printf("Pipe не создался\n");

exit(0);

}

pid = fork();

if (pid < 0) {

printf("Error. Проблемы с fork\n");

exit(0);

} else if (pid > 0) {

close(fd1[0]);

close(fd2[1]);

int res;

tmp = token\_read();

if(!tmp.out){

printf("ERROR\n");

return 1;

}

WritePipe(&tmp.arg1, &tmp.arg2, &tmp.oper, fd1[1]);

ReadPipe(&res, fd2[0]);

write\_str(res);

write(1, "\n", 1);

close(fd1[1]);

close(fd2[0]);

} else {

close(fd1[1]);

close(fd2[0]);

int res, lhs, rhs;

char oper;

read(fd1[0], &lhs, sizeof(int));

read(fd1[0], &rhs, sizeof(int));

read(fd1[0], &oper, sizeof(char));

if(oper == '\*'){

res = lhs \* rhs;

}

else{

res = lhs / rhs;

}

write(fd2[1], &res, sizeof(res));

close(fd1[0]);

close(fd2[1]);

}

return 0;

}

**Принцип работы**

Формат команды: «число» «операция» «число»

Создаются 2 процесса : родительский и дочерний. Для взаимодействия этих процессов

используется . Первый предназначен для передачи дочернему процессу аргументов, которые дочерний процесс должен обработать, а второй — для передачи родительскому процессу результата работы в дочернем процессе для печати в стандартный вывод.

**Примеры выполнения**

test 1:

Входные данные: 6 \* 5

Выходные данные: 30

test 2:

Входные данные: -2 \* 5

Выходные данные: -10

test 3:

Входные данные: 6 / 2

Выходные данные: 3

test 4:

Входные данные: - - 5 / 2

Выходные данные: 2

**Вывод**

Таким образом, мною был изучен fork(). **Fork()**— системный вызов в [Unix-подобных операционных системах](https://ru.wikipedia.org/wiki/Unix-подобная_операционная_система), создающих новый процесс(потомок), который является практически полной копией процесса-родителя, выполняющего этот вызов.