20BCE1025

Abhishek N N

VIT UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
www.vit.ac.in
Vellore ▪ Chennai
CHENNAI CAMPUS
Vandalur - Kelambakkam Road, Chennai - 600127

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
    a. <u>**Barrier**</u>

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:

```
[Thread 2] I print my first message.
[Thread 3] I print my first message.
[Thread 0] I print my first message.
[Thread 1] I print my first message.
The barrier is complete, which means all threads have printed their first message.
[Thread 0] I print my second message.
[Thread 3] I print my second message.
[Thread 2] I print my second message.
[Thread 1] I print my second message.
```

b. Master

Code:

```c
#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:

```
[Thread 0] I print my first message.
[Thread 0] Message by master.
[Thread 3] I print my first message.
[Thread 3] I print my second message.
[Thread 2] I print my first message.
[Thread 2] I print my second message.
[Thread 0] I print my second message.
[Thread 1] I print my first message.
[Thread 1] I print my second message.
```

c. **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:

```
[Thread 1] I print my first message.
[Thread 1] messge by single construct.
[Thread 3] I print my first message.
[Thread 2] I print my first message.
[Thread 0] I print my first message.
[Thread 1] I print my second message.
[Thread 2] I print my second message.
[Thread 0] I print my second message.
[Thread 3] I print my second message.
```

d. <u>Critical</u>

Code:

```c
#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:

```
[Thread 3] I print my first message.
[Thread 3] messge by critical construct.
[Thread 3] I print my second message.
[Thread 2] I print my first message.
[Thread 2] messge by critical construct.
[Thread 0] I print my first message.
[Thread 1] I print my first message.
[Thread 2] I print my second message.
[Thread 0] messge by critical construct.
[Thread 0] I print my second message.
[Thread 1] messge by critical construct.
[Thread 1] I print my second message.
```

e. **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:

```
[6 unordered] [2 thread]
[0 unordered] [0 thread]
[0 ordered] [0 thread]
[1 unordered] [0 thread]
[1 ordered] [0 thread]
[2 unordered] [0 thread]
[2 ordered] [0 thread]
[3 unordered] [1 thread]
[3 ordered] [1 thread]
[4 unordered] [1 thread]
[4 ordered] [1 thread]
[5 unordered] [1 thread]
[5 ordered] [1 thread]
[6 ordered] [2 thread]
[7 unordered] [2 thread]
[7 ordered] [2 thread]
[8 unordered] [3 thread]
[8 ordered] [3 thread]
[9 unordered] [3 thread]
[9 ordered] [3 thread]
```

2. 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);
```

```c
      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:

```
Process no. 42 is a reader
Reader : 42 is reading
Process no. 25 is a reader
Reader : 25 is reading
Process no. 81 is a writer
Writer : 81 acquired lock
Writer : 81 is writing
Process no. 68 is a writer
Writer : 68 is waiting
Process no. 39 is a reader
Reader : 39 is waiting
Process no. 71 is a writer
Writer : 71 is waiting
Process no. 67 is a writer
Writer : 67 is waiting
Process no. 44 is a reader
Reader : 44 is waiting
Writer : 81 released lock
Reader : 39 is reading
Reader : 44 is reading
Writer : 68 acquired lock
Writer : 68 is writing
Writer : 68 released lock
Writer : 71 acquired lock
Writer : 71 is writing
Writer : 71 released lock
Writer : 67 acquired lock
Writer : 67 is writing
Process no. 89 is a writer
Writer : 89 acquired lock
Writer : 89 is writing
Writer : 67 released lock
Writer : 89 released lock
Process no. 81 is a writer
Writer : 81 acquired lock
Writer : 81 is writing
Writer : 81 released lock
Process no. 93 is a writer
Writer : 93 acquired lock
Writer : 93 is writing
Writer : 93 released lock
Process no. 59 is a writer
Writer : 59 acquired lock
Writer : 59 is writing
Process no. 37 is a reader
Reader : 37 is waiting
Writer : 59 released lock
Process no. 8 is a reader
Reader : 8 is reading
```