**Class: Third Year (Computer Science and Engineering)**

**Year: 2025-26 | Semester: Odd**

**Course: Cutting Edge Technologies Lab**

**Course Code: 7CS352**

**Practical No.: 5**

**Exam Seat No.:23510030**

**Title of Practical: Study and Implementation of Synchronization Constructs: critical, barrier, atomic, ordered, nowait**

**Problem 1: Incrementing a Shared Counter**

**Objective:**  
Observe the effect of synchronization constructs in OpenMP while multiple threads increment a shared counter.

**Code:**

**Without Synchronization**

#include <stdio.h>

#include <omp.h>

int main() {

int counter = 0;

#pragma omp parallel for

for (int i = 0; i < 1000; i++) {

counter++;

}

printf("Counter without sync: %d\n", counter);

return 0;

}

**With #pragma omp critical**

#include <stdio.h>

#include <omp.h>

int main() {

int counter = 0;

#pragma omp parallel for

for (int i = 0; i < 1000; i++) {

#pragma omp critical

counter++;

}

printf("Counter with critical: %d\n", counter);

return 0;

}

**Screenshots:**

* Output of both versions showing difference.

**Information:**

* Threads increment a shared variable.
* Without critical: race condition → incorrect result.
* With critical: ensures only one thread updates at a time → correct result.

**Analysis:**

* Compare final counter value.
* Discuss overhead of critical (slower due to thread locking).
* Optional: Measure speedup using omp\_get\_wtime().

**Problem 2: Sum of Array of N = 10⁶ Numbers**

**Objective:**  
Compare atomic vs critical for updating a shared sum in parallel.

**Code (Using #pragma omp atomic):**

#include <stdio.h>

#include <omp.h>

#define N 1000000

int main() {

int arr[N];

long long sum = 0;

for (int i = 0; i < N; i++) arr[i] = i+1;

double start = omp\_get\_wtime();

#pragma omp parallel for

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

#pragma omp atomic

sum += arr[i];

}

double end = omp\_get\_wtime();

printf("Sum using atomic: %lld\n", sum);

printf("Time: %f seconds\n", end-start);

return 0;

}

**Using #pragma omp critical:**

#pragma omp parallel for

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

#pragma omp critical

sum += arr[i];

}

**Screenshots:**

* Output and execution time for both versions.

**Information & Analysis:**

* Atomic is lighter than critical → faster.
* Critical ensures correctness but may reduce parallel efficiency.

**Problem 3: Square of Thread IDs with Barrier**

**Code:**

#include <stdio.h>

#include <omp.h>

int main() {

int squares[10]; // assuming max 10 threads

#pragma omp parallel

{

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

squares[tid] = tid \* tid;

#pragma omp barrier

#pragma omp master

{

printf("Squares of thread IDs: ");

for (int i = 0; i < omp\_get\_num\_threads(); i++)

printf("%d ", squares[i]);

printf("\n");

}

}

return 0;

}

**Screenshots:**

* Output of squares.

**Analysis:**

* Barrier ensures all threads complete calculation before printing.
* Without barrier, master may print incomplete results.

**Problem 4: Ordered Printing**

**Code:**

#include <stdio.h>

#include <omp.h>

int main() {

#pragma omp parallel for ordered

for (int i = 1; i <= 20; i++) {

#pragma omp ordered

printf("%d ", i);

}

printf("\n");

return 0;

}

**Without ordered** → may print numbers out of sequence.

**Screenshots & Analysis:**

* Compare outputs.
* Ordered ensures sequential order even in parallel loops.

**Problem 5: Nowait Example**

**Code:**

#include <stdio.h>

#include <omp.h>

#define N 10

int main() {

int arr[N];

#pragma omp parallel

{

#pragma omp for nowait

for(int i=0;i<N;i++)

arr[i] = i;

#pragma omp for

for(int i=0;i<N;i++)

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

**Observations:**

* Without nowait → correct order.
* With nowait → may print uninitialized elements.

**Problem 6: Parallel Factorial**

**Code using reduction:**

#include <stdio.h>

#include <omp.h>

int main() {

int n = 10;

long long fact = 1;

#pragma omp parallel for reduction(\*:fact)

for (int i = 1; i <= n; i++)

fact \*= i;

printf("Factorial of %d: %lld\n", n, fact);

return 0;

}

**Analysis:**

* Reduction avoids need for critical/atomic.
* Faster and cleaner approach.

**Problem 7: Sum of Squares of First 1 Million Numbers**

**Code:**

#include <stdio.h>

#include <omp.h>

#define N 1000000

int main() {

long long sum = 0;

#pragma omp parallel for reduction(+:sum)

for (int i=1; i<=N; i++)

sum += i\*i;

printf("Sum of squares: %lld\n", sum);

return 0;

}

**Screenshots, Information, Analysis for All Problems**

* Include screenshots of outputs.
* Record execution times with different number of threads.
* **Speedup calculation:**

Speedup=Tparallel​/Tserial​​

**Github Link**