1.4 Threads in Win32

Multithreaded programs in Windows use the functions in the Win32 API. Threads are created by calling function *_beginthreadex*():

```
unsigned long _beginthreadex(
    void* security,                 // security attribute
    unsigned stackSize,             // size of the new thread's stack
    unsigned ( __stdcall *funcStart ) (void *),  // starting address of function to run
    void* argList,                  // arguments to be passed to the thread
    unsigned initFlags,             // initial state of the thread: running or suspended
    unsigned* threadAddr            // thread ID
);
```

If *_beginthreadex*() is successful, it returns a valid thread handle, which must be cast to the Win32 type HANDLE to be used in other functions. It returns 0 if it fails.

The program in Listing 1.3 is a C++/Win32 version of the Java program in Listing 1.1.

- Each thread executes the code in function *simpleThread*(),
- Thread IDs are integers that the user supplies as the fourth argument on the call to function *_beginthreadex()*.
- The *main* thread uses function *WaitForMultipleObjects*() to explicitly wait for both threads to complete before it exits the *main()* function. (Java's *main()* method implicitly waits for its child threads to complete.)
- When both threads have completed, function *GetExitCodeThread()* is used to capture the return values of the threads.

The Windows operating system uses preemptive, priority-based scheduling.

```cpp
#include <iostream>
#include <windows.h>
#include <process.h> // needed for function _beginthreadex()

unsigned WINAPI simpleThread (LPVOID myID) {
// myID receives the 4th argument of _beginthreadex().
// Note: "WINAPI" refers to the "__stdcall" calling convention used to call Win32
// API functions, and "LPVOID" is a Win32 data type defined as void*

    std::cout << "Thread " << (unsigned) myID << " is running" << std::endl;
    return (unsigned) myID;
}

int main() {
    const int numThreads = 2;
  HANDLE threadArray[numThreads];     // array of thread handles
  unsigned threadID;        // returned by _beginthreadex(), but not used
  DWORD rc;     // return code; (DWORD is defined in WIN32 as unsigned long)

    // Create two threads and store their handles in array threadArray
    threadArray[0]=(HANDLE)_beginthreadex(NULL,0,simpleThread,(LPVOID)
            1U,0,&threadID);
    threadArray[1]=(HANDLE)_beginthreadex(NULL,0,simpleThread,(LPVOID)
            2U,0,&threadID);

    // wait for threads to finish
    rc = WaitForMultipleObjects(numThreads,threadArray,TRUE,INFINITE);

    DWORD result1, result2;// these variables will receive the return values
    rc = GetExitCodeThread(threadArray[0],&result1);
    rc = GetExitCodeThread(threadArray[1],&result2);
    std::cout << "thread1:" << result1 << " thread2:" << result2 << std::endl;

    rc = CloseHandle(threadArray[0]);     // release reference to thread when finished
    rc = CloseHandle(threadArray[1]);
    return 0;
}
```

Listing 1.3 A simple concurrent program using C++/Win32.