Processes and Threads in Windows

1 Generic Procedure for Creating Win32 Console Applications

1. In Windows, open the Visual C++ IDE (Start → Programs → Microsoft Visual Studio 6.0 → Microsoft Visual C++ 6.0)
2. Open a new Win32 *Console* Application project (File → New → Project → Win32 Console Application). Note that you need to do this *for each of the three applications.*
3. Specify a project name (i.e., "print", "callprint" or "threads")
4. Click on the "Empty Project" radio button, click the various confirmation buttons ($\boxed{\text{Finish}}$, $\boxed{\text{OK}}$, etc., etc.,...)
5. Select Project → Add to Project → Files
6. Enter the name of your source file
7. Compile and link your program with Build → Build ⟨*project name*⟩.exe
8. Execute it with Build → Execute ⟨*project name*⟩.exe
9. Note that for `print.c`, you should create a project name that matches the directory in the program file `win-call-print.c`, so that `win-call-print.c` can find `print.exe`.
10. Note that the "normal" place to put your source code for each project is `C:\Program Files\Microsoft Visual Studio\MyProjects\`⟨*project name*⟩
11. The software will put the executable program in `C:\Program Files\Microsoft Visual Studio\MyProjects\`⟨*project name*⟩`\Debug`
12. Note that the IDE executes the program in the directory *above* the `Debug` directory, i.e., above where it puts the executable. This is important when executing program 2 on the following page.

2 Procedure

1. Download the source code for each of these programs from the subject web site.
2. For each of program 1, 2 and 3 on page 3, compile and run them, using the generic instructions given in section 1.

Program 1 The program `print.c`. This is identical to the program `print.c` that you experimented with in your last lab.

```
#include <stdio.h>
#include <stdlib.h>

int main( int argc, char *argv[] )
{
        // argv[0] is the program name
        int num = atoi( argv[1] );
        int loops = atoi( argv[2] );
        int i;
        for ( i = 0; i < loops; ++i )
                printf( "%d ", num );
}
```

Program 2 The program `win-call-print.c`. This is like the program `call-print.c` that you experimented with in your last lab.

```
#include <windows.h>
#include <stdio.h>

void main() {
        STARTUPINFO si;
        PROCESS_INFORMATION pi;
        memset( &si, 0, sizeof( si ) );
        si.cb = sizeof( si );
        if ( ! CreateProcess( NULL,
                        "..\\print\\Debug\\print.exe 5 100",
                        NULL, NULL, TRUE, 0, NULL, NULL, &si, &pi ) ) {
                fprintf( stderr, "CreateProcess failed with %d\n", GetLastError() );
                exit( 1 );
        }
        WaitForSingleObject( pi.hProcess, INFINITE );
        CloseHandle( pi.hProcess );
        CloseHandle( pi.hThread );
}
```

3. Modify program 2 so that it creates two processes, both running the program `print.c`.
4. See if you can modify a copy of program 2 so that you can start `notepad.exe`.
5. Modify `win-hello.c`, increasing the number of threads until you find the maximum number of threads that your system can support.
6. The first student who could give me the way to compile any of these programs on the command line in Windows 2000, including all the command line parameters to the compiler, was awarded a prize of $50 HK. The name of the winner is Wendell Lam: his work is shown on the subject web page.

Program 3 A simple program `win-hello.c` that works like the program `hello.c` given in the lecture. We assume that it executes in a directory at the same level as the source code, not in the `Debug` directory.

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define NUM_THREADS 5

DWORD WINAPI print_hello( void *threadid )
{
        printf( "\n%d: Hello World!\n", threadid );
        return 0;
}

int main()
{
        static DWORD threads[ NUM_THREADS ];
        static HANDLE handles[ NUM_THREADS ];
        unsigned int t;
        for ( t = 0; t < NUM_THREADS; ++t ) {
                printf( "Creating thread %d\n", t );
                handles[ t ] = CreateThread( NULL, 0,
                      ( LPTHREAD_START_ROUTINE ) print_hello,
                      ( LPVOID ) t, 0, (LPDWORD) &threads[ t ] );
                if ( ! handles[ t ] ) {
                        printf( "Error: failed to create process %d\n", t );
                        exit( 1 );
                }
        }
        WaitForMultipleObjects( NUM_THREADS, handles, 1, INFINITE );
        return 0;
}
```