Assignment No. 1

PRN: 2020BTECS00025
Name: Sahil Santosh Otari
Course: High Performance Computing Lab
Title of practical: Study and implementation of basic Open MP clauses

Q1. Differentiate between Software and Hardware Threads
Solution:
Hardware Thread:

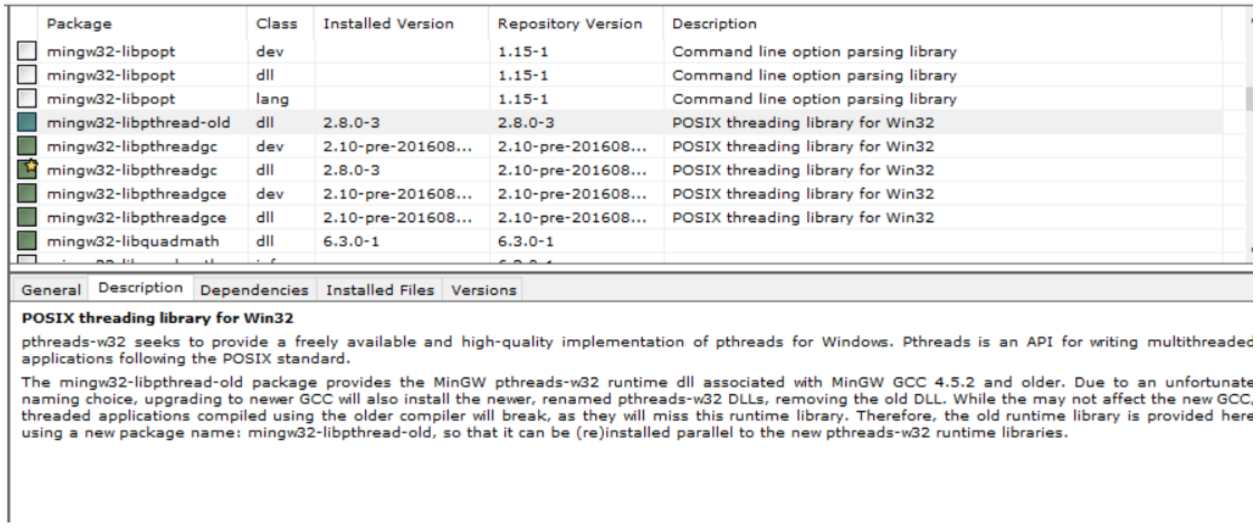
- A "hardware thread" is a physical CPU or core. So, a 4 core CPU can genuinely support 4 hardware threads at once - the CPU really is doing 4 things at the same time.
- One hardware thread can run many software threads.

Software Thread:

- Software threads are threads of execution managed by the operating system.
- Software threads are abstractions to the hardware to make multiprocessing possible.
- If you have multiple software threads, but there are no multiple resources, then these software threads are a way to run all tasks in parallel by allocating resources for limited time (or using some other strategy) so that it appears that all threads are running in parallel. These are managed by the operating system.

Q2. Which type of threads are supported by the processor?
Solution:
Generally, the Hardware Threads are supported by the processor. The
hardware threads are mostly based on the multicore architecture, which is latest
architecture to achieve high performance.
A multithreaded application running on a traditional single-core chip
would have to interleave the threads. On a multi-corechip, however, the threads could be spread across the available cores, allowing
true parallel processing.

Installation:



	Package	Class	Installed Version	Repository Version	Description
☐	mingw32-libpopt	dev		1.15-1	Command line option parsing library
☐	mingw32-libpopt	dll		1.15-1	Command line option parsing library
☐	mingw32-libpopt	lang		1.15-1	Command line option parsing library
■	mingw32-libpthread-old	dll	2.8.0-3	2.8.0-3	POSIX threading library for Win32
■	mingw32-libpthreadgc	dev	2.10-pre-201608...	2.10-pre-201608...	POSIX threading library for Win32
■	mingw32-libpthreadgc	dll	2.8.0-3	2.10-pre-201608...	POSIX threading library for Win32
■	mingw32-libpthreadgce	dev	2.10-pre-201608...	2.10-pre-201608...	POSIX threading library for Win32
■	mingw32-libpthreadgce	dll	2.10-pre-201608...	2.10-pre-201608...	POSIX threading library for Win32
■	mingw32-libquadmath	dll	6.3.0-1	6.3.0-1	

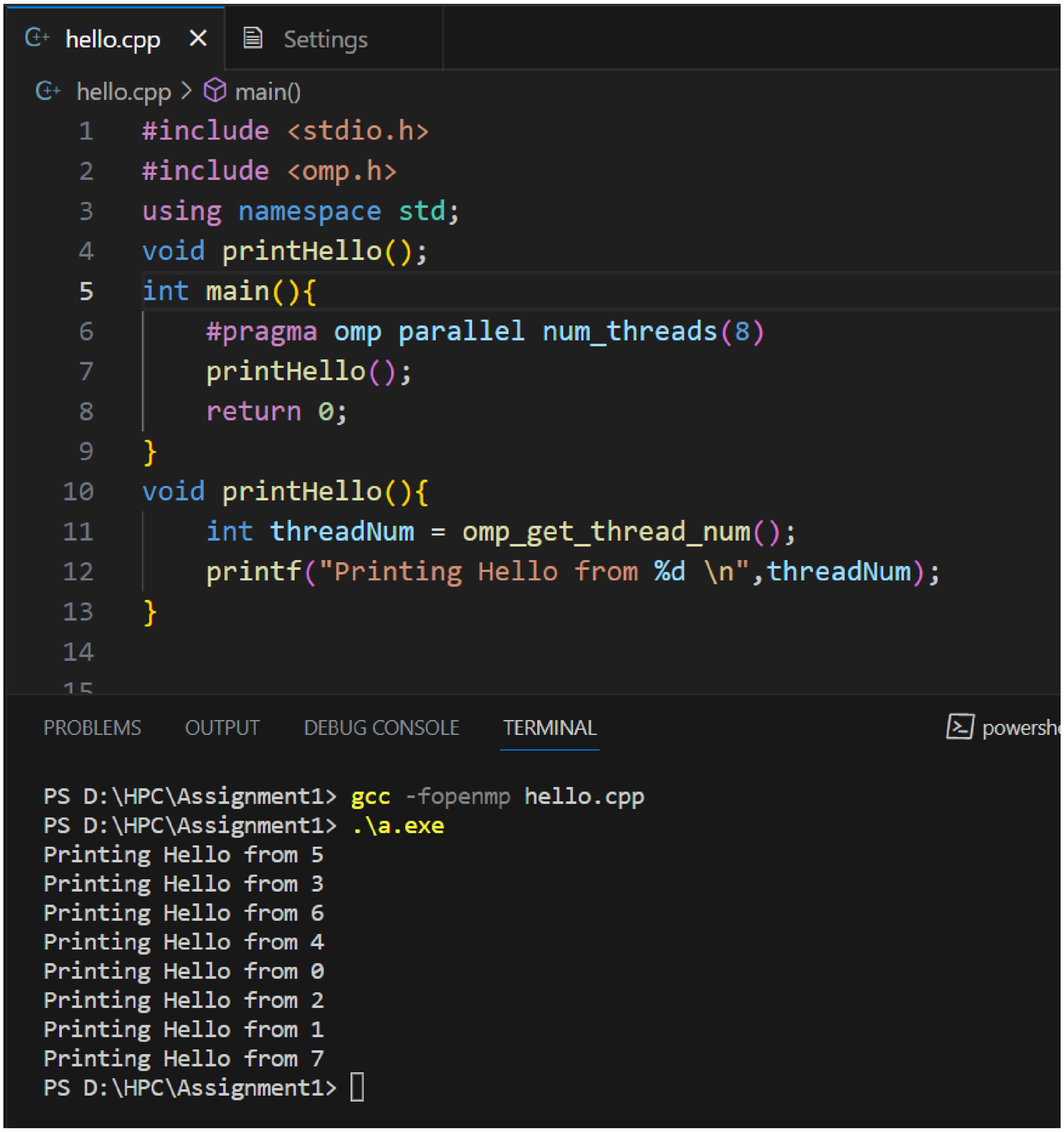
POSIX threading library for Win32

pthreads-w32 seeks to provide a freely available and high-quality implementation of pthreads for Windows. Pthreads is an API for writing multithreaded applications following the POSIX standard.

The mingw32-libpthread-old package provides the MinGW pthreads-w32 runtime dll associated with MinGW GCC 4.5.2 and older. Due to an unfortunate naming choice, upgrading to newer GCC will also install the newer, renamed pthreads-w32 DLLs, removing the old DLL. While the may not affect the new GCC, threaded applications compiled using the older compiler will break, as they will miss this runtime library. Therefore, the old runtime library is provided here using a new package name: mingw32-libpthread-old, so that it can be (re)installed parallel to the new pthreads-w32 runtime libraries.

Problem Statement: Program to print Hello World!

Source Code:



```cpp
#include <stdio.h>
#include <omp.h>
using namespace std;
void printHello();
int main(){
    #pragma omp parallel num_threads(8)
    printHello();
    return 0;
}
void printHello(){
    int threadNum = omp_get_thread_num();
    printf("Printing Hello from %d \n",threadNum);
}
```

```
PS D:\HPC\Assignment1> gcc -fopenmp hello.cpp
PS D:\HPC\Assignment1> .\a.exe
Printing Hello from 5
Printing Hello from 3
Printing Hello from 6
Printing Hello from 4
Printing Hello from 0
Printing Hello from 2
Printing Hello from 1
Printing Hello from 7
PS D:\HPC\Assignment1>
```

Information:
This is a simple example of using OpenMP (Open Multi-Processing) to create a parallel section of code that will be executed by multiple threads. OpenMP is a popular API for parallel programming in C and C++ that allows you to take advantage of multiple CPU cores to perform parallel computations.

Here's some information about the code:

1. “#include<omp.h>”: This line includes the Open MP header file, which provides the necessary functions and directives for parallel programming.
2. “#pragma omp parallel num_threads(8)”: This line begins a parallel section of code using Open MP. It specifies that we want to create a parallel region with 8 threads. Each thread will execute the code within the parallel region concurrently.
3. “Void printHello()”: This function is defined separately from “main” and is responsible for printing a message indicating which thread is executing it.
4. “threadNum = omp_get_thread_num()”: “omp_get_thread_num()” is an Open MP function that retrieves the unique ID of the calling thread. It assigns this ID to the “threadNum” variable.
5. “printf("\n Hello World! from %d \n", threadNum)”: This line uses “printf” to print a message indicating the thread number (“threadNum”). Each thread will print its own thread number, allowing you to see which thread is executing which part of the code.
   When we run this program, we'll observe that the "Hello World!" message is printed multiple times, with each message indicating the thread number. The number of times the message is printed depends on the number of threads specified in the “num_threads()” clause of the “#pragma omp parallel directive (in this case, 10 times). Each thread executes the “printHello” function concurrently, making it a simple parallel program.
6. Command to run : gcc -fopenmp <filename>

Analysis:

The code demonstrates the use of Open MP for parallel programming in C. Here's a short analysis of the code:

- The code utilizes Open MP directives to create a parallel region with 10 threads.
- Within the parallel region, the `printMes()` function is called by each thread.
- The `printMes()` function retrieves and prints the thread's unique identifier.
- As a result, the "Hello World!" message is printed multiple times, each time indicating the thread number.