Московский авиационный институт
(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Операционные системы»

Курсовой проект

Тема: Текстовый процессор

Студент: Туманов Георгий
Группа: 80-201
Преподаватель: Соколов А.А.
Дата:
Оценка:

Москва, 2019

1. Постановка задачи

Написать собственный текстовый процессор. Процессор должен уметь открывать файлы. При этом при открытии нужно предусмотреть 2 режима. Первый - открытие файла только на чтение, а второй - только на запись.
Как будет выглядеть запуск текстового редактора
textProcessor.out myFile.txt readonly 15
Число 15 означает, что вы выделяете 15 kB для fileMapping этого файла. То есть вы по файлу должны ходить только "передвигая" это окно.
Далее у него должны быть предусмотрены следующие функции:
1) поиск текста по регулярному выражению и выводом информации с какого символа он начинается (можно строчку и место в строке).
2) Замена строки (если возможно несколько вариантов замены, то должен предложить какую заменить)
3) Вывести диапазон символов
Можно для поиска использовать наивные алгоритмы. При этом символы из файла вы должны держать только в FileMapping (конечно, вспомогательные переменные и массивы символов небольшого размера использовать можно)

2. Описание программы

Программа при запуске требует следующие аргументы:

- Имя файла, который будет открыт (обязательный параметр)
- Режим, по умолчанию чтение-запись
- Размер блока в килобайтах, по умолчанию 15.

При запуске программа открывает файл в указанном режиме и загружает в память блок указанного размера, начинающийся с начала файла.

Программа имеет следующие функции:

- setpos: устанавливает позицию блока. Из-за особенностей функции mmap, позиция должна делиться нацело на размер страницы.
- move: перемещает блок. Смещение также должно быть кратно размеру страницы.
- diap: выводит на экран границы блока
- mode: выводит на экран режим, в котором открыт файл
- size: выводит на экран размер файла

Функции, требующие доступ на чтение:

- print: печатает указанный диапазон символов блока

- find: выводит все вхождения введённого регулярного выражения в файле

Функции, требующие доступ на запись:

- write: начиная с указанной позиции заменяет все символы указанной строкой

Функции, требующие доступ и на чтение, и на запись:

- replace: заменяет все вхождения первой строки на вторую. Строки должны быть одинаковой длины, так как иначе бы изменился размер блока. Для каждого вхождения программа спрашивает, заменять ли данное вхождение. Варианты ответа:
    - y - заменить данное вхождение
    - n - пропустить данное вхождение
    - a - заменить данное и все последующие вхождения
    - s - пропустить данное вхождение и завершить команду

3. Набор testcases

№	Описание	Ввод

1	Тест с большим файлом	`$./kp.out war-and-peace.txt r 8`
		`> diap`
		`> mode`
		`> size`
		`> print 0 80`
		`> move 4096`
		`> print 0 80`
		`> quit`

2	Тест с одной строкой	`$./kp.out mini_test.txt`
		`> diap`
		`> print 0 35`
		`> find`
		`Find what: .b.`
		`> replace`
		`Replace what: c`
		`Replace with: d`
		`y`
		`n`
		`a`
		`> print 0 35`
		`> quit`

4. Результаты выполнения тестов.

test 1:

```
$ ./kp.out war-and-peace.txt r 8
> diap
Diapason: [0, 8191]
OK.
> mode
Mode: Read only
OK.
> size
File size: 3291641
OK.
> print 0 80
The Project Gutenberg EBook of War and Peace, by Leo Tolstoy

This eBook is
OK.
> move 4096
OK.
> print 0 80
PTER XXVIII

CHAPTER XXIX

CHAPTER XXX

CHAPTER XXXI

CHAPTER XXXII

CH
OK.
> quit
OK.
```

test 2:

```
$ ./kp.out mini_test.txt
> diap
Diapason: [0, 35]
OK.
> print 0 35
aba
abb
abc
bba
bbb
bbc
cba
cbb
```

```
cbc

OK.
> find
Find what: .b.
Found aba at offset 0
Found abb at offset 4
Found abc at offset 8
Found bba at offset 12
Found bbb at offset 16
Found bbc at offset 20
Found cba at offset 24
Found cbb at offset 28
Found cbc at offset 32
OK.
> replace
Replace what: c
Replace with: d
Replace at offset 10? (y/n/s/a): y
Replace at offset 22? (y/n/s/a): n
Replace at offset 24? (y/n/s/a): a
OK.
> print 0 35
aba
abb
abd
bba
bbb
bbc
dba
dbb
dbd

OK.
> quit
OK.
```

5. Листинг программы

file.h:

```
#pragma once
#include <regex>

enum class Answer : int
{
	no,
	yes,
	all,
	stop
};
```

```
class File
{
public:
	File(std::string fname, bool canRead, bool canWrite, int size);
	~File();

	std::vector<std::pair<int,std::string>> Find(std::regex what); //find
ALL entrances of pattern
	void	Replace(std::string	what,	std::string	with,
std::function<Answer(int)> confirm); //replace

	std::string Print(int from, int to); //show diapason of chars
	void Write(int to, std::string what); //set diapason of chars

	void Move(int delta);
	void SetPos(int pos);

	std::pair<int, int> Diapason();
	std::string Mode();
	int FileSize();

private:
	int m_fileDescr, m_fileSize; //file descriptor and size of file
	char *m_map; //mapped piece of file

	bool m_canRead, m_canWrite; //can read from/write to file
	int m_chunkPos, m_chunkSize, m_maxChunkSize; //position in file and
size of chunk

	void Mmap(int pos);
	bool CompareCharString(int off, std::string str);
};
```

file.cpp:

```
#include "file.h"
#include <sys/stat.h>
#include <sys/mman.h>
#include <unistd.h>
#include <fcntl.h>

File::File(std::string fname, bool canRead, bool canWrite, int size) :
	m_canRead(canRead), m_canWrite(canWrite), m_maxChunkSize(size * 1024)
{
	//open mode
	int openflag = O_RDWR;
	//if (!canRead) openflag = O_WRONLY; else //THIS MUST APSENT BECAUSE OF
MMAP WEIRD BEHAVIOUR
	if (!canWrite) openflag = O_RDONLY;

	//open file
	m_fileDescr = open(fname.c_str(), openflag);
	if (!m_fileDescr) throw std::logic_error("cannot open file \"" + fname
```

```
+ "\""); //error

	//file size
	struct stat st;
	if (fstat(m_fileDescr, &st) < 0) throw std::logic_error("cannot get
size of file \"" + fname + "\""); //error
	m_fileSize = (int)st.st_size;

	//mmap
	m_map = nullptr;
	Mmap(0);
}
File::~File()
{
	if (m_map) munmap((void*)m_map, m_chunkSize); //unmap chunk
	close(m_fileDescr); //close file
}

//mmap file
void File::Mmap(int pos)
{
	if (m_map)
	{
	msync((void*)m_map, m_chunkSize, MS_SYNC);
	munmap((void*)m_map, m_chunkSize); //unmap previous chunk
	}
	//setup diapason
	m_chunkPos = pos;
	m_chunkSize = std::min(m_maxChunkSize, m_fileSize - m_chunkPos);
	//mmap mode
	int mode = 0;
	if (m_canRead) mode |= PROT_READ;
	if (m_canWrite) mode |= PROT_WRITE;
	//mmap
	void *vmap = mmap(NULL, m_chunkSize, mode, MAP_SHARED, m_fileDescr,
m_chunkPos);
	if (vmap == MAP_FAILED) throw std::logic_error("cannot mmap file");
	m_map = (char*)vmap;
}

std::vector<std::pair<int,std::string>> File::Find(std::regex what) //find
ALL entrances of pattern
{
	//check if can read
	if (!m_canRead) throw std::logic_error("cannot read from file");

	std::vector<std::pair<int,std::string>> finds;

	int pos = 0;
	while (1)
	{
```

```
	auto match = *std::cregex_iterator(&m_map[pos], &m_map[m_chunkSize], what);
	if (match.size() == 0) break;

	finds.push_back(std::make_pair<int, std::string>(match.position(0) + pos, match[0].str()));
	pos += match.position(0) + match[0].str().size();
	}
	return finds;
}

bool File::CompareCharString(int off, std::string str)
{
	for (int i = 0; i < str.size(); i++) if (m_map[off + i] != str[i]) return false;
	return true;
}
void	File::Replace(std::string what, std::string with, std::function<Answer(int)> confirm) //replace
{
	//check if can read and write
	if (!m_canRead || !m_canWrite) throw std::logic_error("cannot read from and write to file");
	//strings must be same size
	if (what.size() != with.size()) throw std::logic_error("strings must be same length");

	//find "what" in file
	bool all = false, stop = false;
	for (int i = 0; i <= m_chunkSize - what.size() && !stop; i++)
	{
	//compare and confirm
	if (CompareCharString(i, what))
	{
		bool rep = true;
		if (!all)
		{
		Answer ans = confirm(i); //ask
		switch (ans)
		{
		case Answer::no:
			rep = false;
			break;
		case Answer::stop:
			rep = false;
			stop = true;
			break;
		case Answer::all:
			all = true;
			break;
		}
```

```
            }
            if (rep)
            {
            for (int j = 0; j < with.size(); j++) m_map[i + j] = with[j];
//replace
            i += with.size() - 1;
            }
      }
      }
}
std::string File::Print(int from, int to)
{
      //check if can read
      if (!m_canRead) throw std::logic_error("cannot read from file");
      //check diapason
      if (to < from || to < 0 || to >= m_chunkSize)
      throw std::logic_error("incorrect diapason [" + std::to_string(from) +
", " + std::to_string(to) + "]");

      //read from file
      std::string res;
      for (int i = from; i <= to; i++) res += m_map[i];
      return res;
}
void File::Write(int to, std::string what) //set diapason of chars
{
      //check if can write
      if (!m_canWrite) throw std::logic_error("cannot write to file");
      //check diapason
      if (to < 0 || to > m_chunkSize - what.size())
      throw std::logic_error("cannot fit string into chunk at this pos");

      //write to file
      for (int i = 0; i < what.size(); i++) m_map[to + i] = what[i];
}

//move chunk
void File::Move(int delta)
{
      SetPos(m_chunkPos + delta);
}
void File::SetPos(int pos)
{
      if (pos < 0 || pos >= m_fileSize) throw std::logic_error("moving
outside of file");

      int ps = getpagesize();
      if (pos % ps) throw std::logic_error("Position MUST be multiple of page
size: " + std::to_string(ps));

      Mmap(pos);
```

```
}

std::pair<int, int> File::Diapason()
{
	return  std::make_pair<int,  int>(std::move(m_chunkPos),  m_chunkPos  +
m_chunkSize - 1);
}
std::string File::Mode()
{
	if (!m_canRead) return "Write only";
	if (!m_canWrite) return "Read only";
	return "Read and write";
}
int File::FileSize()
{
	return m_fileSize;
}
```

main.cpp:

```
#include <iostream>
#include <functional>
#include <sstream>
#include <vector>
#include <map>

#include "file.h"

using args = std::vector<std::string>;

Answer Ask(int off)
{
	std::string c;
	std::cout << "Replace at offset " << off << "? (y/n/s/a): ";
	std::getline(std::cin, c);

	switch (std::tolower(c[0]))
	{
	case 'y': return Answer::yes;
	case 'a': return Answer::all;
	case 's': return Answer::stop;
	}
	return Answer::no;
}
void help(args)
{
	std::cout << "List of commands:" << std::endl <<
	" quit -- exit program" << std::endl <<
	" help -- shows this text" << std::endl <<
	" setpos P -- set position of chunk in file to P. P mist be multiple of
page size" << std::endl <<
	" move D -- moves chunk by D chars. D must be multiple of page size" <<
std::endl <<
```

```
	" diap -- show chunk borders" << std::endl <<
	" mode -- show file editing mode" << std::endl <<
	" size -- show file size" << std::endl <<
	" print A B -- print chars from A position in chunk to B" << std::endl
<<
	" write A Str -- write string Str to position A" << std::endl <<
	" find -- find regular expression in file" << std::endl <<
	" replace -- replace one string with another. Strings must be same
length" << std::endl;
}
unsigned int toUint(std::string s)
{
	//check if is int
	if (s.size() == 0) throw std::logic_error("not a positive integer");
//empty
	for (int i = 0; i < s.size(); i++)
	if (s[i] < '0' || s[i] > '9') throw std::logic_error("not a positive
integer");
	return atoi(s.c_str()); //convert if is int
}

int main(int argc, char *argv[])
{
	//args: ./a.out filename mode=rw size=15
	std::string filename;
	bool canRead = true, canWrite = true;
	int size = 15;

	//get args
	switch (argc)
	{
	case 4:
	size = atoi(argv[3]);
	if (size <= 0)
	{
		std::cout << "Size must be positive" << std::endl;
		return 3;
	} //break must apsent here
	case 3:
	{
	std::string strmode = argv[2];
	if (strmode != "rw")
	{
		if (strmode == "w") canRead = false;
		else if (strmode == "r") canWrite = false;
		else
		{
		std::cout << "Available modes:" << std::endl <<
			"r -- read only" << std::endl <<
			"w -- write only" << std::endl <<
			"rw -- read and write" << std::endl;
```

```
            return 2;
            }
      }
      } //and here
      case 2:
      filename = argv[1];
      break; //but here must be

      default:
      std::cout << "Usage: " << argv[0] << " filename [mode=rw [size=15]]" <<
std::endl;
      return 1;
      }

      //std::cout << "ARGS: " << filename << " " << (int)mode << " " << size
<< std::endl;
      try
      {
      File file(filename, canRead, canWrite, size); //open file

      std::map<std::string,  std::function<void(args)>>  functions;  //user
functions
      std::map<std::string, unsigned int> argsCount; //functions arguments
count

      bool cycle = true; //loop variable
      std::string input; //contains user input

      //UI functions
      argsCount["quit"] = 0;
      functions["quit"] = [&cycle](args){ cycle = false; };
      argsCount["help"] = 0;
      functions["help"] = help;
      //file functions
      argsCount["setpos"] = 1;
      functions["setpos"] = [&file](args a){ file.SetPos(toUint(a[0])); };
      argsCount["move"] = 1;
      functions["move"] = [&file](args a){ file.Move(toUint(a[0])); };
      argsCount["diap"] = 0;
      functions["diap"] = [&file](args)
      {
            auto a = file.Diapason();
            std::cout << "Diapason: [" << a.first << ", " << a.second << "]"
<< std::endl;
      };
      argsCount["mode"] = 0;
      functions["mode"] = [&file](args)
      {
            std::cout << "Mode: " << file.Mode() << std::endl;
      };
      argsCount["size"] = 0;
```

```
	functions["size"] = [&file](args)
	{
		std::cout << "File size: " << file.FileSize() << std::endl;
	};

	argsCount["print"] = 2;
	functions["print"] = [&file](args a){ std::cout << file.Print(toUint(a[0]), toUint(a[1])) << std::endl; };
	argsCount["write"] = 2;
	functions["write"] = [&file](args a){ file.Write(toUint(a[0]), a[1]); };

	argsCount["find"] = 0;
	functions["find"] = [&file](args a)
	{
		std::string pattern;
		std::cout << "Find what: "; std::getline(std::cin, pattern);

		auto v = file.Find(std::regex(pattern));
		for (std::pair<int, std::string> p : v)
		std::cout << "Found " << p.second << " at offset " << p.first << std::endl;
	};

	argsCount["replace"] = 0;
	functions["replace"] = [&file](args)
	{
		std::string what, with;
		std::cout << "Replace what: "; std::getline(std::cin, what);
		std::cout << "Replace with: "; std::getline(std::cin, with);

		file.Replace(what, with, Ask);
	};

	while (cycle)
	{
		std::cout << "> ";
		std::getline(std::cin, input); //read uer input

		//split string
		std::stringstream ss(input);
		std::vector<std::string> words;
		for (std::string s; ss >> s; ) words.push_back(s); //split into words
		if (words.size() == 0) continue;

		auto iter = functions.find(words[0]);
		if (iter == functions.end()) //not a command in map
		{
		std::cout << "unknowm command \"" << words[0] << "\"" << std::endl;
```

```
            continue;
            }
            //check arguments
            unsigned int argCount = argsCount[words[0]];
            if (argCount != words.size() - 1)
            {
            std::cout << words[0] << " has " << argCount << " arguments" <<
std::endl;
            continue;
            }
            //execute
            auto com = (*iter).second;
            words.erase(words.begin());

            try
            {
            com(words);
            std::cout << "OK." << std::endl;
            }
            catch(std::exception &e)
            {
            std::cout << "ERROR: " << e.what() << std::endl;
            }
      }
      }
      catch (std::exception &e)
      {
      std::cout  <<  "ERROR:  "  <<  e.what()  <<  std::endl;  //error  in
initialisation
      return 4;
      }
      return 0;
}
```

Makefile:

```
all: main.o file.o
    g++ main.o file.o -o kp.out

main.o: main.cpp file.h
    g++ -c main.cpp

file.o: file.cpp file.h
    g++ -c file.cpp
```

6. Выводы:

Научился создавать простые текстовые процессоры, использующие технологию file mapping и осуществляющие поиск регулярных выражений.