بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

An-Najah National University

Faculty of Engineering and Information Technology

Computer Engineering Department

جامعة النجاح الوطنية

كلية الهندسة وتكنولوجيا المعلومات

قسم هندسة الحاسوب

Operating Systems

Assignment #2: 1st semester 2023/2024

Date: 22nd/October4/2023

Assignment Details:

- We have one large array of N items called "packet" of type "double" as a **shared memory** and two variables called "result_serial" and "result_parallel" of type "double".
- Initialize the items in the "packet" to random double value between 1 and N:
  - packet[i] = (double)rand()/(double)(RAND_MAX/N);
- Add all items in the array and store the result in "result_serial"
  - for(0→N) result += packet[i];
  - Compute the sum using simple loop, nothing special.
  - result_serial is in the parent process which is not shared.
- Perform the calculations in parallel:
  - Create M number of processes and use those processes to compute the sum.
  - Divide the problem between those the processes, each process would compute (N/M) items of the sum.
  - If the array does not split evenly between processes, then you can just do the remaining few items in the parent process, or make the last process do few more items.
  - The parent process which started everything waits for all processes to finish.
  - Finally, the parent process collects all partial sums from the sub-processes in "result_parallel" using message passing (check mkfifo):
    - for(0 → M) { get partial_sum from fifo; result_parallel+=partial_sum; }
- Make sure the parallel_result == serial_result
- N is your student ID and M is the right most digit of your student ID + 5.
- Measure the performance of each approach:
  - Display the measured performance/timing results at the end of the execution.
  - Be able to explain your measured values: is the parallel version faster? It might not be, specially on virtual machines with 1 core.

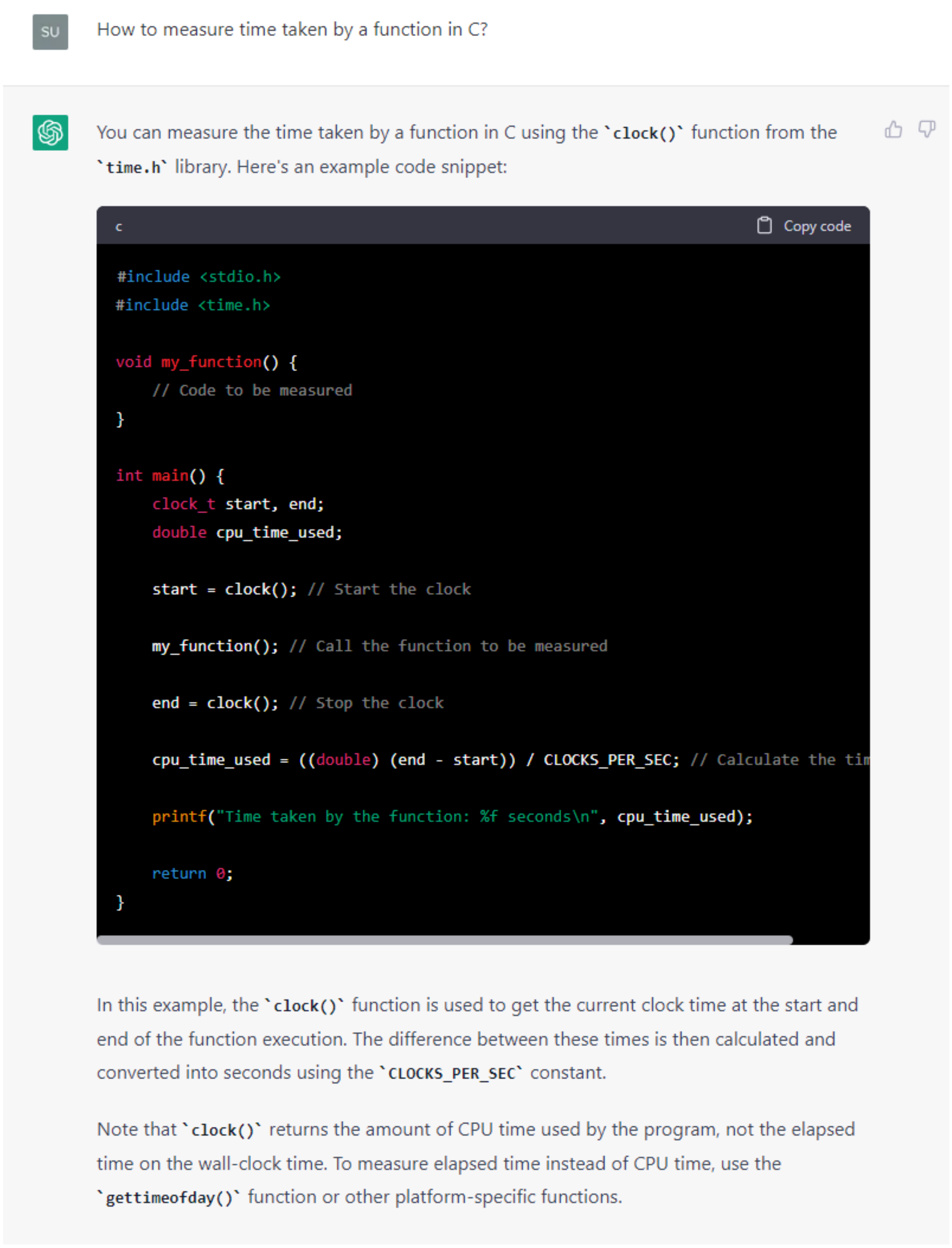
بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Submission Rules:

- You should just submit you working code C file(s) for this assignment.
- 1-minute video demonstration of your working program, and explanation of how the program works (maximum 1 minute):
  - Or a one page report explaining your work instead of the video/demo.
- Optional: try to make your parallel version as fast as possible compared to your serial version:
  - You are free to change the number of processes (M) as you see necessary.
  - Increase number of cores? What happens if you exceed the physical number of cores on your machine.
  - Reduce number of processes (M) to match number of cores?
  - Change the size of the whole packet? Check the cache dimensions in the CPU.
- If you manage to improve the performance, then you should explain how.
- If you change your code parameters to make the parallel version faster, then you must submit two versions: Original M as specified in the assignment AND updated/optimized version

- Deadline is two weeks from today: Sunday 5th/November/2023
  - Assignment weight is 5% from the course's total mark.
- Important note regarding late submissions:
  - Maximum of 1 week late (7 days, until 12th/November), after that no submission will be accepted.
  - 5% per day penalty for late submissions (0.25 mark per day).
  - If you are 7 days late, you would loose 1.75 marks.
- You can work in teams of two **maximum:**
  - Use (StudenID1+StudentID2)/2 as your source for M and N.

Measuring performance:



SU

How to measure time taken by a function in C?

You can measure the time taken by a function in C using the `clock()` function from the `time.h` library. Here's an example code snippet:

```c
#include <stdio.h>
#include <time.h>

void my_function() {
    // Code to be measured
}

int main() {
    clock_t start, end;
    double cpu_time_used;

    start = clock(); // Start the clock

    my_function(); // Call the function to be measured

    end = clock(); // Stop the clock

    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC; // Calculate the ti

    printf("Time taken by the function: %f seconds\n", cpu_time_used);

    return 0;
}
```

In this example, the `clock()` function is used to get the current clock time at the start and end of the function execution. The difference between these times is then calculated and converted into seconds using the `CLOCKS_PER_SEC` constant.

Note that `clock()` returns the amount of CPU time used by the program, not the elapsed time on the wall-clock time. To measure elapsed time instead of CPU time, use the `gettimeofday()` function or other platform-specific functions.

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

An-Najah
National University

Faculty of Engineering and Information Technology

Computer Engineering Department

جامعـة
النجـاح الوطنيـة
كلية الهندسة وتكنولوجيا المعلومات
قسم هندسة الحاسوب

Message Passing Example/Template:

```
// C program to implement one side of FIFO:
// PRODUCER
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;

    // FIFO file path
    char * myfifo = "myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>, <permission>)
    mkfifo(myfifo, 0666);

    char arr[80];
    while (1)
    {
        // Take an input "arr" from user
        // (buffer size is 80 for simplicity)
         fgets(arr, 80, stdin);

         // Open FIFO for write only,
        // write the input "arr" on FIFO and close it
         fd = open(myfifo, O_WRONLY);
         write(fd, arr, strlen(arr)+1);
         close(fd);
    }
    return 0;
}
```

```
// C program to implement one side of FIFO:
// CONSUMER
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd1;

    // FIFO file path
    char * myfifo = "myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>,<permission>)
    mkfifo(myfifo, 0666);

    char str[80];
    while (1)
    {
        // First open in read only,
       // read and close the fifo
        fd1 = open(myfifo,O_RDONLY);
        read(fd1, str, 80);
       close(fd1);

        // Print the read string
        printf("Sender: %s\n", str);

    }
    return 0;
}
```