`

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

**Тема работы**

**“Графический редактор”**

Студент: Зубко Дмитрий Валерьевич

Группа: М8О-208Б-20

Вариант:

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/usernameMAI/OS

**Постановка задачи**

Сделать текстовый редактор наподобие nano, который позволяет редактировать файлы в реальном времени с поддержкой следующих функций:

1. Обычное сохранение и редактирование файла.
2. Поиск строк в файле. Если найденных строк в файле несколько, то при нажатии какой-нибудь клавиши курсор переносится к следующей найденной строке.
3. Подсветка ключевых слов языков C и C++, если открыт файл с расширением .c или .cpp соответственно.

**Общие сведения о программе**

Содержит файл main.cpp.

Содержит makefile:

make:

g++ main.cpp -lncurses -o red

clean:

rm -rf red

**Общий метод и алгоритм решения**

Включаем raw режим. Открываем файл и отображаем его в память. Считываем из памяти данные и записываем их в вектор. Выводим вектор на экран при каждом обновлении.

**Исходный код**

#include <iostream>  
#include <string>  
#include <ncurses.h>  
#include <termios.h>  
#include <ctype.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <vector>  
#include <unistd.h>  
#include <stdlib.h>  
#include<sys/mman.h>  
#include<algorithm>  
  
#define CTRL\_KEY(k) ((k) & 0x1f)  
  
struct config{  
 std::vector<std::string> word\_pluses = {"if", "else", "while", "bool", "int", "char", "string", "struct", "exit", "do"};  
 int row = 0, col = 0, cx = 0, cy = 0, fd, numrows = 0, rowoff = 0, coloff = 0;  
 bool saved = false;  
 bool find\_s = false;  
 bool it\_is\_plus\_plus\_file = false;  
 char\* memptr;  
 std::vector<std::string> rows;  
 struct stat st;  
 std::string file\_name;  
 std::vector<std::pair<int,int> > v;  
 int num\_find\_str = 1;  
 std::string find\_str = " ";  
};  
  
  
config info;  
  
void die(std::string s) {  
  
 perror(s.c\_str());  
 exit(1);  
}  
  
struct termios orig\_termios;  
  
void DisableRawMode(){  
 munmap((void\*)info.memptr, info.st.st\_size);  
 tcsetattr(STDIN\_FILENO, TCSAFLUSH, &orig\_termios);  
 endwin();  
}  
  
void EnableRawMode(){  
 initscr ();  
 raw();  
 noecho();  
 halfdelay(3);  
 keypad(stdscr, true);  
  
 tcgetattr(STDIN\_FILENO, &orig\_termios);  
 struct termios raw = orig\_termios;  
 raw.c\_lflag &= ~(ICANON | ISIG);  
  
 tcsetattr(STDIN\_FILENO, TCSAFLUSH, &raw);  
  
 atexit(DisableRawMode);  
}  
  
void GetInfo(){  
 getmaxyx(stdscr, info.row, info.col);  
 info.row -= 2;  
}  
  
void RefreshScreen() {  
  
 if (info.cy < info.rowoff) {  
 info.rowoff = info.cy;  
 }  
  
 if (info.cy >= info.rowoff + info.row) {  
 info.rowoff = info.cy - info.row + 1;  
 }  
  
 if (info.cx < info.coloff) {  
 info.coloff = info.cx;  
 }  
  
 if (info.cx >= info.coloff + info.col) {  
 info.coloff = info.cx - info.col + 1;  
 }  
  
 std::string s;  
  
 s += "\x1b[?25l";  
 s += "\x1b[H";  
  
 for (int i = 0; i < info.row; ++i) {  
  
 int filerow = i + info.rowoff;  
  
 std::string s1;  
  
 if (info.coloff <= info.rows[filerow].size()) {  
 if (filerow < info.rows.size())  
 s1 = info.rows[filerow].substr(info.coloff);  
 }  
  
 if (s1.size() > info.col)  
 s1 = s1.substr(0, info.col);  
  
 if (!info.it\_is\_plus\_plus\_file) {  
 s += s1;  
 } else {  
 std::string s2;  
 for (int j = 0; j < s1.size(); ++j) {  
 if (std::isalpha(s1[j])) {  
 s2 += s1[j];  
 } else {  
 if (std::count(info.word\_pluses.begin(), info.word\_pluses.end(), s2)) {  
 s += "\x1b[31m" + s2 + "\x1b[39m";  
 } else {  
 s += s2;  
 }  
 s += s1[j];  
 s2.clear();  
 }  
 }  
 if (s2.size()) {  
 if (std::count(info.word\_pluses.begin(), info.word\_pluses.end(), s2)) {  
 s += "\x1b[31m" + s2 + "\x1b[39m";  
 } else {  
 s += s2;  
 }  
 }  
 }  
 s += "\x1b[K";  
 s += "\r\n";  
  
 }  
  
 for (int i = info.row; i < info.row + 2; ++i) {  
 if (i == info.row) {  
 for (int j = 0; j < info.col; ++j)  
 s += "~";  
 s += "\r\n";  
 } else {  
  
 }  
 }  
  
  
 s += ("\x1b[" + std::to\_string(info.cy - info.rowoff + 1) + ";" + std::to\_string(info.cx - info.coloff + 1) + "H");  
 s += "\x1b[?25h";  
  
 write(STDOUT\_FILENO, s.c\_str(), s.size());  
  
 int yy, xx;  
 yy = info.rowoff + info.cy + 1;  
 xx = info.cx + info.coloff + 1;  
 if (info.rowoff) yy -= info.rowoff;  
 if (info.coloff) xx -= info.coloff;  
  
 std::string s3 = " Line " + std::to\_string(info.cy + 1) + " , Column " + std::to\_string(info.cx + 1);  
 s += ("\x1b[" + std::to\_string(info.row + 2) + ";" + std::to\_string(info.col - s3.size() - 2) + "H") + s3;  
 s += ("\x1b[" + std::to\_string(info.cy - info.rowoff + 1) + ";" + std::to\_string(info.cx - info.coloff + 1) + "H");  
  
 std::string s4; // find string  
 s4 = " Find string: " + info.find\_str;  
 s += ("\x1b[" + std::to\_string(info.row + 2) + ";" + std::to\_string(info.col - s3.size() - info.col / 2) + "H") +  
 s4;  
 s += ("\x1b[" + std::to\_string(info.cy - info.rowoff + 1) + ";" + std::to\_string(info.cx - info.coloff + 1) + "H");  
  
 std::string s2;  
 if (!info.saved) {  
 s2 = " Not saved ";  
 } else {  
 s2 = " Saved ";  
 }  
  
 s += ("\x1b[" + std::to\_string(info.row + 2) + ";" + std::to\_string(2) + "H") + s2;  
 s += ("\x1b[" + std::to\_string(info.cy - info.rowoff + 1) + ";" + std::to\_string(info.cx - info.coloff + 1) + "H");  
  
 write(STDOUT\_FILENO, s.c\_str(), s.size());  
  
  
// mvaddstr(info.row + 1, info.col - s3.size() - 2, s3.c\_str());  
}  
  
void InsertChar(int c){  
 if(info.cy >= info.rows.size())  
 info.rows.emplace\_back("");  
  
 if(info.cx < info.rows[info.cy].size()){  
 info.rows[info.cy] = info.rows[info.cy].substr(0, info.cx + info.coloff + 1) +  
 char(c) + info.rows[info.cy].substr(info.cx + 1);  
 }  
 else{  
 info.rows[info.cy] = info.rows[info.cy].substr(0, info.cx + 1) + char(c);  
 }  
  
// if(info.cx < info.rows[info.cy + info.rowoff].size())  
// info.rows[info.cy + info.rowoff] = info.rows[info.cy + info.rowoff].substr(0, info.cx + info.coloff + 1) +  
// char(c) + info.rows[info.cy + info.rowoff].substr(info.cx + info.coloff + 1);  
// else  
// info.rows[info.cy + info.rowoff] = info.rows[info.cy + info.rowoff].substr(0, info.cx + info.coloff + 1) +  
// char(c);  
}  
  
void backspace() {  
 if (info.cx == 0 && info.cy == 0)  
 return;  
  
 if (info.cx != 0) {  
 info.rows[info.cy] = info.rows[info.cy].substr(0, info.cx - 1)  
 + info.rows[info.cy].substr(info.cx);  
 }  
 else{  
  
 if(info.cy >= info.rows.size())  
 return;  
  
 if(info.rows[info.cy].size()) {  
 info.rows[info.cy - 1] += info.rows[info.cy];  
 }  
  
 std::vector<std::string> v2(info.rows.size() - 1);  
 for(int i = 0; i < info.cy ; ++i)  
 v2[i] = info.rows[i];  
 for(int i = info.cy ; i < info.rows.size() - 1; ++i)  
 v2[i] = info.rows[i + 1];  
 info.rows = v2;  
 }  
}  
  
void func\_enter() {  
 info.saved = false;  
 std::vector<std::string> v2(info.rows.size() + 1);  
  
 for (int i = 0; i < info.cy + 1; ++i)  
 v2[i] = info.rows[i];  
  
 if (info.cy + 1 > info.rows.size()) {  
 info.rows.emplace\_back("");  
 } else {  
 v2[info.cy + 1] = "";  
  
 for (int i = info.cy + 1; i < info.rows.size(); ++i)  
 v2[i + 1] = info.rows[i];  
  
 info.rows = v2;  
 }  
  
 if(info.cx < info.rows[info.cy].size()){  
 std::string str1 = info.rows[info.cy].substr(0, info.cx);  
 std::string str2 = info.rows[info.cy].substr(info.cx);  
  
 info.rows[info.cy] = str1;  
 info.rows[info.cy + 1] += str2;  
 }  
  
 if (info.cy < info.rows.size())  
 ++info.cy;  
}  
  
void func\_ctrl\_s(){  
 info.saved = true;  
 FILE \*f = fopen(info.file\_name.c\_str(), "w");  
 if(f == NULL) die("Save file");  
  
 for(int i = 0; i < info.rows.size(); ++i){  
 fprintf(f, "%s\n", info.rows[i].c\_str());  
 }  
  
// fclose(f);  
}  
  
void escape(){  
  
}  
  
void func\_ctrl\_f() {  
  
 std::string s;  
 std::cin >> s;  
 info.num\_find\_str = 0;  
 info.v.clear();  
  
 std::string add\_str\_new;  
 for(int i = 0; i < info.rows.size(); ++i){  
 for(int j = 0; j < info.rows[i].size(); ++j){  
 if(info.rows[i].size() - 1 != j && info.rows[i][j] != '\0' && info.rows[i][j] != '\n'  
 && info.rows[i][j] != ' ' && info.rows[i][j] != '\r'){  
// info.find\_str += info.rows[i][j];  
 add\_str\_new += info.rows[i][j];  
 }  
 else{  
 if(!isspace(info.rows[i][j]))  
 add\_str\_new += info.rows[i][j];  
 if(add\_str\_new == s) {  
 info.v.push\_back(std::make\_pair(i, j));  
 }  
 add\_str\_new = "";  
 }  
 }  
 }  
  
 info.find\_str = s + ' ' + std::to\_string(info.v.size()) + " ";  
// + ' ' + std::to\_string(info.v[0].first) +  
// ' ' + std::to\_string(info.v[0].second);  
 info.find\_s = true;  
  
// std::vector<std::pair<int, int>> v;  
// for (int i = 0; i < info.rows.size(); ++i) {  
// for (int j = 0; j < info.rows[i].size(); ++j) {  
// if (info.rows[i][j] != ' ' && info.rows[i][j] != '\n' && info.rows[i][j] != '\0') {  
// s2 += info.rows[i][j];  
// }  
// if (s2 == s) {  
// v.emplace\_back(i, j);  
// s2 = "";  
// }  
// }  
// }  
  
}  
  
void GetPress() {  
 int c = wgetch(stdscr);  
  
 bool y\_move = info.cy < info.rows.size();  
  
 if(info.find\_s){  
 if(c == 27){ // esc  
 info.find\_s = false;  
 info.find\_str = " ";  
 }  
 if(c == CTRL\_KEY('g')){  
 if(info.num\_find\_str < info.v.size()) {  
 info.cy = info.v[info.num\_find\_str].first;  
 info.cx = info.v[info.num\_find\_str++].second;  
 if(info.rows[info.cy].size() - 1 == info.cx) ++info.cx;  
 }  
 }  
 }  
 else {  
 switch (c) {  
 case CTRL\_KEY('q'):  
 exit(0);  
 break;  
 case KEY\_DOWN:  
 if (info.cy < info.rows.size() - 1)  
 ++info.cy;  
 break;  
 case KEY\_UP:  
 if (info.cy != 0)  
 --info.cy;  
 break;  
 case KEY\_RIGHT:  
 if (y\_move && info.cx < info.rows[info.cy].size()) {  
 ++info.cx;  
 } else if (y\_move && info.cx == info.rows[info.cy].size()) {  
 ++info.cy;  
 info.cx = 0;  
 }  
  
 break;  
 case KEY\_LEFT:  
 if (info.cx != 0)  
 --info.cx;  
 else if (info.cy > 0) {  
 --info.cy;  
 info.cx = info.rows[info.cy].size();  
 }  
 break;  
 case KEY\_BACKSPACE:  
 info.saved = false;  
 backspace();  
 if (info.cx != 0)  
 --info.cx;  
 else if (info.cy > 0) {  
 --info.cy;  
 info.cx = info.rows[info.cy].size();  
 }  
 break;  
 case CTRL\_KEY('s'):  
 func\_ctrl\_s();  
 break;  
 case 10: // enter  
 func\_enter();  
 break;  
 case CTRL\_KEY('f'):  
 func\_ctrl\_f();  
 break;  
 case 27: // ESC  
 info.find\_s = false;  
 info.find\_str = " ";  
 break;  
 default:  
 if (!iscntrl(c) && c != ERR) {  
 InsertChar(c);  
 info.saved = false;  
 if (info.cx == info.rows[info.cy].size() && info.cy != info.numrows) {  
 ++info.cy;  
 info.cx = 0;  
 info.coloff = 0;  
 }  
 if (info.cx < info.rows[info.cy].size())  
 ++info.cx;  
 }  
 }  
 }  
 if(info.cx > info.rows[info.cy].size()) {  
 info.cx = info.rows[info.cy].size();  
 }  
  
// if(info.cx >= info.col) {  
// info.coloff += info.cx - info.col;  
// info.cx = info.col;  
// }  
// if(info.cy >= info.row) {  
// info.rowoff += info.cy - info.row;  
// info.cy = info.row;  
// }  
  
}  
  
void InitColors(){  
 start\_color(); // Инициализация цветов  
  
 if (has\_colors() && COLOR\_PAIRS >= 13) {  
 init\_pair(1, COLOR\_RED, COLOR\_BLACK);  
 init\_pair(2, COLOR\_GREEN, COLOR\_BLACK);  
 init\_pair(3, COLOR\_YELLOW, COLOR\_BLACK);  
 init\_pair(4, COLOR\_BLUE, COLOR\_BLACK);  
 init\_pair(5, COLOR\_MAGENTA, COLOR\_BLACK);  
 init\_pair(6, COLOR\_CYAN, COLOR\_BLACK);  
 init\_pair(7, COLOR\_BLUE, COLOR\_WHITE);  
 init\_pair(8, COLOR\_WHITE, COLOR\_RED);  
 init\_pair(9, COLOR\_BLACK, COLOR\_GREEN);  
 init\_pair(10, COLOR\_BLUE, COLOR\_YELLOW);  
 init\_pair(11, COLOR\_WHITE, COLOR\_BLUE);  
 init\_pair(12, COLOR\_WHITE, COLOR\_MAGENTA);  
 init\_pair(13, COLOR\_BLACK, COLOR\_CYAN);  
 }  
  
 color\_set(11, NULL);  
}  
  
int main(int argc, char \*argv[]) {  
  
 EnableRawMode();  
 GetInfo();  
  
 if (argc >= 2) {  
 info.file\_name = argv[1];  
 info.fd = open(argv[1], O\_RDWR);  
 if (info.fd == -1)  
 die("Open File");  
  
 if (fstat(info.fd, &info.st)) {  
 die("Error fstat");  
 }  
  
 std::string make\_magick = argv[1];  
 if(make\_magick[make\_magick.size() - 1] == 'c' && make\_magick[make\_magick.size() - 2] == '.'){  
 info.it\_is\_plus\_plus\_file = true;  
 }  
  
 info.memptr = (char \*) mmap(nullptr, info.st.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, info.fd, 0);  
 if (info.memptr == MAP\_FAILED) die("Error in file mapping");  
  
 for(int i = 0; i < info.st.st\_size; ++i){  
 if(info.memptr[i] == '\n'){  
 info.rows.push\_back("");  
 }  
 else{  
 if(info.rows.empty()){  
 info.rows.push\_back("");  
 }  
 info.rows[info.rows.size() - 1] += info.memptr[i];  
 }  
 }  
 close(info.fd);  
  
 while (true) {  
 RefreshScreen();  
 GetPress();  
 }  
  
 } else {  
 char c;  
  
 while (true) {  
 char c = '\0';  
  
 if (read(STDIN\_FILENO, &c, 1) == -1 && errno != EAGAIN) die("read");  
  
 if (iscntrl(c)) {  
 printf("%d\r\n", c);  
 } else {  
 printf("%d ('%c')\r\n", c, c);  
 }  
 if (c == CTRL\_KEY('q')) break;  
 }  
  
 }  
}

**Демонстрация работы программы**





**Выводы**Данный курсовой проект познакомил меня с работой терминала. Я закрепил свои знания о memory mapping. Узнал о режимах работы терминала и создал свой текстовый редактор.