**20BCE1025**

**Abhishek N N**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | |  |
|  | | | | | |  |
| Programme | : | **B.Tech.(CSE)** | Semester | : | **Fall ’22-23** | |
| Course | : | **Parallel and Distributed Computing** | Code | : | **CSE4001** | |
| Faculty | : | **R. Kumar** | Slot | : | **L9+L10** | |

1. Write your own code snippet to demonstrate the following
   1. **Barrier**

**Code:**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

omp\_set\_num\_threads(4);

#pragma omp parallel

{

printf("[Thread %d] I print my first message.\n", omp\_get\_thread\_num());

#pragma omp barrier

// One thread indicates that the barrier is complete.

#pragma omp single

{

printf("The barrier is complete, which means all threads have printed their first message.\n");

}

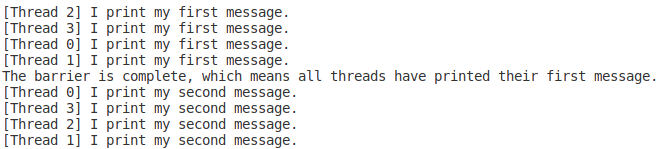
printf("[Thread %d] I print my second message.\n", omp\_get\_thread\_num());

}

return EXIT\_SUCCESS;

}

**Output:**



* 1. **Master**

**Code:**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

omp\_set\_num\_threads(4);

#pragma omp parallel

{

printf("[Thread %d] I print my first message.\n", omp\_get\_thread\_num());

// master only runs (default argument is nowait)

#pragma omp master

{

printf("[Thread %d] Message by master.\n", omp\_get\_thread\_num());

}

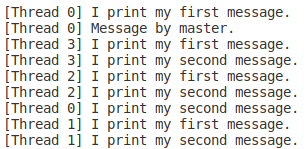
printf("[Thread %d] I print my second message.\n", omp\_get\_thread\_num());

}

return EXIT\_SUCCESS;

}

**Output:**



* 1. **Single**

**Code:**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

omp\_set\_num\_threads(4);

#pragma omp parallel

{

printf("[Thread %d] I print my first message.\n", omp\_get\_thread\_num());

// single: only one thread runs the block(with default barrier after block)

#pragma omp single

{

printf("[Thread %d] messge by single construct.\n", omp\_get\_thread\_num());

}

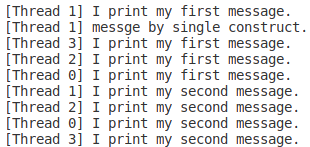
printf("[Thread %d] I print my second message.\n", omp\_get\_thread\_num());

}

return EXIT\_SUCCESS;

}

**Output:**



* 1. **Critical**

**Code:**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

omp\_set\_num\_threads(4);

#pragma omp parallel

{

printf("[Thread %d] I print my first message.\n", omp\_get\_thread\_num());

// critical: only one thread executes the block at a perticular time

#pragma omp critical

{

printf("[Thread %d] messge by critical construct.\n", omp\_get\_thread\_num());

}

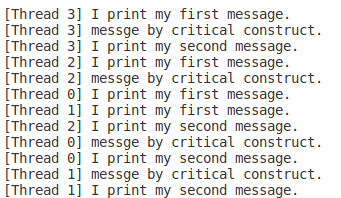
printf("[Thread %d] I print my second message.\n", omp\_get\_thread\_num());

}

return EXIT\_SUCCESS;

}

**Output:**



* 1. **Ordered**

**Code:**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

omp\_set\_num\_threads(4);

#pragma omp parallel for ordered

for(int i=0;i<10;i++){

printf("[%d unordered] [%d thread]\n",i,omp\_get\_thread\_num());

#pragma omp ordered

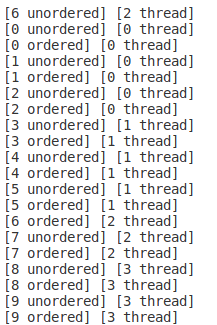
printf("[%d ordered] [%d thread]\n",i,omp\_get\_thread\_num());

}

return EXIT\_SUCCESS;

}

**Output:**



1. Write a parallel program in OpenMP API to implement readers-writers problem. Ensure the usage of suitable low-level synchronization constructs. Document the result of synchronization overhead incurred in your experiment.

**Code:**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <unistd.h>

omp\_lock\_t lock;

void Reader(int tid) {

int a;

// Wait till there is lock

while (1) {

if (!omp\_test\_lock(&lock)) {

printf("\nReader : %d is waiting ", tid);

sleep(1);

}

else break;

}

omp\_unset\_lock(&lock); // No need of lock for reading

printf("\nReader : %d is reading", tid);

sleep(3);

}

void Writer(int tid) {

// First get the lock

while (!omp\_test\_lock(&lock)) {

if (!omp\_test\_lock(&lock)) {

printf("\nWriter : %d is waiting ", tid);

sleep(2);

}

else break;

}

printf("\nWriter : %d acquired lock", tid);

printf("\nWriter : %d is writing", tid);

// perform operation on the shared data

sleep(0.5);

omp\_unset\_lock(&lock); // release the lock after writing

printf("\nWriter : %d released lock", tid);

sleep(2);

}

int main() {

int no, tid;

omp\_init\_lock(&lock);

srand((unsigned)time(NULL));

int totalthreads;

#pragma omp parallel private(tid) shared(totalthreads)

{

totalthreads = omp\_get\_num\_threads();

while (1) {

int randomno = rand() % 100;

// Getting thread ID

tid = omp\_get\_thread\_num();

// half of threads are readers and half are writers

if (randomno < 50) { // calling reader

printf("\nProcess no. %d is a reader ", randomno);

Reader(randomno);

}

else { // calling writer

printf("\nProcess no. %d is a writer ", randomno);

Writer(randomno);

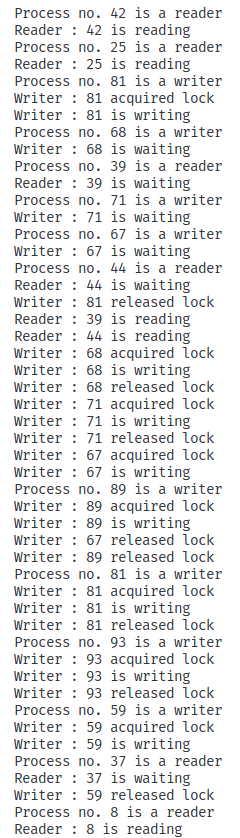
}

}

}

}

**Output:**

****