```
/*
 *  StQ 20.05.2012
 *  StQ 25.05.2017
 *  Multi-threading file I/O
 *
 *  One Program for all Synchronization Strategies:
 *  Set flags appropriately
 *
 *  Remind that the input (account) file is changed
 *  (save it to duplicate experiments)
 */

#define UNICODE
#define _UNICODE
#define _CRT_SECURE_NO_WARNINGS

#include <windows.h>
#include <tchar.h>
#include <process.h>
#include <stdio.h>

#define L 30+1

// Debug: Waiting Time (dt milliseconds)
#define dt 1000

/*
 * Select thread calls as:
 * _beginthreadex IFF 1
 * createThread IFF 0
 */
#define THREAD_CALL 0

/*
 * Select Synchronization Mechanism:
 * To select the appropriate sections of codes SET TO 1
 * ONE AND ONLY ONE of the four following flags
 */
#define FL 1
#define CS 0
#define MT 0
#define SE 0

typedef struct files {
  int id;
  long int rn;
  TCHAR n[L];
  TCHAR s[L];
  int balance;
} files_t;

typedef struct threads {
  // Used for File Locking (a different handle for each thread)
  LPTSTR nameAccount;
  // Used for Critical Sections, Mutexes, Semaphores
  // Same handle for all threads
  HANDLE hAccount;
  // Local (operation) file name for each thread
  LPTSTR nameOperation;
} threads_t;

#if THREAD_CALL
unsigned WINAPI threadFunction(LPVOID);
#else
```

```
DWORD WINAPI threadFunction(LPVOID);
#endif

#if FL
void threadFunctionFL (LPVOID);
#endif

#if CS
void threadFunctionCS (LPVOID);
CRITICAL_SECTION cs;
#endif

#if MT
void threadFunctionMT (LPVOID);
HANDLE mt;
#endif

#if SE
void threadFunctionSE (LPVOID);
HANDLE se;
#endif

int _tmain (int argc, LPTSTR argv [])
{
  OVERLAPPED ov = {0, 0, 0, 0, NULL};
  LARGE_INTEGER filePos;
  HANDLE hAccount, *hThread;
  INT i;
  DWORD n;
#if THREAD_CALL
  unsigned *threadId;
#else
  DWORD *threadId;
#endif
  files_t fileData;
  threads_t *threadData;

  hAccount = CreateFile (argv[1], GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL, NULL);
  if (hAccount == INVALID_HANDLE_VALUE) {
    _tprintf (_T("Open file error: %x\n"), GetLastError ());
    return 2;
  }

  // Display Initial File Content (initial balance)
  _tprintf (_T("Initial Balance:\n"));
  while (ReadFile (hAccount, &fileData, sizeof (files_t), &n, NULL)
    && n > 0) {
    _tprintf (_T(" %d %ld %s %s %d\n"),
      fileData.id, fileData.rn, fileData.n, fileData.s,
      fileData.balance);
  }

  threadData = (threads_t *) malloc ((argc-2) * sizeof (threads_t));
  hThread = (HANDLE *) malloc ((argc-2) * sizeof (HANDLE));
#if THREAD_CALL
  threadId = (unsigned *)malloc((argc - 2) * sizeof(unsigned));
#else
  threadId = (DWORD *)malloc((argc - 2) * sizeof(DWORD));
#endif

#if CS
  InitializeCriticalSection (&cs);
```

```
#endif

#if MT
  mt = CreateMutex (NULL, FALSE, NULL);
#endif

#if SE
  se = CreateSemaphore (NULL, 1, 1, NULL);
#endif

  for (i=0; i<argc-2; i++) {
    threadData[i].nameAccount = argv[1];
    threadData[i].hAccount = hAccount;
    threadData[i].nameOperation = argv[i+2];

#if THREAD_CALL
    hThread[i] = (HANDLE) _beginthreadex (NULL, 0, threadFunction,
      &threadData[i], 0, &threadId[i]);
#else
    hThread[i] = CreateThread (NULL, 0,
      (LPTHREAD_START_ROUTINE) threadFunction, &threadData[i],
      0, &threadId[i]);
#endif

    if (hThread[i] == NULL) {
      ExitProcess(0);
    }
  }

  // Wait until all threads have terminated.
  WaitForMultipleObjects (argc-2, hThread, TRUE, INFINITE);
  for (i=0; i<argc-2; i++) {
    CloseHandle(hThread[i]);
  }

#if CS
  DeleteCriticalSection (&cs);
#endif

  // Display Final File Content (final balance)
  filePos.QuadPart = 0;
  SetFilePointerEx (hAccount, filePos, NULL, FILE_BEGIN);

  _tprintf (_T("Final Balance:\n"));
  while (ReadFile (hAccount, &fileData, sizeof (files_t), &n, NULL)
    && n > 0) {
    _tprintf (_T(" %d %ld %s %s %d\n"),
      fileData.id, fileData.rn, fileData.n, fileData.s,
      fileData.balance);
  }

  CloseHandle (hAccount);

  _tprintf (_T("End Now: "));
  _tscanf (_T("%d"), &n);

  return 0;
}

#if THREAD_CALL
unsigned WINAPI threadFunction(LPVOID lpParam) {
#else
DWORD WINAPI threadFunction(LPVOID lpParam) {
#endif
```

```
  static int tmp = 0;
#if FL
  _tprintf (_T("Using File Locking %d\n"), tmp++);
  threadFunctionFL (lpParam);
#endif
#if CS
  _tprintf (_T("Using Critical Section %d\n"), tmp++);
  threadFunctionCS (lpParam);
#endif
#if MT
  _tprintf (_T("Using Mutexes %d\n"), tmp++);
  threadFunctionMT (lpParam);
#endif
#if SE
  _tprintf (_T("Using Semaphores %d\n"), tmp++);
  threadFunctionSE (lpParam);
#endif

#if THREAD_CALL
  _endthreadex (0);
  return (0);
#else
  ExitThread (0);
#endif
}

#if FL
void threadFunctionFL (LPVOID lpParam) {
  threads_t *data;
  files_t fileDataAccount, fileDataOperation;
  LARGE_INTEGER filePos, fileReserved;
  OVERLAPPED ov = {0, 0, 0, 0, NULL};
  DWORD n;
  HANDLE hAccount, hOperation;

  fileReserved.QuadPart = 1 *sizeof (files_t);

  data = (threads_t *) lpParam;

  hAccount = CreateFile (data->nameAccount, GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL, NULL);
  if (hAccount == INVALID_HANDLE_VALUE) {
    _tprintf (_T("Open file error: %x\n"), GetLastError ());
    return;
  }

  hOperation = CreateFile (data->nameOperation, GENERIC_READ,
    FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
    NULL);
  if (hOperation == INVALID_HANDLE_VALUE) {
    _tprintf (_T("Open file error: %x\n"), GetLastError ());
    return;
  }

  while (ReadFile (hOperation, &fileDataOperation, sizeof (files_t),
         &n, NULL) && n > 0) {
    _tprintf (_T("(Thread %s)  %d %ld %s %s %d\n"), data->nameOperation,
      fileDataOperation.id, fileDataOperation.rn, fileDataOperation.n,
      fileDataOperation.s, fileDataOperation.balance);

    filePos.QuadPart = (fileDataOperation.id-1) * sizeof (files_t);
    ov.Offset = filePos.LowPart;
    ov.OffsetHigh = filePos.HighPart;
```

```
    ov.hEvent = 0;

    LockFileEx (hAccount, LOCKFILE_EXCLUSIVE_LOCK, 0,
      fileReserved.LowPart, fileReserved.HighPart, &ov);
    ReadFile (hAccount, &fileDataAccount, sizeof (files_t), &n, &ov);
    fileDataAccount.balance = fileDataAccount.balance +
      fileDataOperation.balance;
    _tprintf (_T("(Thread %s) Sleeping for Record %d ...\n"),
      data->nameOperation, fileDataOperation.id);
    Sleep (dt);
    _tprintf (_T("... (Thread %s) End Sleeping for Record %d\n"),
      data->nameOperation, fileDataOperation.id);
    WriteFile (hAccount, &fileDataAccount, sizeof (files_t), &n, &ov);
    UnlockFileEx (hAccount, 0, fileReserved.LowPart,
      fileReserved.HighPart, &ov);
  }

  CloseHandle (hAccount);
  CloseHandle (hOperation);
  return;
}
#endif

#if CS
void threadFunctionCS (LPVOID lpParam) {
  threads_t *data;
  files_t fileDataAccount, fileDataOperation;
  LARGE_INTEGER filePos;
  OVERLAPPED ov = {0, 0, 0, 0, NULL};
  DWORD n;
  HANDLE hOperation;

  data = (threads_t *) lpParam;

  hOperation = CreateFile (data->nameOperation, GENERIC_READ,
    FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  if (hOperation == INVALID_HANDLE_VALUE) {
    _tprintf (_T("Open file error: %x\n"), GetLastError ());
    return;
  }

  while (ReadFile (hOperation, &fileDataOperation, sizeof (files_t),
     &n, NULL) && n > 0) {
    _tprintf (_T("(Thread %s)  %d %ld %s %s %d\n"), data->nameOperation,
      fileDataOperation.id, fileDataOperation.rn, fileDataOperation.n,
      fileDataOperation.s, fileDataOperation.balance);

    filePos.QuadPart = (fileDataOperation.id-1) * sizeof (files_t);
    ov.Offset = filePos.LowPart;
    ov.OffsetHigh = filePos.HighPart;

    EnterCriticalSection (&cs);
    ReadFile (data->hAccount, &fileDataAccount, sizeof (files_t),
      &n, &ov);
    fileDataAccount.balance = fileDataAccount.balance +
      fileDataOperation.balance;
    _tprintf (_T("(Thread %s) Sleeping ...\n"), data->nameOperation);
    Sleep (dt);
    _tprintf (_T("... (Thread %s) End Sleeping\n"), data->nameOperation);
    WriteFile (data->hAccount, &fileDataAccount, sizeof (files_t), &n, &ov);
    LeaveCriticalSection (&cs);
  }

  CloseHandle (hOperation);
```

```
  return;
}
#endif

#if MT
void threadFunctionMT (LPVOID lpParam) {
  threads_t *data;
  files_t fileDataAccount, fileDataOperation;
  LARGE_INTEGER filePos;
  OVERLAPPED ov = {0, 0, 0, 0, NULL};
  DWORD n;
  HANDLE hOperation;

  data = (threads_t *) lpParam;

  hOperation = CreateFile (data->nameOperation, GENERIC_READ,
    FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
    NULL);
  if (hOperation == INVALID_HANDLE_VALUE) {
    _tprintf (_T("Open file error: %x\n"), GetLastError ());
    return;
  }

  while (ReadFile (hOperation, &fileDataOperation, sizeof (files_t),
         &n, NULL) && n > 0) {
    _tprintf (_T("(Thread %s)  %d %ld %s %s %d\n"), data->nameOperation,
      fileDataOperation.id, fileDataOperation.rn, fileDataOperation.n,
      fileDataOperation.s, fileDataOperation.balance);

    filePos.QuadPart = (fileDataOperation.id-1) * sizeof (files_t);
    ov.Offset = filePos.LowPart;
    ov.OffsetHigh = filePos.HighPart;

    WaitForSingleObject (mt, INFINITE);
    ReadFile (data->hAccount, &fileDataAccount, sizeof (files_t), &n, &ov);
    fileDataAccount.balance = fileDataAccount.balance +
      fileDataOperation.balance;
    _tprintf (_T("(Thread %s) Sleeping ...\n"), data->nameOperation);
    Sleep (dt);
    _tprintf (_T("... (Thread %s) End Sleeping\n"), data->nameOperation);
    WriteFile (data->hAccount, &fileDataAccount, sizeof (files_t), &n, &ov);
    ReleaseMutex (mt);
  }

  CloseHandle (hOperation);
  return;
}
#endif

#if SE
void threadFunctionSE (LPVOID lpParam) {
  threads_t *data;
  files_t fileDataAccount, fileDataOperation;
  LARGE_INTEGER filePos;
  OVERLAPPED ov = {0, 0, 0, 0, NULL};
  DWORD n;
  HANDLE hOperation;

  data = (threads_t *) lpParam;

  hOperation = CreateFile (data->nameOperation, GENERIC_READ,
    FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  if (hOperation == INVALID_HANDLE_VALUE) {
```

```
    _tprintf (_T("Open file error: %x\n"), GetLastError ());
    return;
  }

  while (ReadFile (hOperation, &fileDataOperation, sizeof (files_t),
         &n, NULL) && n > 0) {
    _tprintf (_T("(Thread %s)  %d %ld %s %s %d\n"), data->nameOperation,
      fileDataOperation.id, fileDataOperation.rn, fileDataOperation.n,
      fileDataOperation.s, fileDataOperation.balance);

    filePos.QuadPart = (fileDataOperation.id-1) * sizeof (files_t);
    ov.Offset = filePos.LowPart;
    ov.OffsetHigh = filePos.HighPart;

    WaitForSingleObject (se, INFINITE);
    ReadFile (data->hAccount, &fileDataAccount, sizeof (files_t), &n, &ov);
    fileDataAccount.balance = fileDataAccount.balance +
      fileDataOperation.balance;
    _tprintf (_T("(Thread %s) Sleeping ...\n"), data->nameOperation);
    Sleep (dt);
    _tprintf (_T("... (Thread %s) End Sleeping\n"), data->nameOperation);
    WriteFile (data->hAccount, &fileDataAccount, sizeof (files_t), &n, &ov);
    ReleaseSemaphore (se, 1, NULL);
  }

  CloseHandle (hOperation);
  return;
}
#endif
```