

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»

Группа: М8О-214Б-23

Студент: Гайдуков А.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 06.11.24

Москва, 2024

Постановка задачи

Вариант 1.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через `pipe1`, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через `pipe2`. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

1 вариант) Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип `int`. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- `pid_t fork(void);` – создает дочерний процесс.
- `int pipe(int *fd);` – создает поток ввода-вывода.
- `dup2(int filedeS, int filedeS2);` - дублирует поток `filedeS` в `filedeS2`.
- `close(int filedeS);` - закрывает поток.
- `int execl(const char *path, const char *arg0, ... /*, (char *)0 */);` - запускает программу с переданными аргументами
-

1. Чтение входных данных:

- Программа считывает имя входного файла с аргумента командной строки или со стандартного ввода.
- Со стандартного ввода читает строку
- Строка содержит числа, которые будут использоваться в программе "summ".

2. Создание каналов:

- Создаются два канала (`pipe`): `fd1` и `fd2`.
- Каналы используются для передачи данных между родительским и дочерним процессами.
- `fd1` будет использоваться для передачи строки `str` в дочерний процесс, а `fd2` - для получения результата от дочернего процесса.

3. Создание дочернего процесса:

- Используется функция `fork()` для создания нового процесса, который является копией родительского процесса.

- `p` - это идентификатор дочернего процесса.

- Если `p` меньше 0, значит `fork()` не удался.

- Если `p` больше 0, то это родительский процесс.

- Если `p` равен 0, то это дочерний процесс.

4. Работа родительского процесса:

- Закрывает ненужные дескрипторы файлов для канала `fd1`.

- Записывает строку `str` в канал `fd1` с помощью `write()`.

- Закрывает дескриптор записи канала `fd1`.

- Ожидает завершения дочернего процесса с помощью `wait()`.

- Закрывает ненужные дескрипторы файлов для канала `fd2` (запись в канал не нужна).

- Читает результат из канала `fd2` с помощью `read()`.

- Выводит полученный результат на стандартный вывод (`stdout`).

- Закрывает дескриптор чтения канала `fd2`.

5. Работа дочернего процесса:

- Закрывает стандартный вывод (`stdout`) и дублирует дескриптор записи канала `fd2` на `STDOUT_FILENO`, чтобы вывод дочернего процесса перенаправлялся в канал.

- Закрывает стандартный ввод (`stdin`) и дублирует дескриптор чтения канала `fd1` на `STDIN_FILENO`, чтобы ввод дочернего процесса считывался из канала.

- Закрывает все дескрипторы файлов, связанные с каналами.

- Выполняет программу "`summ`" с помощью `exec()`, передавая ей имя входного файла (`filename`) как аргумент.

- Если `exec()` возвращает -1, значит произошла ошибка при запуске программы "`summ`".

Код программы

head.c

```
#include "mpio.h"
#include <stdlib.h>
```

```

#include <string.h>
#include <ctype.h>
#include <sys/wait.h>
#include <errno.h>
extern int errno;


#define BUFSIZ 8192


int main(int argc, char* argv[]){
    char file_name[BUFSIZ];
    if(argc < 2){
        stdin_read(file_name, BUFSIZ);
    }else{
        strcpy(file_name, argv[1]);
    }

    char str[BUFSIZ];
    stdin_read(str, BUFSIZ);

    int fd1[2], fd2[2];
    pid_t p;

    if(pipe(fd1) == -1 || pipe(fd2) == -1) {
        stdout_write("Pipe failed!\n", 13);
        return -1;
    }

    p = fork();

    if(p < 0){
        stdout_write("Fork failed!\n", 13);
        return -1;
    }
    else if(p > 0){
        //главный процесс

        char res[100];

        close(fd1[0]);

        write(fd1[1], str, strlen(str) + 1);
        close(fd1[1]);
        wait(NULL);
        close(fd2[1]);

        int len = read(fd2[0], res, 100);
        res[len] = '\n';
    }
}

```

```

        res[len+1] = 0;

        stdout_write(res, strlen(res) + 1);
        stdout_write("\n", 1);
        close(fd2[0]);
    }
    else{
        //дочерний процесс, отсюда нужно запустить следующую программу с передачей
аргументов
        // + не забыть про filename первым аргументом
        /* close(STDOUT_FILENO);
        dup(fd2[1]); */
        dup2(fd2[1], STDOUT_FILENO);
        dup2(fd1[0], STDIN_FILENO);

        /* close(STDIN_FILENO);
        dup(fd1[0]); */

        close(fd1[0]);
        close(fd1[1]);
        close(fd2[0]);
        close(fd2[1]);

        int ret = execl("./summ", "./summ", file_name, NULL);
        if(ret == -1){
            char* sstr = strerror(errno);

            stdout_write(sstr, strlen(sstr) + 1);
            return -1;
        }
    }

    //free(finstr);
    return 0;
}

```

summ.c

```

#include <stdlib.h>
#include <ctype.h>
#include <unistd.h>
#include <ctype.h>
#include "mpio.h"
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define BUFSIZ 1024

```

```

int sum_of_strnums(char* str, int len){
    char num[100];
    int n_len = 0;
    int res = 0;
    for (int i = 0; i < len; i++)
    {
        char c = str[i];
        if(isdigit(str[i]) || str[i] == '-'){
            num[n_len++] = str[i];
        }
        else if((isspace(str[i]) && n_len != 0)){
            num[n_len] = '\0';
            res += atoi(num);
            n_len = 0;
        }
        else if(iscntrl(str[i])){
            num[n_len] = '\0';
            res += atoi(num);
            n_len = 0;
            break;
        }
        else{
            stdout_write("Not a number detected!\n", 23);
            stdout_write(str + i, 1);
            return 0;
        }
    }
    return res;
}

```

```

char* int_to_str(int number, char* string){
    char rev_num[100];
    int len = 0;
    for (int i = 0; number != 0; i++)
    {
        rev_num[i] = '0' + number % 10;
        number /= 10;
        len = i + 1;
    }
    for (int i = len - 1, j = 0; i >= 0; i--, j++)
    {
        string[j] = rev_num[i];
    }
    string[len] = '\0';
    return string;
}

```

```

int main(int argc, char* argv[]){

```

```

if(argc < 2){
    stdout_write("Wrong amount of arguments!\n", 27);
    return -1;
}
char buf[BUFSIZ];

int len = stdin_read(buf, BUFSIZ);
buf[len] = '\0';

char filename[100];
strcpy(filename, argv[1]);
int sum = sum_of_strnums(buf, strlen(buf) + 1);
char res[100];
int_to_str(sum, (char*)res);
int dtr = open(filename, O_CREAT | O_TRUNC | O_RDWR, S_IRUSR | S_IWUSR | S_IRGRP |
S_IWGRP | S_IROTH | S_IWOTH);
if(dtr == -1){
    stdout_write("File error!\n", 12);
    return -1;
}
write(dtr, res, strlen(res));
write(dtr, "\n", 1);

close(dtr);

stdout_write(res, strlen(res) + 1);
return 0;
//res надо записать в файл с именем filename. Через fp2 надо передать результат
обратно
}

```

mpio.c

```

#include <unistd.h>
#include <fcntl.h>

ssize_t stdin_read(void *buf, size_t cap)
{
    return read(STDIN_FILENO, buf, cap);
}

ssize_t stdout_write(const void *data, size_t len)
{
    return write(STDOUT_FILENO, data, len);
}

int mp_fopen(char* filename, int flags, mode_t mode){
    int res = open(filename, flags, O_RDWR);
    if(!res){
        res = open(filename, flags, O_CREAT);
    }
}

```

```

        return res;
    }

    int mp_fclose(int fd){
        return close(fd);
    }

```

Протокол работы программы

Тестирование:

```

gaalex@gaalex-HP-ProBook-445-G7:~/Programs/OS/First_lab$ ./head test.txt
1 23 14
38

```

```

test.txt
1 38
2

```

```

gaalex@gaalex-HP-ProBook-445-G7:~/Programs/OS/First_lab$ ./head test.txt
12312 12 111
12435

```

```

test.txt
1 12435
2

```

```

gaalex@gaalex-HP-ProBook-445-G7:~/Programs/OS/First_lab$ ./head test.txt
-100 100 1
1

```

```

test.txt
1 1
2

```

Strace:

```

$ strace ./head test.txt
execve("./head", [ "./head", "test.txt"], 0x7ffc028f13c8 /* 52 vars */) = 0
brk(NULL)                               = 0x5577e6cfb000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fff71728440) = -1 EINVAL (Недопустимый аргумент)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fd1e7666000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)

```



```

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=84827, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 84827, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd1e7651000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
832 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0"..., 832) =
= 784 pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0"..., 784, 64)
848) pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0"..., 48,
= 48 pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) =
68 newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
= 784 pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0"..., 784, 64)
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd1e7428000
mprotect(0x7fd1e7450000, 2023424, PROT_NONE) = 0
3, 0x28000) = 0x7fd1e7450000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
0x1bd000) = 0x7fd1e75e0000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
3, 0x215000) = 0x7fd1e763e000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
-1, 0) = 0x7fd1e7644000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
close(3) = 0
0x7fd1e7425000 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
arch_prctl(ARCH_SET_FS, 0x7fd1e7425740) = 0
set_tid_address(0x7fd1e7425a10) = 12213
set_robust_list(0x7fd1e7425a20, 24) = 0
rseq(0x7fd1e74260e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7fd1e763e000, 16384, PROT_READ) = 0
mprotect(0x5577e5167000, 4096, PROT_READ) = 0
mprotect(0x7fd1e76a0000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fd1e7651000, 84827) = 0
read(0, 1 2 3
"1 2 3\n", 8192) = 6
pipe2([3, 4], 0) = 0
pipe2([5, 6], 0) = 0
child clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fd1e7425a10) = 12234
close(3) = 0
write(4, "1 2 3\n\0", 7) = 7
close(4) = 0
wait4(-1, NULL, 0, NULL) = 12234
si_ --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=12234, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
close(6) = 0
read(5, "6\0", 100) = 2
write(1, "6\0", 26) = 2
write(1, "\n", 1
) = 1
close(5) = 0

```

```
exit_group(0)                = ?  
+++ exited with 0 +++
```

Вывод

Я научился писать приложения с несколькими процессами. В ходе работы над лабораторной работой я не столкнулся с какими-либо проблемами.