Task 5: Mmap vs. Pipes vs. Threads

Jana Saleh 900204192
Mariam Dahab 900192441
Muhammad Azzazy 900202821

TABLE OF CONTENTS

TABLE OF CONTENTS

ROLES

Role of every team member

Name	Role
Jana Saleh	Implemented threads_compute
Mariam Dahab	Plotted the graphs comparing between the four compute functions and implemented the second part of the bonus
Muhammad Azzazy	Implemented mmap_compute

02

I/O OF MMAP

Inputs and outputs of mmap

INPUTS OF MMAP

The inputs of mmap are in the following order:

- The address is set to 0 to allow the system to map anywhere it chooses.
- The size is in our case (N+2)*unsigned long int, where N is the number of items in the file.
- Some flags such as the following:
  - PROT_READ indicates whether the pages can be read from.
  - PROT_WRITE indicates whether the pages can be written to.
  - MAP_SHARED indicates whether the map is shared with the different processes.
  - MAP_ANONYMOUS indicates that the mapping is not backed by a file and the file descriptor is ignored.
- The file descriptor is set 0 for standard input.
- The offset which must be an integer multiple of the page size is set to 0, but since there is no file it does not matter.

OUTPUTS OF MMAP

The outputs of mmap are in the following order:

- The numbers stored within the file
- The results
- The number of operations done

03

PSEUDOCODE

Pseudocode of the four compute functions

PSEUDOCODE OF MMAP_COMPUTE

```
MMAP_COMPUTE(n_proc, filepath[], f)
        result ← 0
        Number* head ← NULL
        FILE* fh ← fopen(filepath, “r”)
        N ← 0
        while(!feof(fh))
        do
                fscanf(fh, “%lu”, &val)
                append(&head, val)
                N ← N + 1
        unsigned long int *ptr ← mmap(NULL, (N+n_proc+1)*sizeof(unsigned long int), PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, 0, 0)
    if ptr == MAP_FAILED
        then printf("Mapping Failed\n")
    for i ← 1 to N
        do
            ptr[i] ← head->num
            head ← head->next
    for i ← 1 to n_proc-1
        do
                pid_t child_pid ← fork()
```

PSEUDOCODE OF MMAP_COMPUTE

```
        if child_pid == 0
                then If (N-1)/n_proc == 0
                        then ptr[N+i+1] ← (*f)(ptr[N+i], ptr[N+i+1])
                        else
                                for j ← 1 to (N-1)/n_proc
                                        do
                                                ptr[N+i+j+1] ← (*f)(ptr[N+i+j], ptr[N+i+j+1])
                                                ptr[N+n_proc+1] ← ptr[N+n_proc+1]+1
                        exit(0)

wait(NULL)
for k ← ptr[N+n_proc] to N-1
                        do
                                        ptr[N+n_proc] ← ptr[N+n_proc] + 1
                ptr[k+1] ← (*f)(ptr[k], ptr[k+1])
result ← ptr[N]
munmap(ptr, (N+1)*(sizeof(unsigned long int)))
return result
```

PSEUDOCODE OF THREADS_COMPUTE

```
threads_compute( filepath, threads_number, (*f)(int, int)) {
  file = fopen(filepath, "r")
  while (fscanf(file, "%d", &m) != EOF)
    nums[i] = m
    i++
    nums = realloc(nums, (i + 1) * sizeof(int))
    fclose(file)
   N = i
  numbers_per_thread = N / threads_number
  for i ← 0 to  threads_number
    first_num_inthread = i * numbers_per_thread
    patialsum= 0
    if i == threads_number - 1
      last_num_inthread = N
     else
      last_num_inthread = (i + 1) * numbers_per_thread
```

```
for int i ← 0 to threads_number
       for int j = first_num_inthread to th[i].last
       numbers_in_thread[count] = numbers[j]
       count++

// create threads
  for i ← 0  i to threads_number
    pthread_create(&threads[i], NULL, threadfun, (void
*)&th[i])
  // wait for threads to terminate
  for i ← 0 to  n_threads
    pthread_join(threads[i], NULL)
     pthread_mutex_destroy(&mut);
  // add the sums of each thread and store it in
sum_threads
  for i ← 0  to n_threads)
    total_ofThreads = f(total_ofThreads, partialsum)
  return total_ofThreads
```

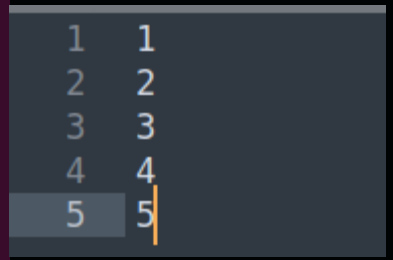
04

EXAMPLES

Simple examples of the four compute functions producing correct results

MMAP_COMPUTE EXAMPLE

```
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ cc main.c
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 1 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 2 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 3 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 4 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 5 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 6 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 7 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 8 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 9 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 10 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 11 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 12 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 13 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 14 numbers.txt add
15
muhammad@muhammad-OMEN-Laptop-15-en0xxx:~/Desktop$ ./a.out 15 numbers.txt add
15
```



EXAMPLE OF THREAD_COMPUTE

data.txt

```
1   1
2   2
3   3
4   4
5   5
6   6
7   7
8   8
9   9
10  10
11  10
12  10
13  10
```

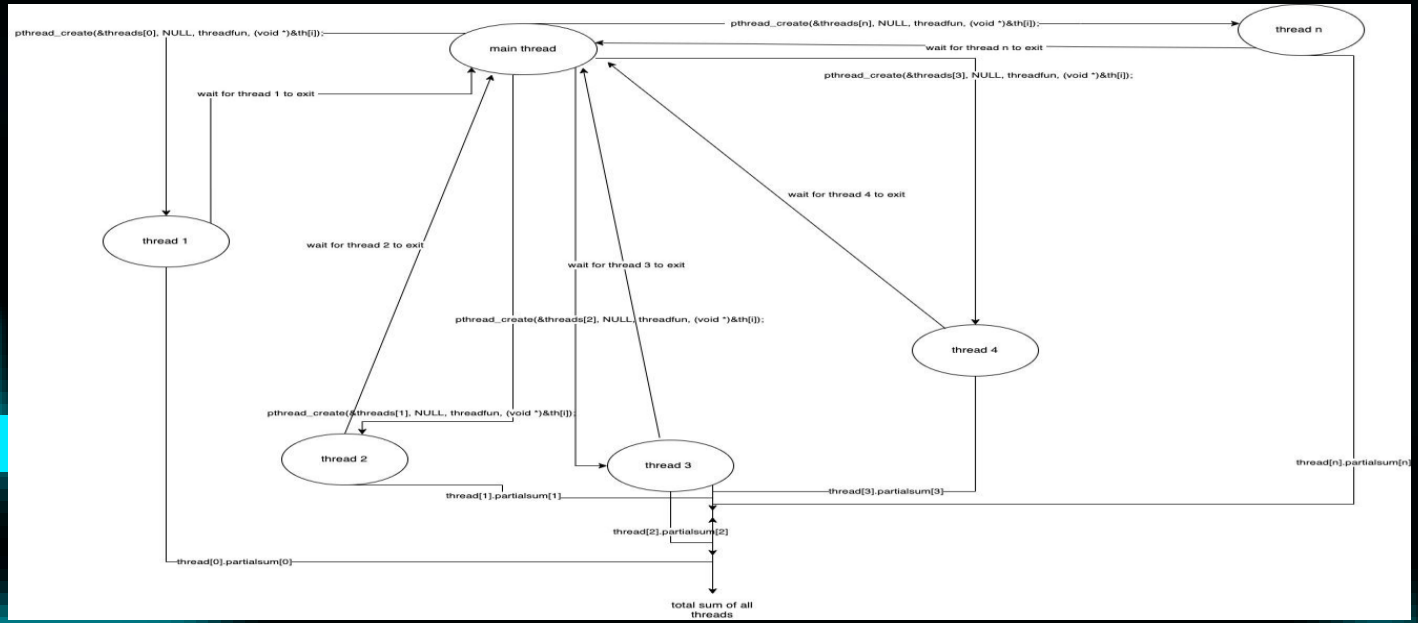
```
./main
result is 85
thread Computation took 0.013991 seconds to execute
```

05

FIGURES

Figures illustrating the processes, threads, pipes, and the flow of data

Figure of threads dataflow



06

TESTS DESIGN

How the two tests were designed and the ranges chosen

- In the case of fixing N . The range of n_procs was chosen based on the capabilities of the computer (the number of cores) . All functions except sequential where given the same range of *n_procs* and all of them were give the same file containing *N* data.

- In case of fixing *n_procs* . The range of *N* was set from the least value, which is 10, to the maximum value the computer can handle, which is 10,000. Again in each run, all functions where given the same file containing the same set of values from consistency.

- In both cases the time of execution was gathered 10 times for each combination of numbers are the average was used as the final answer.

07

FIXING VALUES

Why was each value chosen

- The value of *n_proc* (n_threads) was chosen based on the number of cores in the computer.

- In this case the computer has a quad-core processor, and, therefore, the maximum number of threads that would have an effect is 4.

- Similarly *N* was set to the maximum number that could be handled by the computer which is 10,000.

08

SMOOTHING

By running each set of values 10 times and taking the average . We were able to create a smooth curve for each line in the curve.

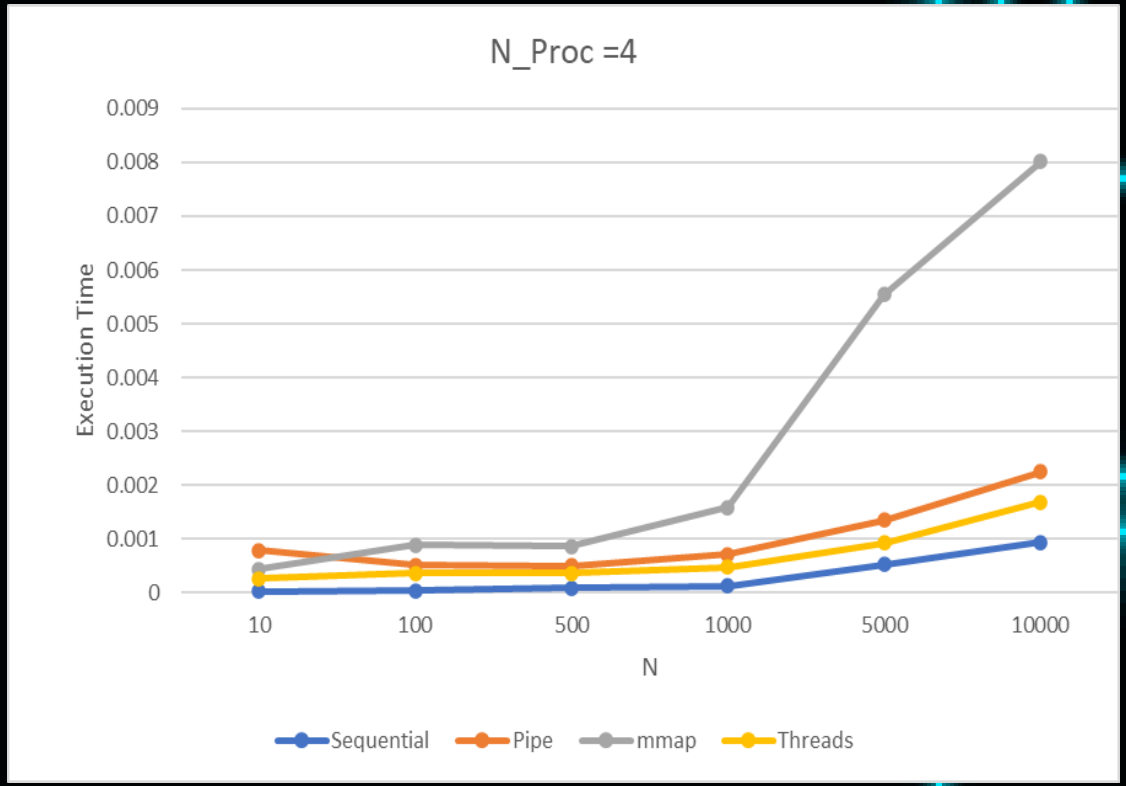
09

RESULTS

Comparing functions using the graphs

Fixing n_proc / n_threads to 4

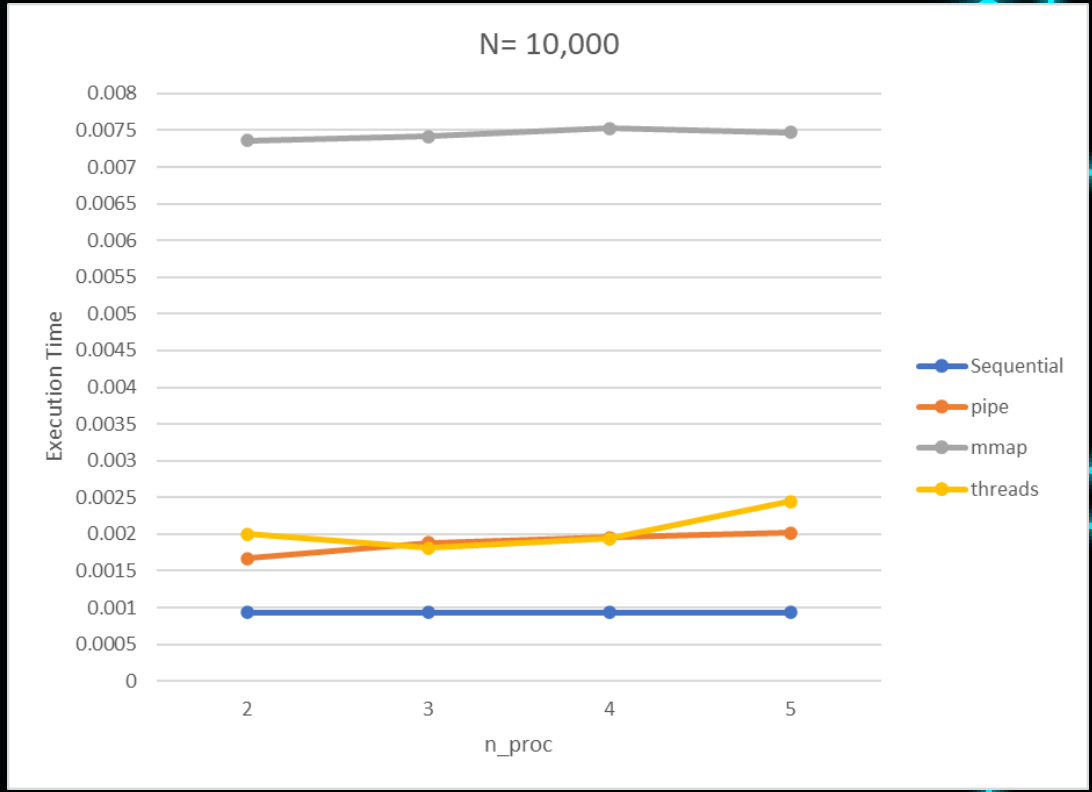
- Although we expected that one of the pipe/thread functions would be the fastest since the data is divided and processed concurrently.
- However, the results different and the sequential took the least time out of the 4 from the beginning.
- Again this is due to the fact that the number of cores limited the dividing of the data and we still believe that using different devices we may get different result.



Fixing N to 10,0000

- Although we expected that one of the pipe/thread functions would be the fastest since the data is divided and processed concurrently.

- However, similar to the previous graph, the results came different and the sequential took the least time out of the 4 from the beginning.

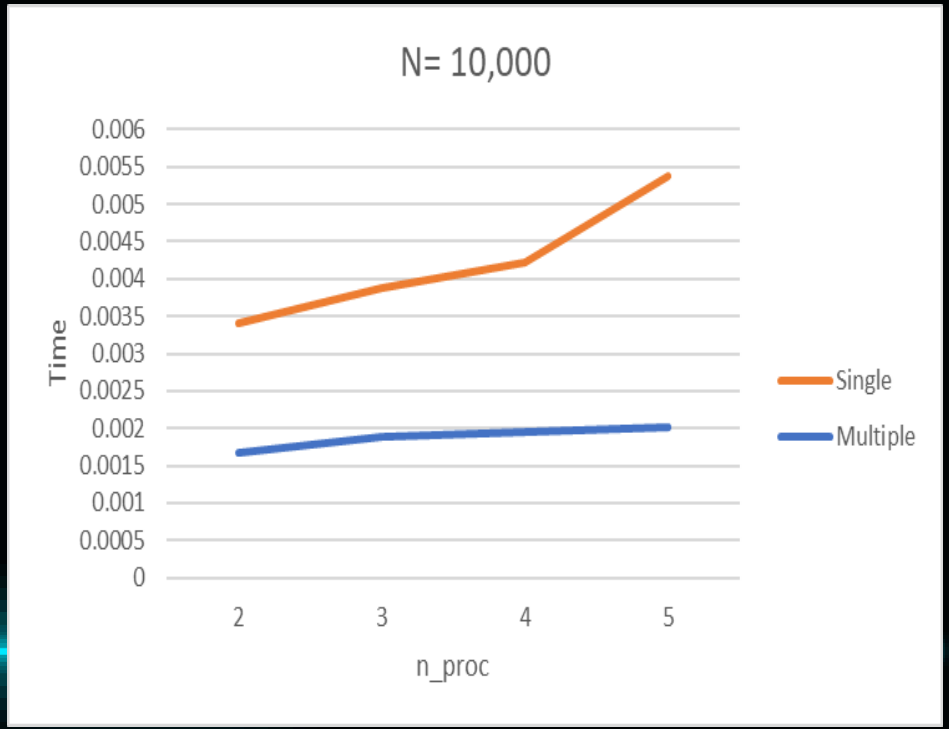


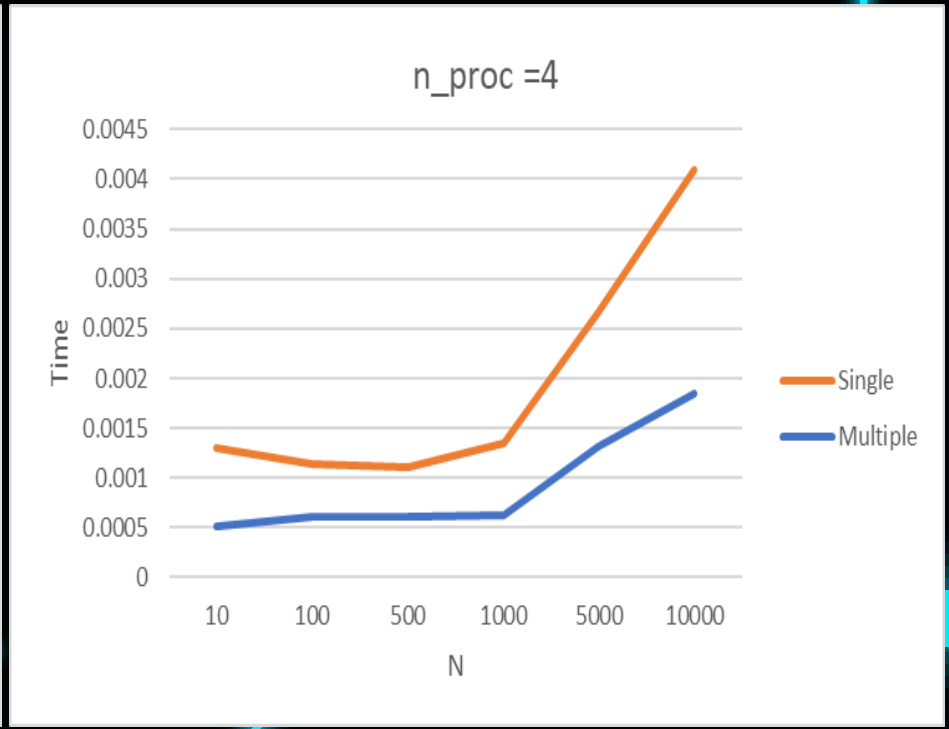
10

BONUS

Comparing functions using the graphs

Since the single pipe function makes the parent until the created child is done before creating another one it is obvious that in both cases (fixing N and n_proc) the multiple pipe function would outpreform since the since can’t be considered as parallel computation





THANKS!

janasaleh@aucegypt.edu
mhdahab@aucegypt.edu
muhammad-azzazy@aucegypt.edu