СПб НИУ ИТМО

кафедра ИПМ

Системное программное обеспечение

Лабораторная работа № 3

Управление памятью

Работу выполнил:

Студент II курса

Группы № 2120

Журавлев Виталий

Санкт-Петербург

2014 г.

**Цель работы:**

Необходимо реализовать алгоритм сортировки. Вариант соответствует варианту по предмету Комбинаторные алгоритмы.

Память для выполнения программы необходимо организовать с помощью:

1. Куч
2. Отображаемых файлов
3. Базовых указателей

**Теоритические основы:**

Управление динамической памятью в той или иной форме требуется в большинстве программ. Необходимость в этом возникает всякий раз, когда требуется создавать структуры данных, размер которых не может быть определен заранее на стадии создания программы. Типичными примерами динамических структур данных могут служить деревья поиска, таблицы имен и связанные списки.

В Windows предусмотрены гибкие механизмы управления динамической памятью программы. Кроме того, Windows предоставляет средства отображения файлов, которые позволяют ассоциировать файл непосредственно с виртуальным адресным пространством процесса.

***Куча***

Windows поддерживает пулы памяти, называемые *кучами* (heaps). Процесс может иметь несколько куч, которые используются для распределения памяти. Во многих случаях одной кучи вполне достаточно, но в силу ряда причин, о которых будет сказано ниже, иногда целесообразно иметь в своем распоряжении несколько куч. Если одной кучи вам хватает, можно обойтись использованием функций управления памятью, предоставляемых библиотекой С (malloc, free, calloc, realloc). Кучи являются объектами Windows и, следовательно, имеют дескрипторы. Дескриптор кучи используется при распределении памяти. У каждого процесса имеется куча, заданная по умолчанию, которую использует функция malloc.

***Отображение файлов***

Динамическая память, распределенная в кучах, должна физически размещаться в файле подкачки. Управление перемещением страниц между физической памятью и файлом подкачки, а также отображением файла подкачки на виртуальное адресное пространство процесса осуществляется средствами ОС, ответственными за управление памятью. По завершении выполнения процесса физическое пространство в этом файле освобождается. Сначала необходимо создать для открытого файла *объект отображения файла* (file mapping object), у которого имеется дескриптор, а затем отобразить этот файл или только некоторую его часть на виртуальное адресное пространство процесса.

***Базовые указатели***

Во многих случаях метод отображения файлов является весьма удобным. Однако предположим, что в программе создается структура данных с указателями, ссылающимися на область отображения файла, и ожидается, что впоследствии к этому файлу будет производиться обращение. В этом случае указатели оказываются установленными относительно виртуального адреса, и не будут иметь смысла при использовании представления объекта отображения в следующий раз. Решение состоит в том, чтобы использовать базовые указатели (based pointers), являющиеся фактически смещениями относительно другого указателя.

***Сортировка***

В качестве сортировки использовалась функция sort (first, last) из стандартной библиотеки C++ <algorithm>. Функция sort сортирует контейнер на полуинтервале

[first, last) с помощью библиотечной сортировки IntroSort. Данная сортировка гарантирует наилучшее время сортировки. Использование компаратора дает возможность сортировать элементы для которых по умолчанию не определены операции сравнения или сортировать элементы в порядке отличном от стандартного.

**Код программы:**

#include "stdafx.h"

#include "time.h"

#include "algorithm"

#include "string"

#include "iostream"

#include "fstream"

#include "windows.h"

using namespace std;

void SortHeaps()

{

 ifstream ifs("Heaps.txt", ifstream::binary);

 filebuf\* pbuf = ifs.rdbuf();

 ifs.seekg(0, ios::end);

 size\_t size = ifs.tellg();

 ifs.seekg(0, ios::beg);

 pbuf->pubseekpos(0, ifs.in);

 HANDLE hHeap = HeapCreate(0, 0x01000, 0);

 char \*strName = (char\*)HeapAlloc(hHeap, 0, size);

 pbuf->sgetn(strName, size);

 ifs.close();

 sort(strName, strName + size);

 ofstream ofs("Heaps.txt", ifstream::binary);

 for (int i = 0; i < size; ++i)

 {

 ofs << strName[i];

 }

 HeapFree(hHeap, 0, strName);

 ofs.close();

}

void SortMappingFiles()

{

 string fname = "Mapping Files.txt";

 wstring wstr(fname.begin(), fname.end());

 LPCWSTR str = (LPCWSTR)wstr.c\_str();

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

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

 DWORD dwFileSize = GetFileSize(hFile, NULL);

 char \*memory = (char\*)MapViewOfFile(hMap, FILE\_MAP\_WRITE, 0, 0, dwFileSize);

 sort(memory, memory + dwFileSize);

 UnmapViewOfFile(memory);

 CloseHandle(hMap);

 CloseHandle(hFile);

}

void SortBasePoints()

{

 string fname = "Base Points.txt";

 wstring wstr(fname.begin(), fname.end());

 LPCWSTR str = (LPCWSTR)wstr.c\_str();

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

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

 DWORD dwFileSize = GetFileSize(hFile, NULL);

 char \*memory = (char\*)MapViewOfFile(hMap, FILE\_MAP\_WRITE, 0, 0, dwFileSize);

 int dataSize = 1;

 char \*data = new char[dwFileSize / dataSize];

 for (int i = 0; i <= dwFileSize / dataSize; ++i)

 {

 data[i] = \*(memory + i\*dataSize);

 }

 sort(data, data + dwFileSize);

 for (int i = 0; i <= dwFileSize / dataSize; ++i)

 {

 memory[i] = \*(data + i\*dataSize);

 }

 UnmapViewOfFile(memory);

 CloseHandle(hMap);

 CloseHandle(hFile);

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

 SYSTEMTIME st, ft;

 GetSystemTime(&st);

 SortHeaps();

 GetSystemTime(&ft);

 cout << "Sorting with Heaps: " << (ft.wMilliseconds - st.wMilliseconds) << " ms" << endl;

 GetSystemTime(&st);

 SortMappingFiles();

 GetSystemTime(&ft);

 cout << "Sorting with Mapping: " << (ft.wMilliseconds - st.wMilliseconds) << " ms" << endl;

 GetSystemTime(&st);

 SortBasePoints();

 GetSystemTime(&ft);

 cout << "Sorting with Base Points: " << (ft.wMilliseconds - st.wMilliseconds) << " ms" << endl;

 return 0;

}

**Вывод:**

В ходе выполнения лабораторной работы были изучены основные методы управления памятью. Результаты замера времени по каждому методу дали понять, что по скорости работы методы распределения примерно равны.

Наиболее простым в реализации и понимании программистом является метод распределения памяти с помощью куч.

Наиболее сложным, но практичным является метод базовых элементов, который по своей сути является модернизацией метода отображения файлов, но без необходимости каждый раз создавать новое отображение при последующем обращении к этому файлу.