## 12.a - Implement the program for threads using Openmp library. Print number of cores.

Subject:- Unix Operating System

System Lab Class :- TYIT

Name PRN

Aditi Sudhir Ghate 2020BTEIT00044

**Objectives:**

1 To learn about openMP for better use of multicore systems.

**Theory:**

An OpenMP program has sections that are sequential and sections that are parallel. In general an OpenMP program starts with a sequential section in which it sets up the environment, initializes the variables, and so on.

When run, an OpenMP program will use one thread (in the sequential sections), and several threads (in the parallel sections).

There is one thread that runs from the beginning to the end, and it's called the *master thread*. The parallel sections of the program will cause additional threads to fork. These are called the *slave* threads.

A section of code that is to be executed in parallel is marked by a special directive (omp pragma). When the execution reaches a parallel section (marked by omp pragma), this directive will cause slave threads to form. Each thread executes the parallel section of the code independently. When a thread finishes, it joins the master. When all threads finish, the master continues with code following the parallel section.

Each thread has an ID attached to it that can be obtained using a runtime library function (called omp_get_thread_num()). The ID of the master thread is 0.

**Program:**

```
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>
#define CHUNKSIZE   10
#define N       100

int main (int argc, char *argv[])
{
  int nthreads, tid, i, chunk;
  float a[N], b[N], c[N];

/* Some initializations */
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;
chunk = CHUNKSIZE;

#pragma omp parallel shared(a,b,c,nthreads,chunk) private(i,tid)
```

```
  {
  tid = omp_get_thread_num();
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }
  printf("Thread %d starting...\n",tid);

  #pragma omp for schedule(dynamic,chunk)
  for (i=0; i < N; i++)
       c[i] = a[i] + b[i];
    printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);
    }

  }  /* end of parallel section */

}
```

**Output:**

```
Number of threads = 2
 Thread 1 starting...
 Thread 0 starting...
 Thread 0: c[10]= 20.000000
 Thread 1: c[0]= 0.000000
 Thread 0: c[11]= 22.000000
 Thread 1: c[1]= 2.000000
 Thread 0: c[12]= 24.000000
 Thread 1: c[2]= 4.000000
Thread 0: c[13]= 26.000000
Thread 1: c[3]= 6.000000
Thread 0: c[14]= 28.000000
Thread 1: c[4]= 8.000000
Thread 0: c[15]= 30.000000
Thread 1: c[5]= 10.000000
Thread 0: c[16]= 32.000000
Thread 1: c[6]= 12.000000
Thread 0: c[17]= 34.000000
Thread 1: c[7]= 14.000000
Thread 0: c[18]= 36.000000
Thread 1: c[8]= 16.000000
Thread 0: c[19]= 38.000000
Thread 1: c[9]= 18.000000
Thread 0: c[20]= 40.000000
Thread 1: c[30]= 60.000000
Thread 0: c[21]= 42.000000
Thread 1: c[31]= 62.000000
Thread 0: c[22]= 44.000000
Thread 1: c[32]= 64.000000
Thread 0: c[23]= 46.000000
Thread 1: c[33]= 66.000000
```

**Conclusion:**

- We learned about the concepts of parallel programming using OpenMP.
- Use of OpenMP in Shared memory programming
- Efficient use of the processor and thereby reducing time by using OpenMP.

**References:**

[1]https://www.codeproject.com/Articles/60176/A-Beginner-s-Primer-to-OpenMP
[2]https://www.researchgate.net/post/Is_there_a_way_to_specify_how_many_cores_a_program_should_run-in_other_words_can_I_control_where_the_threads_are_mapped
[3]https://www.embedded.com/design/mcus-processors-and-socs/4007155/Using-OpenMP-for-programming-parallel-threads-in-multicore-applications-Part-2
[4]http://pages.tacc.utexas.edu/~eijkhout/pcse/html/omp-basics.html