Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**Тема работы**

**“Межпроцессорное взаимодействие через memory-mapped files”**

Студент: Борисов Ян Артурович

Группа: М8О-208Б-20

Вариант: 19

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

<https://github.com/Yannikupy/OS>

**Постановка задачи**

Задача: реализовать программу, в которой родительский процесс создает два дочерних процесса. Родительский процесс принимает строки, которые отправляются в тот или иной дочерний процесс в зависимости от следующего правила: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Оба процесса удаляют гласные из строк. Межпроцессорное взаимодействие осуществляется посредством отображаемых файлов (memory-mapped files).

**Общие сведения о программе**

Программа содержится в файлах parent.c и child.c

**Общий метод и алгоритм решения**

При запуске программы пользователю предлагается ввести имя файла для первого и для второго дочернего процесса. В эти файлы будет записываться вывод соответствующих процессов.

После запуска программы выполняется отображение двух файлов, имена которых известны заранее. Так как операционная система не позволяет выполнить отображение пустого файла, то перед отображением в файлы записываются «пустые» строки. В качестве «пустой» строки используется строка, состоящая из одного системного символа.

Затем создаются два дочерних процесса. Родительский процесс считывает строки с консольного ввода при помощи функции get\_string(). Данная функция считывает строку произвольной длины из стандартного ввода. Затем при помощи функции rand() определяется дочерний процесс, которому отправится эта строка на обработку.

Передача строки дочерним процессам осуществляется с помощью ее копирования в отображенный файл.

Дочерние процессы перенаправляют свой стандартный вывод с помощью dup2 в созданный файл. Затем они заменяют свой образ памяти и выполняют программу child, в которой они считывают строки и удаляют из нее все гласные буквы.

В качестве сигнала используется «пустая» строка. Если дочерний процесс считал «пустую» строку, то ему не нужно ничего выполнять. Если же считана другая строка, то её необходимо обработать. После обработки в отображённый файл вновь записывается «пустая» строка.

Если пользователь нажал Ctrl+D, то родительский процесс посылает обоим дочерним процессам сигнал о завершении работы, закрывает все файлы и завершается сам. Отображаемые файлы, использованные для взаимодействия процессов, удаляются.

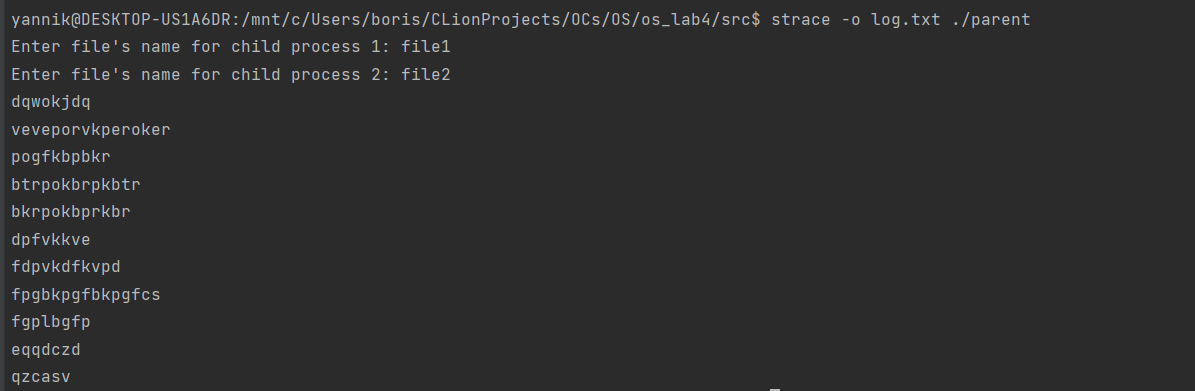
**Исходный код**

**Parent.c**

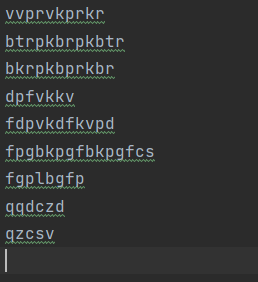
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <string.h>  
#include <unistd.h>  
#include <fcntl.h>  
#include <sys/stat.h>  
#include <sys/mman.h>  
  
#define MAP\_SIZE 4096  
  
// files for mapping  
char\* file1\_name = "file1\_mapped";  
char\* file2\_name = "file2\_mapped";  
  
// empty string as a signal  
char empty = 1;  
char\* empty\_string = &empty;  
  
// scan a string with unknown length  
char\* get\_string() {  
 int len = 0, capacity = 10;  
 char\* s = (char\*)malloc(10 \* sizeof(char));  
 if (s == NULL) {  
 perror("Can't read a string");  
 exit(6);  
 }  
  
 char c;  
 while ((c = getchar()) != '\n') {  
 s[len++] = c;  
 if (c == EOF) {  
 break;  
 }  
 if (len == capacity) {  
 capacity \*= 2;  
 s = (char\*)realloc(s, capacity \* sizeof(char));  
 if (s == NULL) {  
 perror("Can't read a string");  
 exit(6);  
 }  
 }  
 };  
 s[len] = '\0';  
 return s;  
}  
  
  
int main() {  
 srand(time(NULL));  
  
 // creating files for output of child processes  
 printf("Enter file's name for child process 1: ");  
 char\* output\_file1\_name = get\_string();  
  
 printf("Enter file's name for child process 2: ");  
 char\* output\_file2\_name = get\_string();  
  
 int output\_file1 = open(output\_file1\_name, O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IWRITE | S\_IREAD);  
 int output\_file2 = open(output\_file2\_name, O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IWRITE | S\_IREAD);  
 if (output\_file1 < 0 || output\_file2 < 0) {  
 perror("Can't open file");  
 exit(1);  
 }  
  
  
 // creating files for mapping  
 int fd1 = open(file1\_name, O\_RDWR | O\_CREAT, S\_IWRITE | S\_IREAD);  
 int fd2 = open(file2\_name, O\_RDWR | O\_CREAT, S\_IWRITE | S\_IREAD);  
 if (fd1 < 0 || fd2 < 0) {  
 perror("Can't open file");  
 exit(1);  
 }  
  
 // empty files can't be mapped, so we'll put our empty\_string there  
 if (write(fd1, empty\_string, sizeof(empty\_string)) < 0) {  
 perror("Can't write to file");  
 exit(1);  
 }  
 if (write(fd2, empty\_string, sizeof(empty\_string)) < 0) {  
 perror("Can't write to file");  
 exit(1);  
 }  
  
 // mapping files  
 char\* file1 = mmap(NULL, MAP\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd1, 0);  
 char\* file2 = mmap(NULL, MAP\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd2, 0);  
 if (file1 == MAP\_FAILED || file2 == MAP\_FAILED) {  
 perror("Can't map a file");  
 exit(2);  
 }  
  
  
 // creating child processes  
 pid\_t pid1 = fork();  
 if (pid1 < 0) {  
 perror("Can't create child process");  
 exit(3);  
 }  
  
 if (pid1 > 0) { // parent  
 pid\_t pid2 = fork();  
 if (pid2 < 0) {  
 perror("Can't create child process");  
 exit(3);  
 }  
  
 if (pid2 > 0) { // parent  
  
 while (1) {  
 char\* s = get\_string();  
  
 if (rand() % 100 + 1 <= 80) {  
 strcpy(file1, s);  
 if (s[0] == EOF) {  
 strcpy(file2, s);  
 break;  
 }  
 }  
 else {  
 strcpy(file2, s);  
 if (s[0] == EOF) {  
 strcpy(file1, s);  
 break;  
 }  
 }  
 }  
 if (munmap(file1, MAP\_SIZE) < 0 || munmap(file2, MAP\_SIZE) < 0) {  
 perror("Can't unmap files");  
 exit(4);  
 }  
 if (close(fd1) < 0 || close(fd2) < 0) {  
 perror("Can't close files");  
 exit(5);  
 }  
 if (remove(file1\_name) < 0 || remove(file2\_name) < 0) {  
 perror("Can't delete files");  
 exit(6);  
 }  
 }  
 else { // child2  
 // redirecting output  
 if (dup2(output\_file2, STDOUT\_FILENO) < 0) {  
 perror("Can't redirect stdout for child process");  
 exit(7);  
 }  
  
 char\* arr [] = {"2", NULL};  
 execv("child", arr);  
  
 // it won't go here if child executes  
 perror("Can't execute child process");  
 exit(8);  
 }  
 }  
 else { // child1  
 // redirecting output  
 if (dup2(output\_file1, STDOUT\_FILENO) < 0) {  
 perror("Can't redirect stdout for child process");  
 exit(7);  
 }  
  
 char\* arr [] = {"1", NULL};  
 execv("child", arr);  
  
 // it won't go here if child executes  
 perror("Can't execute child process");  
 exit(8);  
 }  
}

**Child.c**

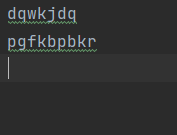
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <fcntl.h>  
#include <sys/stat.h>  
#include <sys/mman.h>  
  
#define MAP\_SIZE 4096  
  
// files for mapping  
char\* file1\_name = "file1\_mapped";  
char\* file2\_name = "file2\_mapped";  
  
// empty string as a signal  
char empty = 1;  
char\* empty\_string = &empty;  
  
void removeChar(char \*str, char garbage) {  
  
 char \*src, \*dst;  
 for (src = dst = str; \*src != '\0'; src++) {  
 \*dst = \*src;  
 if (\*dst != garbage) dst++;  
 }  
 \*dst = '\0';  
}  
  
void delete\_vowels(char \*str) {  
 int length = strlen(str);  
 char \*front = str;  
 char \*back = str + length - 1;  
  
 while (front <= back) {  
 if((\*front == 'a') || (\*front == 'e') || (\*front == 'i') || (\*front == 'o') || (\*front == 'u') || (\*front == 'y') ||  
 (\*front == 'A') || (\*front == 'E') || (\*front == 'I') || (\*front == 'O') || (\*front == 'U') || (\*front == 'Y')){  
 removeChar(str, \*front);  
 }  
 ++front;  
 }  
}  
  
int main(int argc, char\* argv[]) {  
 char\* file\_name;  
 if (argv[0][0] == '1') {  
 file\_name = file1\_name;  
 }  
 else if (argv[0][0] == '2') {  
 file\_name = file2\_name;  
 }  
 else {  
 perror("Unknown file");  
 exit(8);  
 }  
 // opening a file for mapping  
 int fd = open(file\_name, O\_RDWR | O\_CREAT, S\_IWRITE | S\_IREAD);  
 if (fd < 0) {  
 perror("Can't open file");  
 exit(1);  
 }  
  
 // mapping file  
 char\* file = mmap(NULL, MAP\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  
 if (file == MAP\_FAILED) {  
 perror("Can't map a file");  
 exit(2);  
 }  
  
 while (1) {  
 // waiting for a string  
 while (strcmp(file, empty\_string) == 0) {}  
  
 // terminating if Ctrl+D was pressed  
 if (file[0] == EOF) {  
 if (munmap(file, MAP\_SIZE) < 0) {  
 perror("Can't unmap file");  
 exit(4);  
 }  
 exit(0);  
 }  
  
 char\* string = (char\*)malloc(strlen(file) \* sizeof(char));  
 strcpy(string, file);  
 delete\_vowels(string);  
 printf("%s\n", string);  
 fflush(stdout);  
 strcpy(file, empty\_string);  
 free(string);  
 }  
}

**Демонстрация работы программы**

File1



File2

  
  
**log.txt**

execve("./parent", ["./parent"], 0x7fffd187b670 /\* 16 vars \*/) = 0  
brk(NULL) = 0x7fffd1114000  
arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fffd8709570) = -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=34820, ...}) = 0  
mmap(NULL, 34820, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f2b068f1000  
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\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\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\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) = 0x7f2b068b0000  
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\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  
mmap(NULL, 2036952, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f2b066b0000  
mprotect(0x7f2b066d5000, 1847296, PROT\_NONE) = 0  
mmap(0x7f2b066d5000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25000) = 0x7f2b066d5000  
mmap(0x7f2b0684d000, 303104, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0x7f2b0684d000  
mmap(0x7f2b06898000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f2b06898000  
mmap(0x7f2b0689e000, 13528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f2b0689e000  
close(3) = 0  
arch\_prctl(ARCH\_SET\_FS, 0x7f2b068b1540) = 0  
mprotect(0x7f2b06898000, 12288, PROT\_READ) = 0  
mprotect(0x7f2b068fd000, 4096, PROT\_READ) = 0  
mprotect(0x7f2b068ed000, 4096, PROT\_READ) = 0  
munmap(0x7f2b068f1000, 34820) = 0  
time(NULL) = 1637843635 (2021-11-25T15:33:55+0300)  
fstat(1, {st\_mode=S\_IFCHR|0660, st\_rdev=makedev(0x4, 0x2), ...}) = 0  
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0  
brk(NULL) = 0x7fffd1114000  
brk(0x7fffd1135000) = 0x7fffd1135000  
fstat(0, {st\_mode=S\_IFCHR|0660, st\_rdev=makedev(0x4, 0x2), ...}) = 0  
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...}) = 0  
write(1, "Enter file's name for child proc"..., 39) = 39  
read(0, "file1\n", 4096) = 6  
write(1, "Enter file's name for child proc"..., 39) = 39  
read(0, "file2\n", 4096) = 6  
openat(AT\_FDCWD, "file1", O\_WRONLY|O\_CREAT|O\_TRUNC, 0600) = 3  
openat(AT\_FDCWD, "file2", O\_WRONLY|O\_CREAT|O\_TRUNC, 0600) = 4  
openat(AT\_FDCWD, "file1\_mapped", O\_RDWR|O\_CREAT, 0600) = 5  
openat(AT\_FDCWD, "file2\_mapped", O\_RDWR|O\_CREAT, 0600) = 6  
write(5, "\1\0\0\0\0\0\0\0", 8) = 8  
write(6, "\1\0\0\0\0\0\0\0", 8) = 8  
mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7f2b068f0000  
mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 6, 0) = 0x7f2b066a0000  
clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f2b068b1810) = 1486  
clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f2b068b1810) = 1487  
read(0, "dqwokjdq\n", 4096) = 9  
read(0, "veveporvkperoker\n", 4096) = 17  
read(0, "pogfkbpbkr\n", 4096) = 11  
read(0, "btrpokbrpkbtr\n", 4096) = 14  
read(0, "bkrpokbprkbr\n", 4096) = 13  
read(0, "dpfvkkve\n", 4096) = 9  
read(0, "fdpvkdfkvpd\n", 4096) = 12  
read(0, "fpgbkpgfbkpgfcs\n", 4096) = 16  
read(0, "fgplbgfp\n", 4096) = 9  
read(0, "eqqdczd\n", 4096) = 8  
read(0, "qzcasv\n", 4096) = 7  
read(0, "", 4096) = 0  
munmap(0x7f2b068f0000, 4096) = 0  
--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=1486, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---  
munmap(0x7f2b066a0000, 4096) = 0  
close(5) = 0  
close(6) = 0  
unlink("file1\_mapped") = 0  
unlink("file2\_mapped") = 0  
exit\_group(0) = ?  
+++ exited with 0 +++

**Выводы**В ходе выполнения данной работы мы расширили свои навыки работы с процессами и освоили технологию «файл маппинга», научились использовать ее правильно.