**20BCE1550**

**Samridh Anand Paatni**

**CSE4001 Lab 05**

**OMP Synchronization Constructs**

**Q1.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define N 10

#define OMP\_NUM\_THREADS 4

/\*

1. Write your own code snippet to demonstrate the following

    a. Barrier

    b. Master

    c. Single

    d. Critical

    e. Ordered

\*/

void barrier() {

    printf("\na. Barrier:\n");

    int a[1000], b[1000], i = 0, sum = 0;

    for (i = 0; i < 1000; i++) {

        a[i] = rand() % 100;

        b[i] = rand() % 10;

    }

    #pragma omp parallel private(i)

    {

        for (i = 0; i < 1000; i++) {

            a[i] = a[i] - b[i];

        }

        #pragma omp barrier

        #pragma omp for reduction(+ : sum)

        for (i = 0; i < 1000; i++)

        {

            sum += a[i];

        }

    }

    printf("sum: %d\n", sum);

}

void master() {

    printf("\nb. Master\nwithout 'master':\n");

    #pragma omp parallel

    {

        printf("hello, from thread %d\n", omp\_get\_thread\_num());

    }

    printf("\nwith master:\n");

    #pragma omp parallel

    {

        #pragma omp master

        {

            printf("hello, from thread %d\n", omp\_get\_thread\_num());

        }

    }

}

void single() {

    printf("\nc. Single:\n");

    int a=0, b=0;

    #pragma omp parallel num\_threads(4)

    {

        #pragma omp single

        a++;

        #pragma omp critical

        b++;

    }

    printf("single: %d | critical: %d\nsingle runs once, critical is run once per thread\n", a, b);

}

void critical() {

    printf("\nd. Critical:\n");

    int i; int max; int a[N];

    for (i = 0; i < N; i++) {

        a[i] = rand();

        printf(

            "a[%d] = %d\tthread no %d\n",

            i,

            a[i],

            omp\_get\_num\_threads()

        );

    }

    max = a[0];

    #pragma omp parallel for

    for (i = 1; i < N; i++) {

        if (a[i] > max) {

            #pragma omp critical

            {

                if (a[i] > max) max = a[i];

            }

        }

    }

    printf("\nmax = %d\t%d threads\n", max, omp\_get\_num\_threads());

}

void ordered() {

    printf("\ne. Ordered:\nwithout ordered:\n");

    int i = 0;

    int n = 10;

    #pragma omp parallel shared(n) private(i)

    {

        #pragma omp for

        for (i = 0; i < n; i++) {

            printf("thread %d at index %d\n", omp\_get\_thread\_num(), i);

        }

    }

    printf("\nwith ordered:\n");

    #pragma omp parallel shared(n) private(i)

    {

        #pragma omp for ordered

        for (i = 0; i < n; i++) {

            #pragma omp ordered

            {

                printf("thread %d at index %d\n", omp\_get\_thread\_num(), i);

            }

        }

    }

}

int main() {

    barrier();

    master();

    single();

    critical();

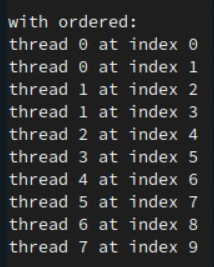
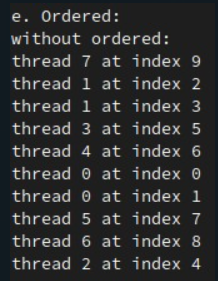
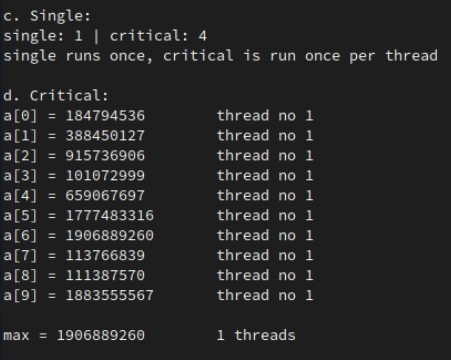
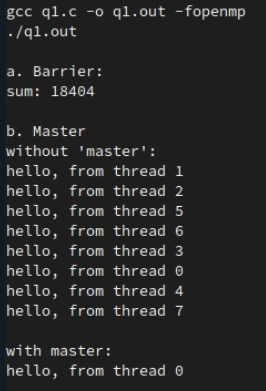
    ordered();

    printf("\n");

    return 0;

}

**Output:**

****

**Q2**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#include <unistd.h>

#include <time.h>

#define MAX\_SLEEP 10

omp\_lock\_t lock;

int cr = 0;

void reader(int i) {

    #pragma omp critical

    {

        cr++;

        if (cr == 1) {

            omp\_set\_lock(&lock);

            printf("lock set by reader %d \n", i);

        }

    }

    printf(

        "reader %d (on thread %d) is reading\n",

        i,

        omp\_get\_thread\_num()

    );

    sleep(rand() % MAX\_SLEEP);

    #pragma omp critical

    {

        cr--;

        if (cr == 0) {

            omp\_unset\_lock(&lock);

            printf("lock unset by reader %d\n", i);

        }

    }

}

void writer(int i) {

    omp\_set\_lock(&lock);

    printf("lock set by writer %d\n", i);

    printf(

        "writer %d (on thread %d) is writing\n",

        i,

        omp\_get\_thread\_num()

    );

    sleep(rand() % MAX\_SLEEP);

    omp\_unset\_lock(&lock);

    printf("lock unset by writer %d\n", i);

}

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

    printf("Readers-writers in parallel\n");

    srand(clock());

    omp\_init\_lock(&lock);

    #pragma omp parallel sections num\_threads(8)

    {

        #pragma omp section

        {

            writer(0);

        }

        #pragma omp section

        {

            reader(0);

        }

        #pragma omp section

        {

            reader(1);

        }

        #pragma omp section

        {

            reader(2);

        }

        #pragma omp section

        {

            writer(1);

        }

        #pragma omp section

        {

            writer(2);

        }

        #pragma omp section

        {

            reader(3);

        }

        #pragma omp section

        {

            reader(4);

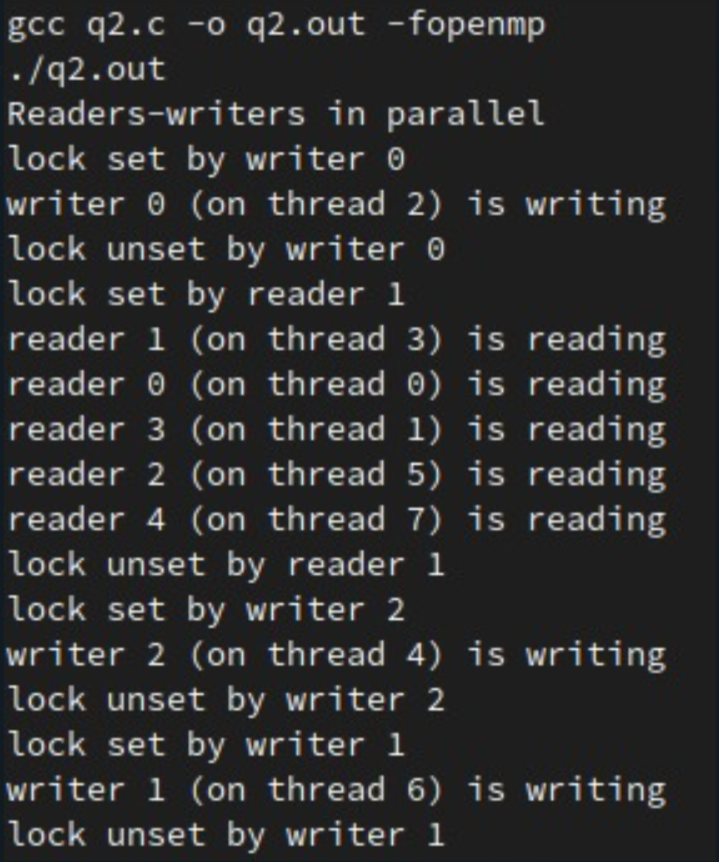
        }

    }

    return 0;

}

**Output:**

****