ELEC-H-473 — Exercise 03
Multi-threaded execution (on a multi-core architecture)

1. Description of the lab

Objectives

1. Learn how to create multiple threads
2. Quantify potential acceleration
3. Understand the limits of acceleration due to memory bottlenecks

2. Exercise

- Read the source code provided bellow and understand different concepts implemented (it is not necessary to implement this one)
- Understand the thread manipulation functions (thread creation, mutex)
- Start from the exercises that you have already did in 2 previous session (use both threshold and max function)
- Initiate 2, 4 and 8 threads that work on different image subsets (synchronization is not necessary)
- Think of different image subset shapes, what would be the ideal one?

Example

```
// sample_multithread_c_program.c
// compile with: /c
//
//  Bounce - Creates a new thread each time the letter 'a' is typed.
//  Each thread bounces a happy face of a different color around
//  the screen. All threads are terminated when the letter 'Q' is
//  entered.
//

#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>

#define MAX_THREADS  32

// The function getrandom returns a random number between
// min and max, which must be in integer range.
#define getrandom( min, max ) (SHORT)((rand() % (int)(((max) + 1) - \
                              (min))) + (min))

int main( void );                   // Thread 1: main
void KbdFunc( void  );              // Keyboard input, thread dispatch
void BounceProc( void * MyID );     // Threads 2 to n: display
void ClearScreen( void );           // Screen clear
void ShutDown( void );              // Program shutdown
void WriteTitle( int ThreadNum );   // Display title bar information

HANDLE  hConsoleOut;                // Handle to the console
HANDLE  hRunMutex;                  // "Keep Running" mutex
HANDLE  hScreenMutex;               // "Screen update" mutex
int     ThreadNr;                   // Number of threads started
CONSOLE_SCREEN_BUFFER_INFO csbiInfo; // Console information


int main() // Thread One
{
    // Get display screen information & clear the screen.
    hConsoleOut = GetStdHandle( STD_OUTPUT_HANDLE );
    GetConsoleScreenBufferInfo( hConsoleOut, &csbiInfo );
    ClearScreen();
```

```
    WriteTitle( 0 );

    // Create the mutexes and reset thread count.
    hScreenMutex = CreateMutex( NULL, FALSE, NULL );  // Cleared
    hRunMutex = CreateMutex( NULL, TRUE, NULL );      // Set
    ThreadNr = 0;

    // Start waiting for keyboard input to dispatch threads or exit.
    KbdFunc();

    // All threads done. Clean up handles.
    CloseHandle( hScreenMutex );
    CloseHandle( hRunMutex );
    CloseHandle( hConsoleOut );
}

void ShutDown( void ) // Shut down threads
{
    while ( ThreadNr > 0 )
    {
        // Tell thread to die and record its death.
        ReleaseMutex( hRunMutex );
        ThreadNr--;
    }

    // Clean up display when done
    WaitForSingleObject( hScreenMutex, INFINITE );
    ClearScreen();
}

void KbdFunc( void ) // Dispatch and count threads.
{
    int         KeyInfo;

    do
    {
        KeyInfo = _getch();
        if ( tolower( KeyInfo ) == 'a' &&
             ThreadNr < MAX_THREADS )
        {
            ThreadNr++;
            _beginthread( BounceProc, 0, &ThreadNr );
            WriteTitle( ThreadNr );
        }
    } while( tolower( KeyInfo ) != 'q' );

    ShutDown();
}

void BounceProc( void *pMyID )
{
    char    MyCell, OldCell;
    WORD    MyAttrib, OldAttrib;
    char    BlankCell = 0x20;
    COORD   Coords, Delta;
    COORD   Old = {0,0};
    DWORD   Dummy;
    char    *MyID = (char*)pMyID;

    // Generate update increments and initial
    // display coordinates.
    srand( (unsigned int) *MyID * 3 );

    Coords.X = getrandom( 0, csbiInfo.dwSize.X - 1 );
    Coords.Y = getrandom( 0, csbiInfo.dwSize.Y - 1 );
    Delta.X = getrandom( -3, 3 );
    Delta.Y = getrandom( -3, 3 );

    // Set up "happy face" & generate color
    // attribute from thread number.
    if( *MyID > 16)
        MyCell = 0x01;          // outline face
    else
        MyCell = 0x02;          // solid face
    MyAttrib =  *MyID & 0x0F;   // force black background

    do
    {
        // Wait for display to be available, then lock it.
```

```
            WaitForSingleObject( hScreenMutex, INFINITE );

            // If we still occupy the old screen position, blank it out.
            ReadConsoleOutputCharacter( hConsoleOut, &OldCell, 1,
                                        Old, &Dummy );
            ReadConsoleOutputAttribute( hConsoleOut, &OldAttrib, 1,
                                        Old, &Dummy );
            if (( OldCell == MyCell ) && (OldAttrib == MyAttrib))
                WriteConsoleOutputCharacter( hConsoleOut, &BlankCell, 1,
                                             Old, &Dummy );

            // Draw new face, then clear screen lock
            WriteConsoleOutputCharacter( hConsoleOut, &MyCell, 1,
                                         Coords, &Dummy );
            WriteConsoleOutputAttribute( hConsoleOut, &MyAttrib, 1,
                                         Coords, &Dummy );
            ReleaseMutex( hScreenMutex );

            // Increment the coordinates for next placement of the block.
            Old.X = Coords.X;
            Old.Y = Coords.Y;
            Coords.X += Delta.X;
            Coords.Y += Delta.Y;

            // If we are about to go off the screen, reverse direction
            if( Coords.X < 0 || Coords.X >= csbiInfo.dwSize.X )
            {
                Delta.X = -Delta.X;
                Beep( 400, 50 );
            }
            if( Coords.Y < 0 || Coords.Y > csbiInfo.dwSize.Y )
            {
                Delta.Y = -Delta.Y;
                Beep( 600, 50 );
            }
        }
        // Repeat while RunMutex is still taken.
        while ( WaitForSingleObject( hRunMutex, 75L ) == WAIT_TIMEOUT );
    }

    void WriteTitle( int ThreadNum )
    {
        enum {
            sizeOfNThreadMsg = 80
        };
        char    NThreadMsg[sizeOfNThreadMsg];

        sprintf_s( NThreadMsg, sizeOfNThreadMsg,
                   "Threads running: %02d.  Press 'A' "
                   "to start a thread,'Q' to quit.", ThreadNum );
        SetConsoleTitle( NThreadMsg );
    }

    void ClearScreen( void )
    {
        DWORD    dummy;
        COORD    Home = { 0, 0 };
        FillConsoleOutputCharacter( hConsoleOut, ' ',
                                    csbiInfo.dwSize.X * csbiInfo.dwSize.Y,
                                    Home, &dummy );
    }
```