1. Разработать приложение, реализующее отображение файла в память.
2. Разработать приложение, реализующее чтение и запись в файл без отображения в память.
3. Сравнить быстродействие двух программ.

# Приложение, реализующее отображение файла в память

В рамках выполнения задания был изучен порядок отображения файла в память. Отображение файла в память (на память) — это способ работы с файлами в некоторых операционных системах, при котором всему файлу или некоторой непрерывной его части ставится в соответствие определённый участок памяти (диапазон адресов оперативной памяти).

void \* mmap(void \*start, size\_t length, int prot , int flags, int fd, off\_t offset) - функция mmap отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым описателем fd, в память, начиная с адреса start. Последний параметр (адрес) необязателен, и обычно бывает равен 0. Настоящее местоположение отраженных данных возвращается самой функцией mmap, и никогда не бывает равным 0.

Листинг 1. Приложение, реализующее отображение файла в память

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

#include <ctype.h>

#import <string.h>

#include <time.h>

int main() {

int fd = open("/home/komush/Desktop/file", O\_RDWR, S\_IRUSR | S\_IWUSR);

struct stat sb;

if (fstat(fd, &sb) < 0)

perror("couldn't get file size ");

printf("file size is %ld\n", sb.st\_size);

char message[17] = "Printed with mmap";

clock\_t start, end;

double cpu\_time\_used;

start = clock();

char \*file\_in\_memory = mmap(NULL, 17,

PROT\_READ | PROT\_WRITE,

MAP\_SHARED, fd, 0);

memcpy(file\_in\_memory, message, 17);

printf("%s\n", file\_in\_memory);

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("Time = %f", cpu\_time\_used);

munmap(file\_in\_memory, sb.st\_size);

close(fd);

}

Результат выполнения листинга 1 приведен на рисунке 1.

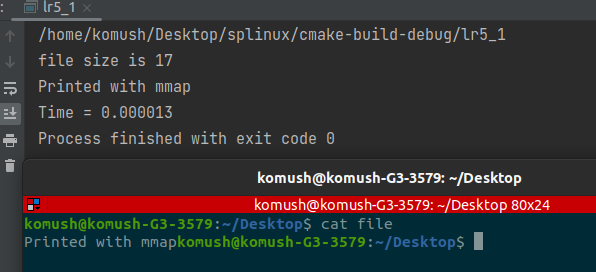


Рисунок 1. – Результат работы приложения

# Приложение, реализующее чтение и запись в файл без отображения в память

В рамках разработки приложения был повторно проработан порядок чтения и записи в файл без отображения в память. Использовались системные вызовы open(), read(), write().

Листинг 2. Приложение, чтение и запись в файл без отображения в память

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

#include <ctype.h>

#include <time.h>

int main() {

int fd = open("/home/komush/Desktop/file", O\_RDWR, S\_IRUSR | S\_IWUSR);

char buff[20];

char message[20] = "Printed without mmap";

clock\_t start, end;

double cpu\_time\_used;

start = clock();

if (read(fd, buff, 20) < 0)

perror("couldn't read file ");

write(fd, message, 20);

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("Time = %f\n", cpu\_time\_used);

printf("Read: %s\n", buff);

close(fd);

}

Результат выполнения листинга 2 приведен на рисунке 2.

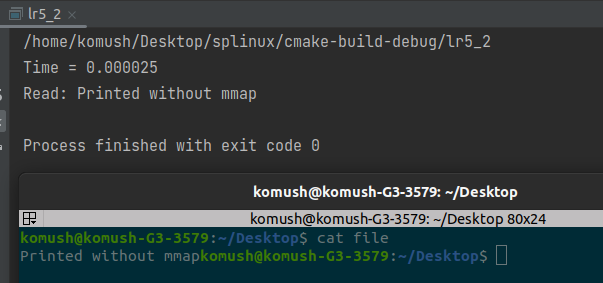


Рисунок 2. – Результат работы приложения

# Сравнение быстродействия разработанных выше программ

Для сравнения быстродействия 2 способов работы с файлами в коде обоих приложений используется функция clock() (см. Листинг1, Листинг 2). Функция показывает, сколько времени прошло с момента инициализации во время запуска процесса.

В рамках лабораторной работы было выявлено, что в случае с отображением файла в память расходуется меньшее количество времени (в данной реализации 0,000013 секунд, см. Рисунок 1), чем в случае работы с файлом без его отображения в память (в данной реализации 0,000025 секунд, см. Рисунок 2). Обуславливается это тем, что файлу сразу ставится в соответствие участок памяти, что требует меньших затрат при записи и чтении.

# Вывод

Разработано приложение, реализующее отображение файла в память. Были продемонстрированы чтение содержимого файла из памяти и запись в файл через память.

Так же разработано приложение, реализующее чтение и запись в файл без отображения в память.

По итогу сравнения быстродействия двух программ, было выявлено, что в первом случае требуемые действия выполняются за меньший промежуток времени, чем во втором. Обуславливается тем, что файлу сразу ставится в соответствие участок памяти, что требует меньших затрат при записи и чтении.