بسم الله الرحمن الرحيم

**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 2024/2025, Date: 23th/October/2024**

# Assignment Details: Goal is to demonstrate the performance of various Inter Process Communication techniques (part1).

- We have two large arrays of N items called "packet1" and "packet2" of type "double" declared and randomly initialized in main/parent process. The goal is to add those two packets and produce a third packet of size N too.
- Initialize the items in the two packets as follows:
  - "packet1" to random double value between 0 and N:
    - packet1[i] = (double)rand()/(double)(RAND_MAX/N);
  - "packet2" to random double value between 0 and 10:
    - packet2[i] = (double)rand()/(double)(RAND_MAX/10);

1. Add all items in the array serially and store the result in "result_packet1"

   Start-timer

   - for(0→N) result_packet1[i] = pow(packet1[i], packet2[i]);

   End-timer

   - Compute using simple loop, nothing special.
   - result_packet1 is in the parent process and is not shared.
     - Measure the performance of this technique: (end timer – start timer).
   - take a timestamp at the start of the loop and another timestamp at the end of the loop and subtract the two. See section (Measuring performance) for more details.

2. Perform the calculations in parallel and send the results to the parent process using shared memory:
   Start-timer
   - Create M number of processes and use those processes to perform the calculations.
   - Divide the problem between those processes, each process would compute (N/M) items.
   - The result array is declared as shared memory in the parent process ("result_packet2") and all child processes must open it and start writing their part to this shared memory.
   - 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.
   - Wait for all processes to finish.

   End-timer

- Make sure the result_packet1 == result_packet2 (you need a loop for this).
- Display the performance of this approach too (end timer – start timer).

3. Repeat step 2 but using message passing (check the mkfifo example on the slides).
   - Store the results in an array called result_packet3.
   - Make sure the result_packet1 == result_packet3 (you need a loop for this)
   - Measure the performance of this approach too.

## Results collection:

- Display the measured performance/timing results of each approach at the end of the execution.
- Be able to explain your measured values: is the parallel version faster? why?

## Submission Rules:

- N is your student ID and M is the right most digit of your student ID + 5.
- You should 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 (1-2 minute):
  - Or a report explaining your work instead of the video/demo (1-2 page).
- Optional: try to make your parallel version (in step 2, a.k.a shared memory) as fast as possible:
  - You are free to change the number of processes (M) as you see necessary.
  - Increase number of cores (in the case of a virtual machine)? What happens if you exceed the physical number of cores on your machine.
  - Reduce number of processes (M) to match number of cores?
  - Change how the packets are split between processes? 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 as specified in the assignment AND updated/optimized version

- Deadline is 1 week from today: Friday 1st/November/2024

- Assignment weight is 5% from the course's total mark.

- Important note regarding late submissions:
  - Maximum of 1 week late (7 days, until 8th/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 Copy code

```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 tim

    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.