



Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М80-207Б-20

Студент: Мерц С.П.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 07.11.21

Москва, 2021.

Содержание

- 1 Постановка задачи.
- 2 Общие сведения о программе.
- 3 Общий метод и алгоритм решения.
- 4 Код программы.
- 5 Демонстрация работы программы.
- 6 Вывод.

Постановка задачи

Составить и отладить программу на языке C, родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода. 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.

Общие сведения о программе

Программа состоит из файлов parent.c, child1.c, child2.c. В файле parent.c храниться родительский процесс и создание дочернего процесса, также перенаправление потока вывода. В child1.c. мы задаем первому ребенку перевести строки в верхний регистр. В child2.c. мы задаём убрать все двойные пробелы.

Программа использует следующие системные вызовы:

- 1 **read** – для чтения данных из входного потока.
- 2 **write** – для записи данных в файл или выходной поток.
- 3 **pipe** – для создания канала, через который процессы могут обмениваться информацией.
- 4 **fork** – для создания дочернего процесса.
- 5 **dup2** – для перенаправления потока вывода.
- 6 **execve** - замена образа памяти процесса

Общий метод и алгоритм решения

В программе parent.c создаем дочерние процессы. В fork() проверим создание и работу дочерних процессов. Далее создаем три канала pipe, так как у нас идет передача между родительским процессом в первый дочерний, затем посылаем из первого дочернего процесса во второй, и уже из второго опять передаем в родительский. Пропишем проверки для pipe. Дальше пропишем в двух отдельных файлах child1.c, child2.c. выполнение задания для двух дочерних процессах, где child1.c переводит строку в верхний регистр, а child2.c убирает двойные пробелы. Пишем вывод программы.

Код программы

parent.c:

```
#include <stdio.h>
#include <unistd.h>

int Spawning_Child_Processes (char *fname, int read, int write) {
    switch (fork()) {
        case -1:
            return -1;
        case 0: //child
            {
                char *args[] = {NULL};
                if(dup2(read, 0) == -1)
                    printf("dup2 error!");
                if(dup2(write, 1) == -1)
                    printf("dup2 error!");
                if(execv(fname, args) == -1)
                    printf("execv error!");
                return 0;
            }
        default:
            break;
    }
    return 0;
}

int main() {
    int pipe1[2], pipe2[2], pipe3[2];

    if (pipe(pipe1) == -1)
        printf("Pipe1 error!");
    if (pipe(pipe2) == -1)
        printf("Pipe2 error!");
    if (pipe(pipe3) == -1)
        printf("Pipe3 error!");

    if (Spawning_Child_Processes("./child1", pipe1[0], pipe2[1])) {
        perror("fork error");
        return -1;
    }
    if (Spawning_Child_Processes("./child2", pipe2[0], pipe3[1])) {
        perror("fork error");
        return -1;
    }

    printf("Enter string:\n");
    char c;
    while ((c = getchar()) != EOF) {
        write(pipe1[1], &c, 1);
        read(pipe3[0], &c, 1);
        printf("%c", c);
        fflush(stdout);
    }
}
```

```

    }
    return 0;
}

```

child1.c:

```

#include <unistd.h>
#include <stdio.h>

char toUpper(char c) {
    if (c >= 'a' && c <= 'z')
        return c - ('a' - 'A');
    return c;
}

int main() {
    char c;
    while ((c = getchar()) != EOF) {
        printf("%c", toUpper(c));
        fflush(stdout);
    }
    return 0;
}

```

child2.c:

```

#include <unistd.h>
#include "stdio.h"

void PrintChar(char x) {
    printf("%c", x);
    fflush(stdout);
}

int main() {
    char c;
    while ((c = getchar()) != EOF) {
        PrintChar(c);
        if (c == ' ') {
            while ((c = getchar()) == ' ')
                PrintChar(0);
            PrintChar(c);
        }
    }
    return 0;
}

```

Демонстрация работы программы

Enter string:

fGG jfFjj

FGG JFFJJ

Enter string:

FFFFf kkK

FFFFF KKK

Системные вызовы(strace)

```
execve("./parent", [ "./parent" ], 0x7ffff88e36b0 /* 20 vars */) = 0
```

```
brk(NULL) = 0x7fffd6e33000
```

```
arch_prctl(0x3001 /* ARCH_??? */, 0x7fffded0d5a0) = -1 EINVAL (Invalid argument)
```

```
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
```

```
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
```

```
fstat(3, {st_mode=S_IFREG|0644, st_size=34764, ...}) = 0
```

```
mmap(NULL, 34764, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1af0927000
```

```
close(3) = 0
```

```
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
```

```
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0\0"... , 832) = 832
```

```
pread64(3, "\6\0\0\0\4\0\0\0a\0\0\0\0\0\0\0a\0\0\0\0\0\0\0a\0\0\0\0\0\0\0"... , 784, 64) = 784
```

```
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"... , 32, 848) = 32
```

```
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"... , 68, 880) = 68
```

```

    fstat(3, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 0

    mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f1af0920000

    pread64(3,
"\6\0\0\0\4\0\0\0a\0\0\0\0\0\0\0a\0\0\0\0\0\0\0a\0\0\0\0\0\0\0"... , 784, 64) =
784

    pread64(3,
"\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0", 32, 848) =
32

    pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>
\263"... , 68, 880) = 68

    mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f1af0720000

    mprotect(0x7f1af0745000, 1847296, PROT_NONE) = 0

    mmap(0x7f1af0745000, 1540096, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7f1af0745000

    mmap(0x7f1af08bd000, 303104, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19d000) = 0x7f1af08bd000

    mmap(0x7f1af0908000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f1af0908000

    mmap(0x7f1af090e000, 13528, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1af090e000

    close(3) = 0

    arch_prctl(ARCH_SET_FS, 0x7f1af0921540) = 0

    mprotect(0x7f1af0908000, 12288, PROT_READ) = 0

    mprotect(0x7f1af0963000, 4096, PROT_READ) = 0

    mprotect(0x7f1af095d000, 4096, PROT_READ) = 0

    munmap(0x7f1af0927000, 34764) = 0

    pipe([3, 4]) = 0

    pipe([5, 6]) = 0

    pipe([7, 8]) = 0

    clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f1af0921810) = 621

```

```

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f1af0921810) = 622

fstat(1, {st_mode=S_IFCHR|0660, st_rdev=makedev(0x4, 0x1), ...}) = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
brk(NULL) = 0x7fffd6e33000
brk(0x7fffd6e54000) = 0x7fffd6e54000
write(1, "Enter string:\n", 14Enter string:
) = 14
fstat(0, {st_mode=S_IFCHR|0660, st_rdev=makedev(0x4, 0x1), ...}) = 0
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...}) = 0
read(0, abcd DggfF Ggdfg abcd
"abcd DggfF Ggdfg abcd\n", 4096) = 30
write(4, "a", 1) = 1
read(7, "A", 1) = 1
write(1, "A", 1A) = 1
write(4, "b", 1) = 1
read(7, "B", 1) = 1
write(1, "B", 1B) = 1
write(4, "c", 1) = 1
read(7, "C", 1) = 1
write(1, "C", 1C) = 1
write(4, "d", 1) = 1
read(7, "D", 1) = 1
write(1, "D", 1D) = 1
write(4, " ", 1) = 1
read(7, " ", 1) = 1
write(1, " ", 1 ) = 1
write(4, " ", 1) = 1
read(7, "\\0", 1) = 1
write(1, "\\0", 1) = 1

```


write(4, " ", 1)	= 1
read(7, "\0", 1)	= 1
write(1, "\0", 1)	= 1
write(4, "D", 1)	= 1
read(7, "D", 1)	= 1
write(1, "D", 1D)	= 1
write(4, "g", 1)	= 1
read(7, "G", 1)	= 1
write(1, "G", 1G)	= 1
write(4, "g", 1)	= 1
read(7, "G", 1)	= 1
write(1, "G", 1G)	= 1
write(4, "f", 1)	= 1
read(7, "F", 1)	= 1
write(1, "F", 1F)	= 1
write(4, "F", 1)	= 1
read(7, "F", 1)	= 1
write(1, "F", 1F)	= 1
write(4, " ", 1)	= 1
read(7, " ", 1)	= 1
write(1, " ", 1)	= 1
write(4, " ", 1)	= 1
read(7, "\0", 1)	= 1
write(1, "\0", 1)	= 1
write(4, " ", 1)	= 1
read(7, "\0", 1)	= 1
write(1, "\0", 1)	= 1
write(4, " ", 1)	= 1
read(7, "\0", 1)	= 1
write(1, "\0", 1)	= 1
write(4, " ", 1)	= 1
read(7, "\0", 1)	= 1
write(1, "\0", 1)	= 1
write(4, " ", 1)	= 1

read(7, "\0", 1)	= 1
write(1, "\0", 1)	= 1
write(4, " ", 1)	= 1
read(7, "\0", 1)	= 1
write(1, "\0", 1)	= 1
write(4, " ", 1)	= 1
read(7, "\0", 1)	= 1
write(1, "\0", 1)	= 1
write(4, "G", 1)	= 1
read(7, "G", 1)	= 1
write(1, "G", 1G)	= 1
write(4, "g", 1)	= 1
read(7, "G", 1)	= 1
write(1, "G", 1G)	= 1
write(4, "d", 1)	= 1
read(7, "D", 1)	= 1
write(1, "D", 1D)	= 1
write(4, "f", 1)	= 1
read(7, "F", 1)	= 1
write(1, "F", 1F)	= 1
write(4, "g", 1)	= 1
read(7, "G", 1)	= 1
write(1, "G", 1G)	= 1
write(4, " ", 1)	= 1
read(7, " ", 1)	= 1
write(1, " ", 1)	= 1
write(4, "a", 1)	= 1
read(7, "A", 1)	= 1
write(1, "A", 1A)	= 1
write(4, "b", 1)	= 1
read(7, "B", 1)	= 1

```

write(1, "B", 1B)                = 1
write(4, "c", 1)                  = 1
read(7, "C", 1)                   = 1
write(1, "C", 1C)                 = 1
write(4, "d", 1)                  = 1
read(7, "D", 1)                   = 1
write(1, "D", 1D)                 = 1
write(4, "\n", 1)                 = 1
read(7, "\n", 1)                  = 1
write(1, "\n", 1
)                                  = 1

read(0, 0x7fffd6e342b0, 4096)     = ? ERESTARTSYS (To be restarted
if SA_RESTART is set)

--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, "", 4096)                  = 0
exit_group(0)                      = ?
+++ exited with 0 +++

```

Вывод

В данной лабораторной работе я познакомился с потоками в операционной системе. Передавал данные между потоками через pipe, перенаправляя стандартные потоки ввода/вывода. Было весело приостанавливать процессы из консоли и обнаруживать их в таблице процессов.