Activity No. 13	
Parallel Algorithms and Multithreading	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 02/11/2025**
Section: CPE21S4	**Date Submitted: 02/11/2025**
Name(s): Crishen Luper S. Pulgado	**Instructor:** Engr. Jimlord Quejado

A. Output(s) and Observation(s)

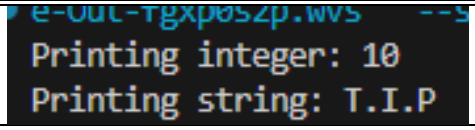
Screenshot	
Analysis	Based on the code, a function is created so that the thread named t1 will work on that function. The main thread pauses and waits until the new thread t1 is finished executing the print function and that's what the purpose of the join(). After the print function completes, the t1 thread end itself and the main thread continues after the join() and finishes the program. If the .join() is not called, it will crash the program since the main function could end while the thread t1 is still running.

Table 13-1. Simple One-Threaded Example

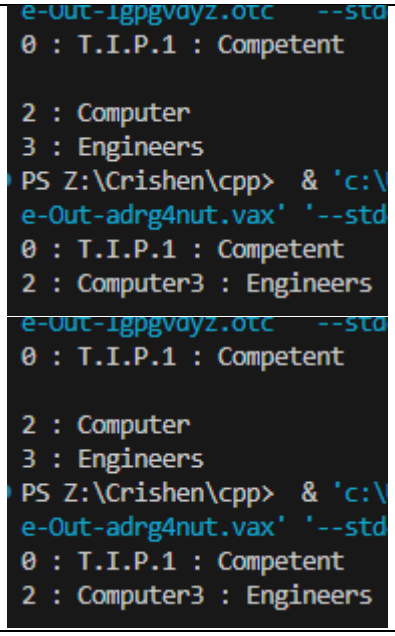
Screenshot	
Analysis	Based on the code, it creates different threads for each string and will be printed by different threads. All of the threads start executing concurrently. The other for loop is for the join() function which ensures that the main thread waits for each thread to finish before continuing. This shows how multiple threads can execute the same function with different data. However upon execution order, it may vary each time since threads runs concurrently which runs independently and the CPU decides their scheduling.

Table 13-2. Multithread Example

B. Answers to Supplementary Activity

1.) **Write a definition of multithreading and its advantages/disadvantages.**

Multithreading is a process in which a program is broken down into smaller blocks of execution referred to as threads. The threads are independent of each ad share resources such as memory whereby the tasks can be carried out concurrently. This is useful in enhancing the performance through the use of multiple CPU cores efficiently. The advantages of multithreading are it leverages multiple CPU cores to execute tasks in parallel which reduces the overall execution time. Runs tasks in the background without stopping the main thread, which keeps apps responsive. It makes it easier to handle a lot of work or many tasks at once, like in computers or real-time systems. Whereas, the disadvantages of multithreading are where it raises a problem such as deadlock, race condition, and starvation. Deadlock occurs when two or more threads are blocked since they are each waiting for shared resources that the other threads hold. Race condition is when two or more threads simultaneously access shared resources and at least one of them modifies the resource. Because the threads compete to read and write the data, the sequence in which they execute determines the outcome, which can be unpredictable or inaccurate. A thread experiences starvation when it is consistently denied access to shared resources due to other threads gaining priority, which stops it from running and making progress (GeeksforGeeks, 2025b).

2.) **Rationalize the use of multithreading by providing at least 3 use-cases.**

One of the use or applications of multithreading is it increases responsiveness in applications. A multithreaded web browser allows user to interact with part of a webpage while a video or content loads in another thread. Another case that uses multithreading is for the economy resources since threads share memory and resources of the process which avoids the need for separate allocations. Example is in Solaris, creating a process is 30 times slower than creating a thread and context switching between processes is 5 times slower than switching between threads. Another case is that we may have a blocking I/O that slows down an application. Through using another thread where it handles the I/O may speed up the process (GeeksforGeeks, 2025b).

3.) **Differentiate between parallelism and concurrency.**

Concurrency is the ability to handle several tasks at once without necessarily carrying them out at the same time. Rather, by sharing time on the same processing resource, tasks advance. Concurrency is a technique that uses a single processing unit to reduce the system's response time. While parallelism is the execution of numerous tasks concurrently on multiple processing units. In this case, tasks are broken down into smaller subtasks that appear to be processed simultaneously or in parallel. It uses several processors to boost the system's throughput and computing speed (GeeksforGeeks, 2025).

Part B.

Screenshot

```cpp
#include <iostream>
#include <thread>
#include <mutex>

int globalVar = 0;

std::mutex mtx;

void add(int value) {
    std::lock_guard<std::mutex> lock(mtx);
    globalVar += value;
    std::cout << "Thread adding " << value << ", globalVar = " << globalVar << std::endl;
}

int main() {
    std::thread t1(add, 5);
    std::thread t2(add, 10);
    std::thread t3(add, 15);

    std::cout << "\n[Before any join] globalVar = " << globalVar << std::endl;

    t1.join();
    std::cout << "[After T1.join()] globalVar = " << globalVar << std::endl;

    t2.join();
    std::cout << "[After T2.join()] globalVar = " << globalVar << std::endl;

    t3.join();
    std::cout << "[After T3.join()] globalVar = " << globalVar << std::endl;

    return 0;
}
```

```
[Before any join] globalVar = 0
Thread adding 5, globalVar = 5
Thread adding 10, globalVar = 15
Thread adding 15[After T1.join()] globalVar = 30
, globalVar = 30
[After T2.join()] globalVar = 30
[After T3.join()] globalVar = 30
PS Z:\Crishen\cpp> ^C
PS Z:\Crishen\cpp>  & 'c:\Users\user\.vscode\extens
e-Out-udvd2a5v.kda' '--stderr=Microsoft-MIEngine-Er
oft-MIEngine-Pid-osd5mqft.ifl' '--dbgExe=C:\x5cmsys
[Before any join] globalVar = 0
Thread adding 5, globalVar = 5
Thread adding 10, globalVar = 15
Thread adding 15, globalVar = 30
[After T1.join()] globalVar = 30
[After T2.join()] globalVar = 30
[After T3.join()] globalVar = 30
PS Z:\Crishen\cpp>
```

Analysis

In the code, I added mutex for safety threading access. What happens in the code is that the global variable is shared by all threads. The function add executes for each of the threads, adding its parameter value to the global variable. The mutex is used to prevent data races which two threads changing the global variable at the same time. So, the t1 thread adds 5, t2 adds 10, and t3 adds 15. Before the join function, the global variable may still be 0 or partially updated since threads might not have finished yet. After t1 join, it is guaranteed that t1 finishes executing the function but others may still be running. When t2 joined, both t1 and t2 are finished which the

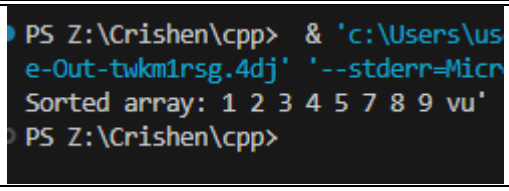
	global variable includes both additions. When t3 is finished, all of the threads are finished and the total sum is 30.

Part C:

Code

```
#include <iostream>
#include <vector>
#include <thread>

void merge(std::vector<int>& arr, int left, int mid, int right) {
    std::vector<int> temp;
    int i = left, j = mid + 1;

    while (i <= mid && j <= right) {
        if (arr[i] <= arr[j]) temp.push_back(arr[i++]);
        else temp.push_back(arr[j++]);
    }

    while (i <= mid) temp.push_back(arr[i++]);
    while (j <= right) temp.push_back(arr[j++]);

    for (int k = 0; k < temp.size(); k++) {
        arr[left + k] = temp[k];
    }
}

void mergeSort(std::vector<int>& arr, int left, int right) {
    if (left >= right) return;
    int mid = (left + right) / 2;

    std::thread t1([&]() { mergeSort(arr, left, mid); });
    std::thread t2([&]() { mergeSort(arr, mid + 1, right); });

    t1.join();
    t2.join();

    merge(arr, left, mid, right);
}
```

```
int main() {
    std::vector<int> arr = {8, 3, 7, 4, 2, 9, 5, 1};

    mergeSort(arr, 0, arr.size() - 1);

    std::cout << "Sorted array: ";
    for (int n : arr) std::cout << n << " ";
    std::cout << std::endl;

    return 0;
}
```

Output		

Analysis:
I choose merge sort as for applying multithreading technique since the merger sort is using a divide and conquer algorithm. The merge sort splits the array into halves and sorts it separately. This means that they do not depend on each other's result while sorting so it does not cause a data race. Applying a multithreading can sort both of the halves of an array simultaneously since on a normal merge sort, the left is usually sorted first the next is the right part. What happens in the code is that, the array is split into half; for the left side is [8,3,7,4]; the right is [2,9,5,1]. The thread t1 sorts the left half and the thread t2 sorts the right half. While t1 is dividing and sorting the left side, t2 is doing the same for the right side at the same time. The & means they capture the variables by reference allowing the threads to modify the same array. Now, the main thread waits the t1 and t2 to finish by using the join function. After both sides are sorted, they merge together. Hence, the threading allows to work both halves of the array in parallel.

C. Conclusion & Lessons Learned

In conclusion, this activity demonstrates how multithreading works. Through multithreading, we can achieve concurrency and parallelism. Concurrency means that multiple tasks are in progress at once but not running at the same instant. While parallelism means working on a task that runs at the same time on different CPU cores. With multithreading, we can make the program or an application execute a task simultaneously which allows us to make more time efficient as well as the space. In the procedure, it demonstrates how multithreading works and I observe that the results of the execution changes as the code runs since it runs concurrently so it depends which one finishes it first. We can also apply multithreading on different algorithms such as merge sort so that we can sort them both of the half of the array at the same time. In this activity, I think I learned well the importance of multithreading and how is it applied. I think I face a challenge in the application of multithreading in an algorithm.

D. Assessment Rubric

E. External References

GeeksforGeeks. (2025, August 7). Difference between Concurrency and Parallelism. GeeksforGeeks. https://www.geeksforgeeks.org/operating-systems/difference-between-concurrency-and-parallelism/

GeeksforGeeks. (2025b, October 3). *Multithreading in C++*. GeeksforGeeks. https://www.geeksforgeeks.org/cpp/multithreading-in-cpp/

GeeksforGeeks. (2025b, August 29). *Multithreading in OS different models*. GeeksforGeeks. https://www.geeksforgeeks.org/operating-systems/multithreading-in-operating-system/