**ИТМО Кафедра Информатики и прикладной математики**

Отчет по лабораторной работе №4

«Управление процессами»

**Выполнил: студент группы P3217**

**Плюхин Дмитрий**

**Преподаватель: Зыков А. Г.**

**2017 год**

1. **Задание**

Разработать алгоритм и программу, реализующую следующие действия:

Сгенерировать (или использовать из предыдущих заданий) массив данных, который будет использоваться для сортировки и поиска в нём элементов.

Создать три отдельных процесса, в каждом из которых:

a) Отсортировать исходный массив по заданному в командной строке шаблону.

b) Осуществить поиск элемента, указанного также в командной строке, в отсортированном массиве.

Вывести временные характеристики и ID процессов.

Определить количество открытых дескрипторов.

1. **Листинг основной части программы**

Файл main.cpp содержит функцию, получающую на вход четыре строковые константы : имя файла для сортировки, шаблон сортировки, имя файла для поиска, строка для поиска. Функция позволяет отсортировать предложения в текстовом файле, а затем выполнить поиск по какому-либо шаблону и вывести на печать предложения в отсортированном порядке, содержащие шаблон, по которому производится поиск. Далее приведены ключевые функции из файла main.cpp.

int sort(const char\* fileName, const char\* sortTemplate){

char sortedFileName[PATH\_MAX\_LEN];

strncpy(sortedFileName,fileName,PATH\_MAX\_LEN);

strcat(sortedFileName,".srtd");

CopyFile(fileName, sortedFileName, FALSE);

HANDLE hFile = CreateFile(fileName,GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, 0, NULL);

DWORD fileSize = GetFileSize(hFile,NULL);

HANDLE hMap = CreateFileMapping(hFile, NULL, PAGE\_READWRITE, 0, fileSize, NULL);

char\* mappedFile = (char\*)MapViewOfFile(hMap, FILE\_MAP\_ALL\_ACCESS, 0, 0, fileSize);

//replaceSpaces(mappedFile,fileSize);

char\* mappedFileNull = mappedFile;

HANDLE hSentencesHeap = HeapCreate(HEAP\_GENERATE\_EXCEPTIONS | HEAP\_NO\_SERIALIZE, HEAP\_SIZE\_DEFAULT, 0);

DWORD numOfSentences = getNumOfSentences(mappedFile, fileSize);

PSENTENCE sentences = (PSENTENCE)HeapAlloc(hSentencesHeap, HEAP\_ZERO\_MEMORY, numOfSentences\*sizeof(SENTENCE));

DWORD index = 0;

DWORD accumulator = 0;

for (int i = 0; i < numOfSentences; i++){

(sentences+i)->pointer = mappedFile + accumulator;

(sentences+i)->length = getSentenceLength(mappedFile + accumulator);

accumulator += (sentences+i)->length;

}

if (strcmp(sortTemplate,"asc")==0){

qsort(sentences, numOfSentences, sizeof(SENTENCE), Compare);

} else if (strcmp(sortTemplate,"desc")==0){

qsort(sentences, numOfSentences, sizeof(SENTENCE), CompareDesc);

}

HANDLE hFileS = CreateFile(sortedFileName,GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, 0, NULL);

DWORD fileSizeS = GetFileSize(hFileS,NULL);

HANDLE hMapS = CreateFileMapping(hFileS, NULL, PAGE\_READWRITE, 0, fileSizeS, NULL);

char\* mappedFileS = (char\*)MapViewOfFile(hMapS, FILE\_MAP\_ALL\_ACCESS, 0, 0, fileSizeS);

char\* mappedFileNullS = mappedFileS;

accumulator = 0;

for (int i = 0; i < numOfSentences; i++){

strncpy(mappedFileS+accumulator,(sentences+i)->pointer,(sentences+i)->length);

accumulator += (sentences+i)->length;

}

UnmapViewOfFile(mappedFileS);

CloseHandle(hMapS);

CloseHandle(hFileS);

UnmapViewOfFile(mappedFile);

CloseHandle(hMap);

CloseHandle(hFile);

return 0;

}

int search(const char\* fileName, const char\* searched){

char tmpFileName[PATH\_MAX\_LEN];

strncpy(tmpFileName,fileName,PATH\_MAX\_LEN);

strcat(tmpFileName,".tmp");

DWORD searchedLen = strlen(searched);

CopyFile(fileName, tmpFileName, FALSE);

HANDLE hFile = CreateFile(tmpFileName,GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, 0, NULL);

DWORD fileSize = GetFileSize(hFile,NULL);

HANDLE hMap = CreateFileMapping(hFile, NULL, PAGE\_READWRITE, 0, fileSize, NULL);

char\* mappedFile = (char\*)MapViewOfFile(hMap, FILE\_MAP\_ALL\_ACCESS, 0, 0, fileSize);

char\* mappedFileNull = mappedFile;

mappedFile[fileSize - 1] = 0;

for (DWORD i = 0; i < fileSize; i++){

if (check(mappedFile+i,searched)){

showSentence(mappedFile+i,fileSize-i-1,i);

i += searchedLen - 1;

}

}

UnmapViewOfFile(mappedFile);

CloseHandle(hMap);

CloseHandle(hFile);

return 0;

}

int main(int argc, char\* argv[]){

setlocale(LC\_ALL,"Russian");

if (argc != 5){

cout << "Usage : main name\_of\_file\_for\_sorting template\_for\_sorting name\_of\_file\_for\_searching str\_for\_searching" << endl;

return 1;

}

sort(argv[1],argv[2]);

search(argv[3],argv[4]);

}

Файл launcher.cpp содержит код запуска трех процессов для сортировки внешнего файла, поиска в отсортированном файле и вывода на печать ID процессов и их временных характеристик по завершении их выполнения, а также количество открытых дескрипторов.

int showTime(const char\* overture, LPSYSTEMTIME systemTime){

cout << overture << systemTime->wHour << ":" << systemTime->wMinute << ":" << systemTime->wSecond << ":" << systemTime->wMilliseconds;

return 0;

}

int main(int argc, char\* argv[]){

setlocale(LC\_ALL,"Russian");

HANDLE\* processHandles;

DWORD\* processIds;

DWORD numOfOpenedHandles;

processHandles = (HANDLE\*)malloc(3\*sizeof(HANDLE));

processIds = (DWORD\*)malloc(3\*sizeof(DWORD));

STARTUPINFO startUpInfo;

FILETIME creationFileTime;

SYSTEMTIME creationSystemTime;

FILETIME exitFileTime;

SYSTEMTIME exitSystemTime;

FILETIME userFileTime;

SYSTEMTIME userSystemTime;

FILETIME kernelFileTime;

SYSTEMTIME kernelSystemTime;

SECURITY\_ATTRIBUTES StdOutSA ={sizeof(SECURITY\_ATTRIBUTES), NULL, TRUE};

HANDLE hFileLog = CreateFile("log.out",GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, &StdOutSA, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

PROCESS\_INFORMATION processInformationMorning;

PROCESS\_INFORMATION processInformationEvening;

PROCESS\_INFORMATION processInformationDay;

const char\* commandLineMorning = "main data/lullaby.txt asc data/lullaby.txt.srtd утро";

const char\* commandLineDay = "main data/ghosts.txt desc data/ghosts.txt.srtd ночь";

const char\* commandLineEvening = "main data/widow.txt asc data/widow.txt.srtd вечер";

GetStartupInfo(&startUpInfo);

startUpInfo.dwFlags = STARTF\_USESTDHANDLES;

startUpInfo.hStdOutput = hFileLog;

CreateProcess(NULL, (LPSTR)commandLineMorning, NULL, NULL, TRUE, 0, NULL, NULL, &startUpInfo, &processInformationMorning);

CreateProcess(NULL, (LPSTR)commandLineDay, NULL, NULL, TRUE, 0, NULL, NULL, &startUpInfo, &processInformationDay);

CreateProcess(NULL, (LPSTR)commandLineEvening, NULL, NULL, TRUE, 0, NULL, NULL, &startUpInfo, &processInformationEvening);

processHandles[0] = processInformationMorning.hProcess;

processHandles[1] = processInformationDay.hProcess;

processHandles[2] = processInformationEvening.hProcess;

processIds[0] = processInformationMorning.dwProcessId;

processIds[1] = processInformationDay.dwProcessId;

processIds[2] = processInformationEvening.dwProcessId;

WaitForMultipleObjects(3,processHandles,TRUE,INFINITE);

for (int i = 0; i < 3; i++){

cout << "Process " << processIds[i];

GetProcessTimes(processHandles[i],&creationFileTime,&exitFileTime,&kernelFileTime,&userFileTime);

FileTimeToSystemTime(&creationFileTime,&creationSystemTime);

FileTimeToSystemTime(&exitFileTime,&exitSystemTime);

FileTimeToSystemTime(&userFileTime,&userSystemTime);

FileTimeToSystemTime(&kernelFileTime,&kernelSystemTime);

cout << showTime(" was created at ",&creationSystemTime);

cout << showTime(", was exited at ",&exitSystemTime);

cout << showTime(", so it was handling by kernel in ",&kernelSystemTime);

cout << showTime(", and by user in ",&userSystemTime) << endl;

GetProcessHandleCount(processHandles[i],&numOfOpenedHandles);

cout << "In the process " << numOfOpenedHandles << " descriptors were opened" << endl;

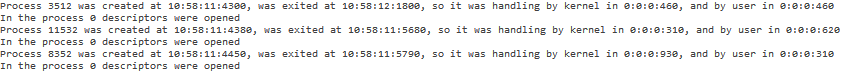
}

CloseHandle(hFileLog);

}

**3. Результаты работы программы**

В результате работы программы получены следующие числовые характеристики:



1. **Вывод**

Таким образом, механизм управления процессами в среде Windows не представляет особенной сложности : процесс создается при помощи функции CreateProcess, множество параметров которой обеспечивают исключительную гибкость, например, можно произвольно установить поток ввода или вывода, а также настроить параметры командной строки. Функция GetProcessTimes позволяет получить временные характеристики процесса, однако неудобство составляет необходимость ручной конвертации возвращаемых данных в более удобный для манипуляции формат SYSTEMTIME.