Университет ИТМО

Кафедра ИПМ

Системное программное обеспечение

Лабораторная работа № 4

Управление потоками

Работу выполнил:

Студент 2 курса

Группы № 2125

Назарьев Сергей

Санкт-Петербург

2014 г.

**Цель работы:**

1. Усовершенствуйте программу sortMT таким образом, чтобы в случаях, когда количество потоков, заданное в командной строке, равно 0, программа определяла количество процессоров, установленных в локальной системе, с помощью функции GetSystemInfo. Задавая количество потоков равным различным кратным количества процессоров (используя коэффициенты кратности 1, 2, 4 и так далее), определите, изменяется ли при этом производительность.
2. Видоизмените программу sortMT таким образом, чтобы рабочие потоки не приостанавливались при их создании. Сказываются ли, и если да, то каким именно образом, возникающие при этом нежелательные условия состязаний на работе программы?
3. В программе sortMT, еще до того, как создаются потоки, выполняющие сортировку, осуществляется считывание всего файла основным потоком. Видоизмените программу таким образом, чтобы каждый поток самостоятельно считывал необходимую часть файла.

**Код программы:**

#include "Everything.h"

#define DATALEN 56

#define KEYLEN 8

**typedef** **struct** \_RECORD {

TCHAR key[KEYLEN];

TCHAR data[DATALEN];

} RECORD;

#define RECSIZE sizeof (RECORD)

**typedef** RECORD \* LPRECORD;

**typedef** **struct** \_THREADARG { /\* Thread argument \*/

DWORD iTh; /\* Thread number: 0, 1, 3, ... \*/

LPRECORD lowRecord; /\* Low Record \*/

LPRECORD highRecord; /\* High record \*/

} THREADARG, \*PTHREADARG;

**static** DWORD WINAPI **SortThread** (PTHREADARG pThArg);

**static** **int** **KeyCompare** (LPCTSTR, LPCTSTR);

**static** VOID **MergeArrays** (LPRECORD, DWORD);

**static** DWORD nRec;

**static** HANDLE \* pThreadHandle;

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

{

HANDLE hFile, mHandle;

LPRECORD pRecords = NULL;

DWORD lowRecordNum, nRecTh,numFiles, iTh;

LARGE\_INTEGER fileSize;

BOOL noPrint;

**int** iFF, iNP;

PTHREADARG threadArg;

LPTSTR stringEnd;

numFiles = atoi(argv[**1**]);

**if**(!numFiles) {

SYSTEM\_INFO info;

GetSystemInfo(&info);

numFiles = info.dwNumberOfProcessors;

}

hFile = CreateFile (argv[**2**], GENERIC\_READ | GENERIC\_WRITE,

**0**, NULL, OPEN\_EXISTING, **0**, NULL);

**if** (!SetFilePointer(hFile, **2**, **0**, FILE\_END) || !SetEndOfFile(hFile));

mHandle = CreateFileMapping(hFile, NULL, PAGE\_READWRITE, **0**, **0**, NULL);

**if** (!GetFileSizeEx (hFile, &fileSize));

nRec = (DWORD)fileSize.QuadPart / RECSIZE; /\* Total number of records. Note assumed limit \*/

nRecTh = nRec / numFiles; /\* Records per thread. \*/

threadArg = malloc (numFiles \* **sizeof** (THREADARG)); /\* Array of thread args. \*/

pThreadHandle = malloc (numFiles \* **sizeof** (HANDLE));

pRecords = MapViewOfFile(mHandle, FILE\_MAP\_ALL\_ACCESS, **0**, **0**, **0**);

CloseHandle (mHandle);

lowRecordNum = **0**;

**for** (iTh = **0**; iTh < numFiles; iTh++) {

threadArg[iTh].iTh = iTh;

threadArg[iTh].lowRecord = pRecords + lowRecordNum;

threadArg[iTh].highRecord = pRecords + (lowRecordNum + nRecTh);

lowRecordNum += nRecTh;

pThreadHandle[iTh] = (HANDLE)\_beginthreadex (

NULL, **0**, SortThread, &threadArg[iTh], **0**, NULL); // CREATE\_SUSPENDED

}

**for** (iTh = **0**; iTh < numFiles; iTh++)

ResumeThread (pThreadHandle[iTh]);

WaitForSingleObject (pThreadHandle[**0**], INFINITE);

**for** (iTh = **0**; iTh < numFiles; iTh++)

CloseHandle (pThreadHandle[iTh]);

stringEnd = (LPTSTR) pRecords + nRec\*RECSIZE;

\*stringEnd =\_T('\0');

\_tprintf (\_T("%s"), (LPCTSTR) pRecords);

UnmapViewOfFile(pRecords);

**if** (!SetFilePointer(hFile, -**2**, **0**, FILE\_END) || !SetEndOfFile(hFile));

CloseHandle(hFile);

free(threadArg);

free(pThreadHandle);

**return** **0**;

}

DWORD WINAPI **SortThread** (PTHREADARG pThArg)

{

DWORD groupSize = **2**, myNumber, twoToI = **1**;

DWORD\_PTR numbersInGroup;

LPRECORD first;

myNumber = pThArg->iTh;

first = pThArg->lowRecord;

numbersInGroup = (DWORD)(pThArg->highRecord - first);

qsort (first, numbersInGroup, RECSIZE, KeyCompare);

**while** ((myNumber % groupSize) == **0** && numbersInGroup < nRec) {

WaitForSingleObject (pThreadHandle[myNumber + twoToI], INFINITE);

MergeArrays (first, numbersInGroup);

numbersInGroup \*= **2**;

groupSize \*= **2**;

twoToI \*=**2**;

}

**return** **0**;

}

**static** VOID **MergeArrays** (LPRECORD p1, DWORD nRecs)

{

DWORD iRec = **0**, i1 = **0**, i2 = **0**;

LPRECORD pDest, p1Hold, pDestHold, p2 = p1 + nRecs;

pDest = pDestHold = malloc (**2** \* nRecs \* RECSIZE);

p1Hold = p1;

**while** (i1 < nRecs && i2 < nRecs) {

**if** (KeyCompare ((LPCTSTR)p1, (LPCTSTR)p2) <= **0**) {

memcpy (pDest, p1, RECSIZE);

i1++; p1++; pDest++;

}

**else** {

memcpy (pDest, p2, RECSIZE);

i2++; p2++; pDest++;

}

}

**if** (i1 >= nRecs)

memcpy (pDest, p2, RECSIZE \* (nRecs - i2));

**else** memcpy (pDest, p1, RECSIZE \* (nRecs - i1));

memcpy (p1Hold, pDestHold, **2** \* nRecs \* RECSIZE);

free (pDestHold);

**return**;

}

**int** **KeyCompare** (LPCTSTR pRec1, LPCTSTR pRec2)

{

DWORD i;

TCHAR b1, b2;

LPRECORD p1, p2;

**int** Result = **0**;

p1 = (LPRECORD)pRec1;

p2 = (LPRECORD)pRec2;

**for** (i = **0**; i < KEYLEN && Result == **0**; i++) {

b1 = p1->key[i];

b2 = p2->key[i];

**if** (b1 < b2) Result = -**1**;

**if** (b1 > b2) Result = +**1**;

}

**return** Result;

}

**Вывод:**