ՀՏԴ 681/3

Ինֆորմատիկա

Սվետլանա ԵՐԻՑՅԱՆ
ԱրՊՀ կիրառական մաթեմատիկայի և ինֆորմատիկայի ամբիոնի դոցենտի պաշտոնակատար
e-mail. s.a_ericyan@mail.ru

Անուշ ՀԱՐՈՒԹՅՈՒՆՅԱՆ
ԱրՊՀ կիրառական մաթեմատիկայի և ինֆորմատիկայի ամբիոնի դասախոս
e-mail. asrian.anush@mail.ru

Վազգեն ԱՌՍՏԱՄՅԱՆ
ԱրՊՀ կիրառական մաթեմատիկայի և ինֆորմատիկայի ամբիոնի դոցենտ, մ.գ.թ.
e-mail.varustamyan@rambler.ru

ՖԱՅԼԵՐԻ ՄԻԱՁՈՒԼՄԱՆ ԽՆԴՐԻ ԻՐԱԿԱՆԱՑՄԱՆ ԲԱԶՄԱՀՈՍՔԱՅԻՆ ԾՐԱԳՐԻ ՍՏԵՂԾՈՒՄԸ ՆԵՐԱԾՄԱՆ/ԱՐՏԱԾՄԱՆ ԳՈՐԾՈՂՈՒԹՅՈՒՆՆԵՐԻ ԱՍԻՆԽՐՈՆ ԵՂԱՆԱԿԻ ՀԻՄԱՆ ՎՐԱ

Սույն հոդվածում ներկայացված է հավելված՝ Windows օպերացիոն համակարգերի կիրառական ծրագրավորման ինտերֆեյսի (API) միջոցով, որը միաձուլում է կարգավորված ֆայլեր:
Բանալի բառեր՝ ասինխրոն, բազմահոսքային, overlapped, ռեկուրսիա, ֆայլերի միաձուլում

СОЗДАНИЕ МНОГОПОТОЧНОЙ ПРОГРАММЫ СЛИЯНИЯ ФАЙЛОВ НА ОСНОВЕ АСИНХРОННЫХ ОПЕРАЦИЙ ВВОДА-ВЫВОДА

В статье представлено приложение, использующее интерфейс прикладного программирования Windows (API), цель которого слияние упорядоченных файлов.

Ключевые слова: ***асинхронный ввод-вывод, многопоточный, overlapped, рекурсия, слияние файлов.***

CREATING A MULTI-STREAM FILE MERGER SOLUTION BASED ON ASYNCHRONOUS INPUT-OUTPUT OPERATIONS

This article provides an application that uses the Windows Application Programming Interface (API) that merges adjusted files.

Key words: ***asynchronous Input/Output, multithreading, overlapped, recursion, file merging.***

Մշակված է հավելված Windows օպերացիոն համակարգերի կիրառական ծրագրավորման ինտերֆեյսի (API) միջոցով, որտեղ օգտագործված է ծրագրային հոսքեր, ասինխրոն ներածում/արտածում, ռեկուրսիա:

Օգտագործված է ծրագրի արդյունավետությունը և արագությունը ապահովելու համար այլ միջոցներ այնպիսին, ինչպիսինն է օրինակ overlapped:

Ասինխրոն ծրագրավորման հիմնական գաղափարը կայանում է նրանում, որպեսզի իրականացվի մեթոդների առանձին կանչեր և առանց սպասելու այդ կանչերի ավարտին զուգահեռ իրականացնել մեկ այլ աշխատանք:

Սինխրոն գործարկման ժամանակ հավելվածն ունի միայն մեկ հոսք: Ասինխրոն ծրագրավորման օգնությամբ հնարավորություն ենք ստանում գործարկել բազմաթիվ զուգահեռ հոսքեր և դրանց գործարկման ընթացքում օգտագործողի կողմից սահմանել նոր գործողություններ[2]:

Ալգորիթմը նախատեսված է 2^n քանակով ֆայլերի համար, այլապես բերել այդ դեպքի:

Խնդիրը կայանում էր հետևյալում՝ստեղծել ծրագիր, որի միջոցով անհրաժեշտ է միաձուլել մի քանի տեսակավորված ֆայլերի պարունակությունները մեկ այլ ֆայլի մեջ տեսակավորված վիճակում՝ օգտագործելով ներածման/արտածման ասինխրոն մոտեցումներից որևէ մեկը: Տվյալ ծրագրի իրականացման համար ավելի էֆեկտիվ է օգտագործել բազմահոսքային սիմետրիկ մոդելի մեթոդը:

Ծրագիրը ստեղծված է հետևյալ ալգորիթմով: Ռեկուրսիայի առաջին մակարդակում ունենալով 2^n քանակով ֆայլեր կատարում է զույգ առ զույգ միաձուլում իրականացնող բոլոր հոսքերը: Հաջորդ մակարդակում արդեն ունենք 2 անգամ քիչ քանակով ֆայլեր: Նույն պրոցեսը կրկնվում է այնքան մինչև ստացվի 1 ֆայլ, որը վերջնական պատասխանն է:

Այն մեծ կիրառություն ունի վիճակագրությունում՝ վերջնական որոշում կայացնելու համար, բարդ համակարգերի տեսության մեջ, մասնավորապես զանգվածային սպասարկման ոլորտում:

Ստորև ներկայացված է այդ ծրագրի կոդը:

```
// asynchrMThread.cpp
// asynchrMThread output_file input_file1 input_file2  …

#include <tchar.h>
#include <windows.h>
#include <stdio.h>
#include <process.h>
#define  _STATICLIB
#define _MT
#define  BUF_SIZE  512
typedef struct  _THREADARG{
HANDLE  hIn1;
HANDLE  hIn2;
HANDLE  hOut;
} THREADARG;
static  unsigned int WINAPI  ThMergeFiles(void* pThArg);
int _tmain(int argc, _TCHAR* argv[])
{ struct TempFile{
  TCHAR TempFileName [MAX_PATH];
}*tempName;
HANDLE * hIn;
THREADARG  *Tharg;
int i,iFile,nThs,nTemp;
for( i=1;((argc-2)>>i)>=2;i++);
nThs=i;
nTemp=argc-2-1;
Tharg=(THREADARG*) malloc( nTemp* sizeof(THREADARG));
tempName=(TempFile*)malloc(nTemp*sizeof(TempFile));
hIn=(HANDLE*)malloc((argc-2+ nTemp)*sizeof(HANDLE));
for (iFile =0;iFile<argc-2; iFile++)
hIn[ iFile]=CreateFile(argv[iFile +2],GENERIC_READ,0,NULL,OPEN_EXISTING,
0  ,NULL);
for (i=0;iFile< argc-2+nTemp-1; iFile++,i++)
 {GetTempFileName(_T("."),_T("mrg"),0, tempName[i].TempFileName );
  hIn [iFile]= CreateFile(tempName[i].TempFileName,
```

```
      GENERIC_WRITE|GENERIC_READ, 0,NULL,
CREATE_ALWAYS,FILE_FLAG_DELETE_ON_CLOSE,NULL);
     }
    hIn[iFile]=CreateFile(argv[1],GENERIC_WRITE|GENERIC_READ,
,NULL,CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);
    int n=0,iTh,m,k=0;
    unsigned  int ThId;
    int nThPass=(argc-2)/2;
    m= argc-2;
    HANDLE *ThreadHandle;
    ThreadHandle=(HANDLE*)malloc(( nTemp)*sizeof(HANDLE));
    while(nThPass>=1)
      { for( iTh=0;iTh <nThPass; iTh++)
      {
      Tharg[iTh+k].hIn1= hIn [n +iTh*2];
      Tharg[iTh+k].hIn2= hIn [n +iTh*2+1];
      Tharg[iTh+k].hOut= hIn[m +iTh];
      ThreadHandle[iTh+k]=(HANDLE)_beginthreadex(NULL,0,
ThMergeFiles,(void*)&Tharg[iTh+k],0,&ThId);
      }

    WaitForMultipleObjects(nThPass, ThreadHandle+k,TRUE,INFINITE  );
    n+=m ;k+=nThPass; m+= nThPass; nThPass/=2;
     }
      int  inBuffer1 [BUF_SIZE*5];DWORD nIn1;OVERLAPPED ov={0,0,0,0,NULL};
      printf("test1.dat\n");
      ReadFile (hIn[0],inBuffer1, BUF_SIZE*sizeof(int)*5,&nIn1,&ov);
      for(i=0;i<nIn1/sizeof(int);i++)
      {printf("%d  ",inBuffer1[i]);
      if((i+1)%15==0)
      printf("\n");
      }
      printf("\n");
      printf("test2.dat\n");
      ReadFile (hIn[1], inBuffer1, BUF_SIZE*sizeof(int)*5,&nIn1,&ov);
      for(i=0;i<nIn1/sizeof(int);i++)
      {printf("%d  ",inBuffer1[i]);
```

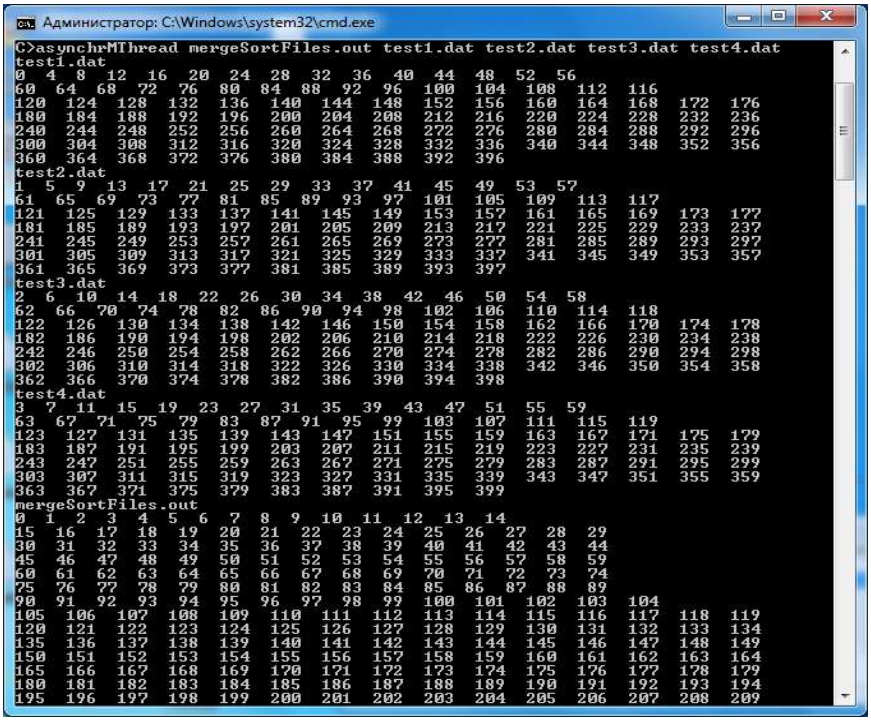
```
  if((i+1)%15==0)
  printf("\n");}
  printf("\n");
  printf("test3.dat\n");
  ReadFile (hIn[2], inBuffer1, BUF_SIZE*sizeof(int)*5,&nIn1,&ov);
  for(i=0;i<nIn1/sizeof(int);i++)
  {printf("%d  ",inBuffer1[i]);
  if((i+1)%15==0)
  printf("\n");
  }
  printf("\n");
  printf("test4.dat\n");
  ReadFile (hIn[3], inBuffer1, BUF_SIZE*sizeof(int)*5,&nIn1,&ov);
  for(i=0;i<nIn1/sizeof(int);i++)
  {printf("%d  ",inBuffer1[i]);
  if((i+1)%15==0)
  printf("\n");
  }
  printf("\n");
  printf("mergeSortFiles.out\n");
  ReadFile (hIn[argc-2+nTemp-1], inBuffer1, BUF_SIZE*sizeof(int)*5,&nIn1,&ov);
  for(i=0;i<nIn1/sizeof(int);i++)
  {printf("%d  ",inBuffer1[i]);
  if((i+1)%15==0)
  printf("\n");}
  printf("\n");
for(iTh=0;iTh < nTemp; iTh++)
  CloseHandle(ThreadHandle[iTh]);
for(iTh=0;iTh <argc-2+nTemp; iTh++)
  CloseHandle(hIn[iTh]);
free(tempName);
free(Tharg);
free(hIn);
free(ThreadHandle);
return 0;
 }
```

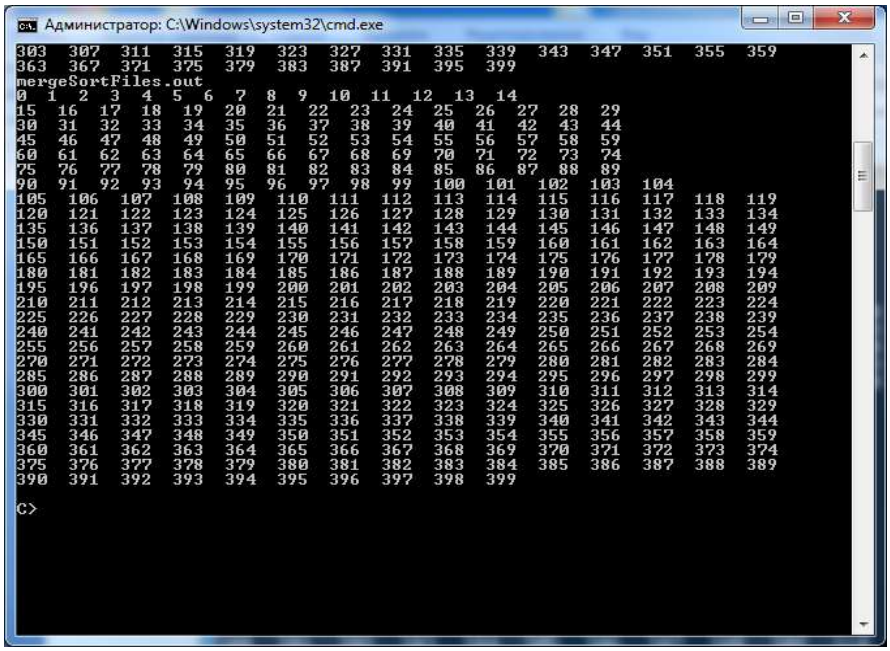
```
    static  unsigned int  WINAPI  ThMergeFiles(void *pvThArg)
     {
      OVERLAPPED ov={0,0,0,0,NULL};
      THREADARG*pThArg=(THREADARG*)pvThArg;
       int  inBuffer1 [BUF_SIZE], inBuffer2[BUF_SIZE],outBuffer [ BUF_SIZE];
      LARGE_INTEGER  FileSize1, FileSize2,FilePos;
      LONGLONG curPosIn1, curPosIn2, curPosOut;
      FilePos.QuadPart=0;
    FileSize1.LowPart=GetFileSize((THREADARG*)pThArg-
>hIn1,(LPDWORD)&FileSize1.HighPart);
      FileSize2.LowPart=GetFileSize( pThArg-
>hIn2,(LPDWORD)&FileSize2.HighPart);
      DWORD nIn1,nIn2,nOut;
      int  i1,i2,i3;
      i1=i2=i3=0;
       curPosIn1=curPosIn2=curPosOut =0;
      BOOL ok1,ok2;
      ok1=ok2=TRUE;
      ReadFile (pThArg->hIn1, inBuffer1, BUF_SIZE*sizeof(int),&nIn1,&ov);
      curPosIn1+= nIn1;
      ReadFile  (pThArg->hIn2, inBuffer2, BUF_SIZE*sizeof(int),&nIn2,&ov);
      curPosIn2+= nIn2;
      while(1){
      if  (  inBuffer1[i1] < inBuffer2[i2])
      {
      outBuffer[i3]= inBuffer1[i1]; ++i3; ++i1;
      if ( i1* sizeof(int)== nIn1)
      {
      if(ReadFile( pThArg->hIn1, inBuffer1,
BUF_SIZE*sizeof(int),&nIn1,NULL)&&nIn1>0)
      {i1=0;curPosIn1+= nIn1;}
      else
      {ok1=FALSE;break;}
      }
      }
      else
      {
```

```
outBuffer[i3]= inBuffer2[i2]; i3++; i2++;
if(i2* sizeof(int)== nIn2)
{
if(ReadFile( pThArg->hIn2, inBuffer2,
BUF_SIZE*sizeof(int),&nIn2,NULL)&&nIn2>0)
{i2=0;curPosIn2+= nIn2;}
else
{ok2=FALSE;break;}
}
}
if (i3== BUF_SIZE)
{
WriteFile(pThArg->hOut,outBuffer, BUF_SIZE*sizeof(int) ,&nOut,&ov);
i3=0;
FilePos.QuadPart+=nOut;
ov.Offset=FilePos.LowPart;
ov.OffsetHigh=FilePos.HighPart;
}
}
if (ok1==FALSE && ok2==TRUE)
{
WriteFile(pThArg->hOut,outBuffer, i3*sizeof(int) ,&nOut,NULL);
curPosOut+=nOut;
WriteFile(pThArg->hOut,inBuffer2+i2, nIn2-i2*sizeof(int) ,&nOut,NULL);
curPosOut+=nOut;
while(ReadFile( pThArg->hIn2, inBuffer2,
BUF_SIZE*sizeof(int),&nIn2,NULL)&&nIn2>0)
{
curPosIn2+= nIn2;
WriteFile (pThArg->hOut,inBuffer2,nIn2,&nOut,NULL);
curPosOut+=nOut;
}
}
if (ok1== TRUE && ok2== FALSE)
{
WriteFile(pThArg->hOut,outBuffer, i3*sizeof(int) ,&nOut,NULL);
curPosOut+=nOut;
```

```
    WriteFile(pThArg->hOut,inBuffer1+i1, nIn1-i1*sizeof(int) ,&nOut,NULL);
    curPosOut+=nOut;
  while(ReadFile( pThArg->hIn1, inBuffer1,
BUF_SIZE*sizeof(int),&nIn1,NULL)&&nIn1>0)
    {
    curPosIn1+= nIn1;
    WriteFile (pThArg->hOut,inBuffer1, nIn1 ,&nOut,NULL);
    curPosOut+=nOut;
    }
    }
    _endthreadex(0);
    return 0;
    }
```

Գործարկելով այս ծրագիրը՝ որպես արդյունք կունենանք հետևյալը՝





Windows-ում ներածման/արտածման գործողությունների ասինխրոն իրականացումն ապահովվում է հետևյալ երեք մեթոդներով [1]՝

- բազմահոսքային ներածում/ արտածում (**Multihreaded I/O**): Պրոցեսի կամ պրոցեսների խմբի մեջ յուրաքանչյուր հոսք իրականացնում է սովորական սինխրոն ներածում/արտածում, իսկ այդ ժամանակ մնացած հոսքերը կարող են շարունակել իրենց գործարկումները:

- փոխաածածկվող ներածում/արտածում (**Overlapped I/O**): Իրականացնելով ընթերցման, գրառման կամ ինչ-որ այլ ներածման/արտածման գործողություն՝ հոսքը շարունակում է իր գործարկումը: Եթե գործարկման շարունակման համար հոսքին անհրաժեշտ է ներածման/արտածման արդյունքները, ապա այն սպասում է մինչև համապատասխան դեսկրիպտորը դառնա հասանելի կամ մինչև չհասնի նշված պատահարին (**event**):

- ավարտման պրոցեդուրաներ (կամ ընդլայնված ներածում/արտածում) (**Completion routines** (**extended I/O**)): Երբ հասնում է ներածման/արտածման գործողությունների ավարտը, համակարգը կանչում է հատուկ ավարտման պրոցեդուրա, որն իրականացվում է հոսքի ներսում:

Բոլոր ՕՀ-երի հիմնական պրոբլեմը այն գործողություններն են, որոնք պահանջում են գործարկման երկարատև ժամանակ: Օգտագործողը գործարկում է այդպիսի ծրագիր, և այն արգելափակում է ծրագրի հիմնական

հոսքն այնքան ժամանակ, մինչև չավարտվի այդ ծրագրի աշխատանքը: Արդյունքում մենք ունենում ենք օգտագործողի համար անհարմար ՕՀ և դժգոհ հաճախորդներ:

Ասինխրոնացումը ներկայումս համարվում է բավականին տարածված թեմա: Հիմնականում այն օգտագործվում է ցանցային ծրագրավորման համար, սակայն բացի դրանից լինում են այնպիսի դեպքեր, երբ ծրագրի աշխատունակության և ժամանակի ծախսման քչացման համար նույնպես օգտագործում են նշված մոտեցումը:

Ասինխրոն ներածման/արտածման գործողությունների համար **Windows**-ում նախատեսված են երեք մեթոդներ: Ամենատարածված և ամենահարմար մեթոդ է հանդիսանում այն, որը հիմնված է հոսքերի օգտագործման վրա: Յուրաքանյուր հոսք պատասխանում է որոշակի սահմանված հաջորդականությամբ գործողությունների իրականացմանը: Այդ գործողությունները կարող են բաղկացած լինել մեկ կամ մի քանի հաջորդաբար իրականացվող, արգելափակվող ներածման/արտածման գործողությունից: Բացի դրանից, յուրաքանչյուր հոսք պետք է տնօրինի սեփական ֆայլային դեսկրիպտորով կամ կապուղով [6]:

Փոխածածկվող ներածում/արտածումը հնարավորություն է տալիս, օգտագործելով մեկ ֆայլային դեսկրիպտոր, մեկ հոսքով իրականացնել ասինխրոն գործողություններ, բայց առանձին գործողության համար «հոսք–ֆայլային դեսկրիպտոր» զույգի փոխարեն պետք է ներկայացվի պատահարի դեսկրիպտորը: Այդ դեպքում պահանջվում է կազմակերպել գործարկման ավարտման սպասում, յուրաքանչյուր ներածման/արտածման գործողության համար առանձին և հետո մաքրել համակարգային ռեսուրսները կամ էլ կատարել հաջորդական գործարկման ղեկավարման համար այլ գործողություններ:

Մյուս կողմից, ընդլայնված ներածում/արտածումը ավտոմատ կանչում է ավարտման կոդը և չի պահանջում ավելորդ պատահարների օգտագործում:

Փոխածածկվող ներածման/արտածման մեկ անվիճելի առավելություններից է հանդիսանում այն, որ այն հնարավորություն է տալիս ստեղծել ավարտման պորտեր, սակայն այդ առավելության գինը քչանում է այն պատճառով, որ ակտիվ աշխատող հոսքերի խմբի մեջ կարող են օգտագործված լինել սեմաֆորներ: Ավարտման պորտերի լրացուցիչ թերություն է հանդիսանում այն, որ դրանք չեն թույլատրում ջնջել նրանց միացված դեսկրիպտորները:

Գրականություն

1. **Харт Джонсон**, «Системное программирование в среде Windows», Москва, Санкт-Петербург, Киев, 2005г.
2. **Вирт Н.**, Алгоритмы и структуры данных, М.: Мир, 1989г.
3. **Beveridge, Ji and Wiener, Robert,** Moltithreading Applications in Win32, 1997, Addison-wesley developers press, 376 pg.
4. **Brain, Marshall and Reeves, Ron**, Win32 System Services, 2000, Prentice Hall, 720 pg
5. **Cohen, Aaron, Woodring, Mike and Petrusha, Ronald,** Win32 Multithreaded Programming, 1998, December 11, 1997 by O'Reilly Media, 724 pages, Paperback
6. **Douglas C. Schmidt**, C++ Network Programming: Mastering Complexity with ACE and Patterns, 2001, Addison-Wesley Professional, 336 pg.

Հոդվածը տպագրության է երաշխավորել խմբագրական խորհրդի անդամ, ֆ.մ.գ.թ. Գ.Հ.Սահակյանը։