Process Coordination

Introduction

The goal of this project is to have a clear understanding on the concepts of process coordination, pthreads and semaphores.

The project contains 3 problems which are as follows:

1. Editor-Reporters Problem
2. Dish Washing Problem
3. Concurrent Merge Sort

Problem Description

Editor-Reporters Problem

- There are multiple editors and multiple reports
- Each editor can select an available report and choose to publish it or reject it
- Once a report is published it will not be considered by any editor Once an editor rejects a report, it will not be considered again by that editor
- The goal is to parallelise this process as much as possible.

Dish Washing Problem

- Given a dish washing junction, there are m students, n taps, n-1 scrotch bites with soap-dishes
- A student can rub a utensil with scrotch brite and after that wash utensil with water
- There is no priority for the students, a student acquire the tap till all utensils get washed.
- When student get a tap, he looks for the scrotch brite in the left or right of his tap
- A student should coordinate with his neighbors to wash all utensils
- Arrangement of taps and scrotch brite is: TSTST
- A neighbour can use adjacent scrotch brite when a neighbouring student is using tap.

Utensils	**With scrotch brite**	**With tap/water**
Plates	4 units	5 units
Glasses	3 units	3 units
Spoons	2 units	1 unit

Concurrent Merge Sort

- Merge sort consists of splitting the array into halves and sorting the two halves
- The two halves are independent of each other, sorting them can be parallelised
- The goal is to parallelise merge sort and compare performance with normal merge sort

Solution Approach

Editor-Reporters Problem

- The chosen approach consists of using one semaphore for each report and one semaphore for the entire list of semaphores
- To check the available report, an editor needs to acquire the semaphore for the entire list
- To select a report to read, the editor then acquires the semaphore of that particular report
- Once the report is selected the semaphore for the entire list is released and after the editor decides on an action (accept/reject) and performs it, the semaphore of that particular report is released

Functions descriptions

- generate_random_number - function takes in a range and returns a random value within that range
- begin - function is called by each editor thread. Performs all actions of editors

Sample runs

Report No	Editor 1	Editor 2	Editor 3	Editor 4
1		Accepted	Rejected	
2			Accepted	Rejected
3	Rejected	Rejected	Rejected	Rejected
4		Accepted		
5		Accepted		
6	Accepted			
7			Accepted	
8	Rejected	Accepted	Rejected	Rejected
9	Rejected	Rejected	Accepted	Rejected
10	Rejected	Rejected	Rejected	Rejected

Table 1.1. Editors = 4 and Reports = 10

Report No	Editor 1	Editor 2	Editor 3	Editor 4	Editor 5	Editor 6	Editor 7
1			Rejected	Accepted			
2						Accepted	Rejected
3	Rejected	Rejected					Accepted
4				Accepted			
5					Accepted		
6	Rejected	Rejected	Rejected		Rejected		Accepted
7					Accepted	Rejected	

Table 1.2. Editors = 7 and Reports = 7

R No	E 1	E 2	E 3	E 4	E 5	E 6	E 7	E 8	E 9	E 10
1				A						
2	R	R						A		R
3	A									
4			A							

Table 1.3. Editors = 10 and Reports = 4

Dish Washing Problem

Current Approach

- Initialise semaphore for students and taps.
- Store the current time at start.
- Create threads for students.
- Check for the empty tap to acquire it.
- Check to the right for the scrotch brite is available or not, if available capture it otherwise look to the left. Keep doing it unless we get the scrotch brite.
- After getting the scrotch brite, use it on all utensils. Then release the scrotch brite.
- Wash the utensils and exit the function.
- Wait for the threads to join and then destroy semaphores.
- Print the Time taken by students from time 0.

Previous Approach Used

- Push all the students in the queue.
- Pop first n students from the queue and assign them the taps, n is the number of taps.
- Create threads for the popped students.
- If a student finds a scrotch brite on his left or right side then the student takes that up, uses it for plate, glass and spoon (9 units of time) and then leaves it.
- If a student does not find scrotch brite, then he should wait.
- After applying scrotch brite on all utensils, students release that and wash the utensils under tap.
- After washing utensils under tap, student leave the tap, another student who is in the waiting queue is assigned a new thread and assigned the tap
- All threads will work simultaneously, they have the same time of execution
- Mutex locks are used at the time of taking up scrotch brite, releasing scrotch time and releasing tap.

Function descriptions for Current Approach

- Begin - Will take care of the tap assignment and scrotch brite assignment. And release the resources after it is done.

Function descriptions for Previous Approach

- tapAssignment - This function allocates taps to first n students and creates threads for the same.
- task - This function is called when thread is created, it calculates the completion time for each student. It also handles mutex lock at the time of assigning scrotch brite, releasing scrotch brite and releasing taps.

Concurrent Merge Sort

- Merge sort was implemented using single thread, multi thread and multi process approaches
- When the array size falls below a threshold K, selection sort is performed
- For the multi-process approach, a shared memory was created between the parent, left child and parent, right child
- Runtime was calculated by running mergesort 10 times and calculating the average.

Functions used in Normal Merge Sort

- generate_random_number - This function generates a random number between start and end range provided in the arguments.
- merge_sort - This function performs the merge sort algorithm, it divides the array in two halves and performs a selection sort algorithm when size of array becomes less than k.

Functions used in multi-thread based Merge Sort

- generate_random_number - This function generates a random number between start and end range provided in the arguments.
- merge_sort - This function performs the merge sort algorithm, it divides the array in two halves and performs a selection sort algorithm when size of array becomes less than k.

Readings

No of elements	Single process	Multi thread	Multi process
100	0.000112	0.006318	0.000253
1000	0.000803	0.050867	0.000254
10000	0.060979	0.880375	0.000634

Table 3.1. K = 5

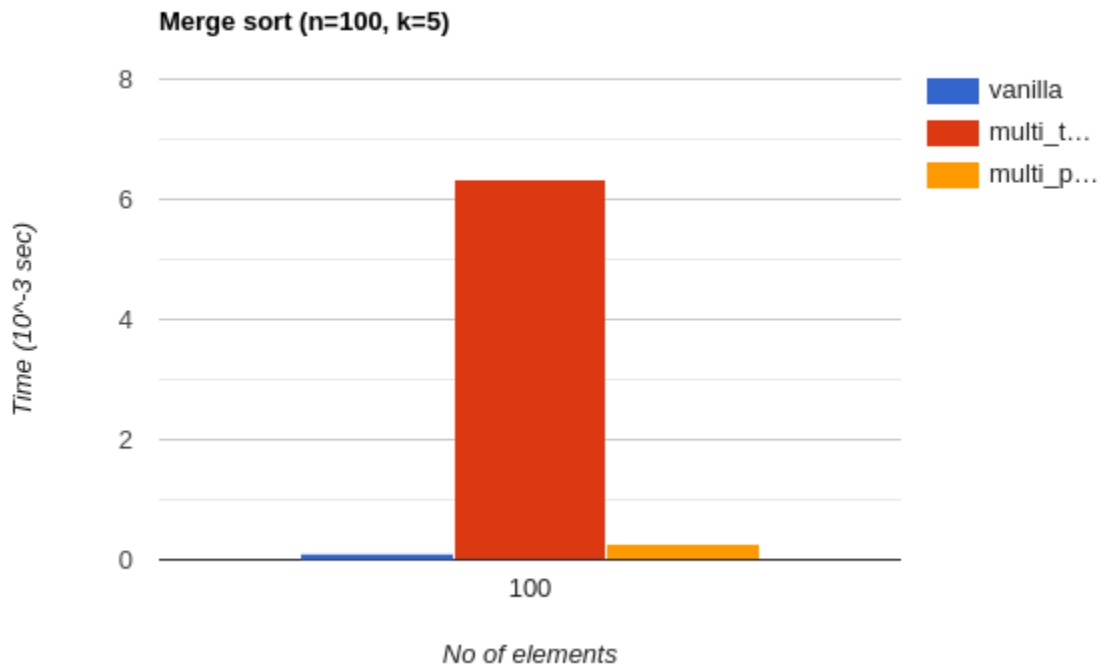


Fig 3.1 n=100 and k=5

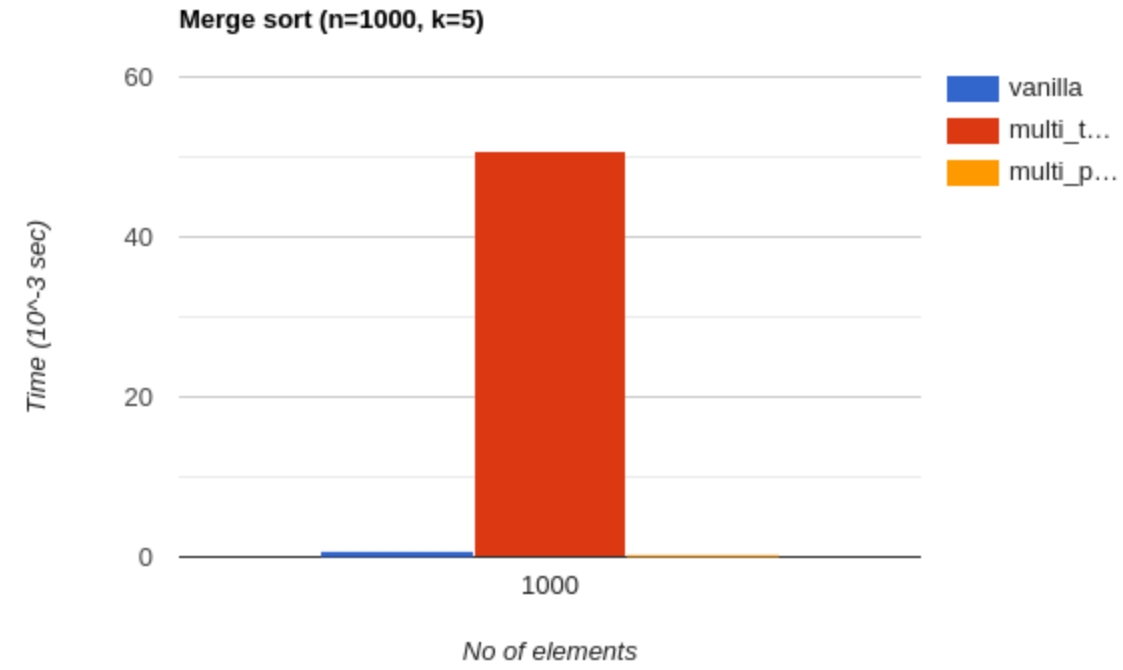


Fig 3.2 n=1000 and k=5

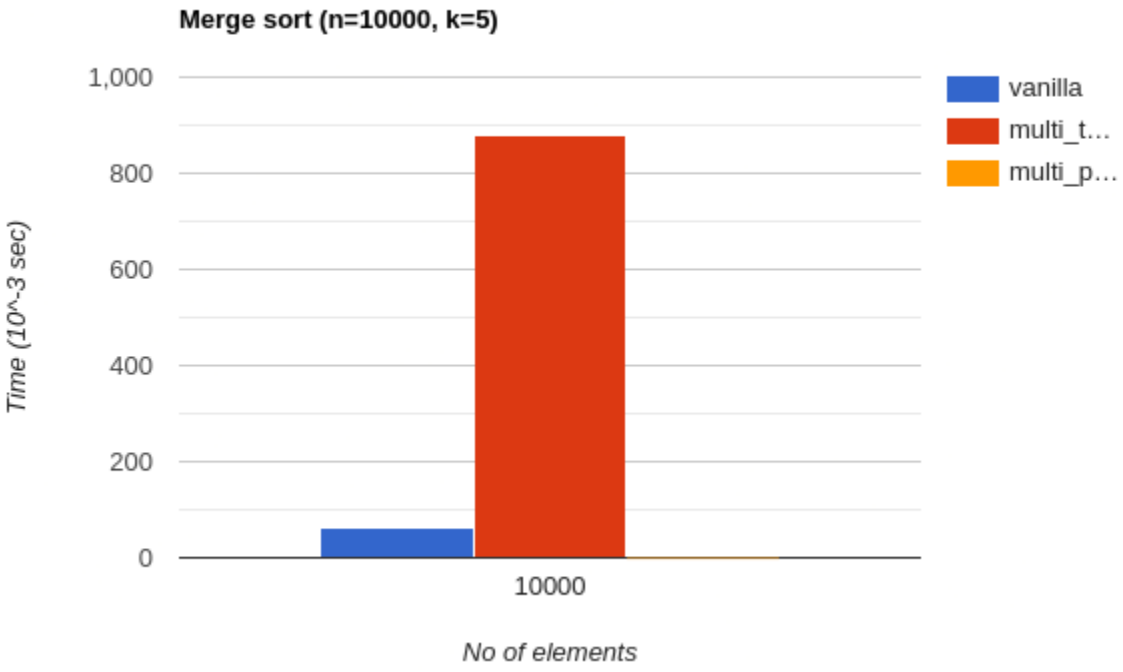


Fig 3.3 n=10000 and k=5

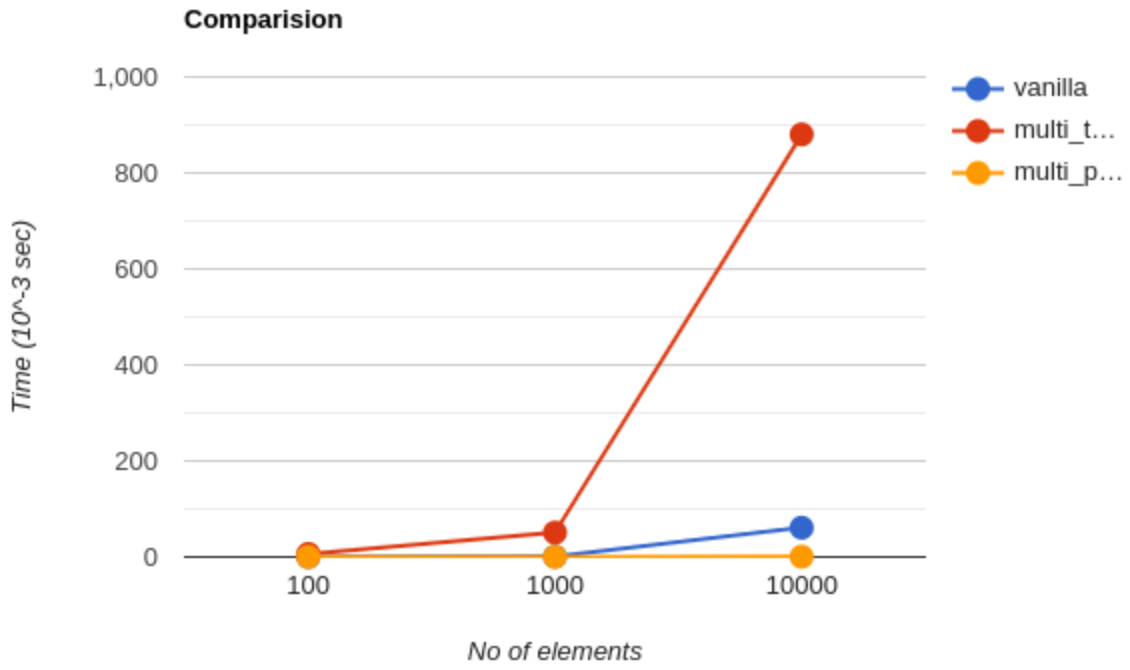


Fig 3.4 k=5

No of elements	Single process	Multi thread	Multi process
100	0.000073	0.000332	0.000339
1000	0.000668	0.006617	0.000232
10000	0.012214	0.089816	0.000502

Table 3.2 K = 50

No of elements	Single process	Multi thread	Multi process
100	0.000092	0.000255	0.000077
1000	0.002100	0.002861	0.000336
10000	0.013430	0.027644	0.000515

Table 3.3 K = 500

Inference

- Merge sort performs a lot worse when threads are used
- Of the 3 approaches, multi process gives the best performance

Work Distribution

Problem	Assigned to
Editor-Reporters Problem	Debashish Roy, Nithish Raja.
Dish Washing Problem	Debashish Roy, Purnima Grover, Shikha Raghuwanshi.
Concurrent Merge Sort	Nithish Raja, Purnima Grover, Shikha Raghuwanshi.

Problems Faced

Dish Washing Problem - We tried to optimize it for large values but the calculation of that without actually sleep() is very difficult. So at the end we have used ‘sleep’.

Concurrent Merge Sort - Trying to create a shared memory between each parent and its child causes segfault. Shared memory is released only after system restart

Shortcomings

Editors Reporters Problem - A bottleneck is noticeable in acquiring the lock for the list of locks. However, that is intentional. It is done so to better simulate the situation (Rather than making editors compete for reports, they now take wait for one to select a report)

Dish Washing Problem - For large values of n and m, threads could not be created. Starvation Problem for the scrotch brite

Merge sort - There are cache misses when the first left part is accessed and after that right part is accessed at the time of using multi-process.

Learnings

Editor-Reporters Problem

- Implement the single threaded version and parallelise it later
- Make sure to release any lock that is obtained

Dish Washing Problem

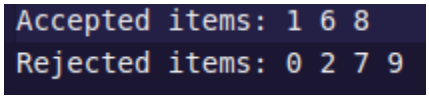
- First run the threads with small input value otherwise it will cause errors and later it will be very difficult to debug.
- Access global variables with the help of locks every time otherwise race conditions may occur and it will be very difficult to find out what is wrong.

Merge sort

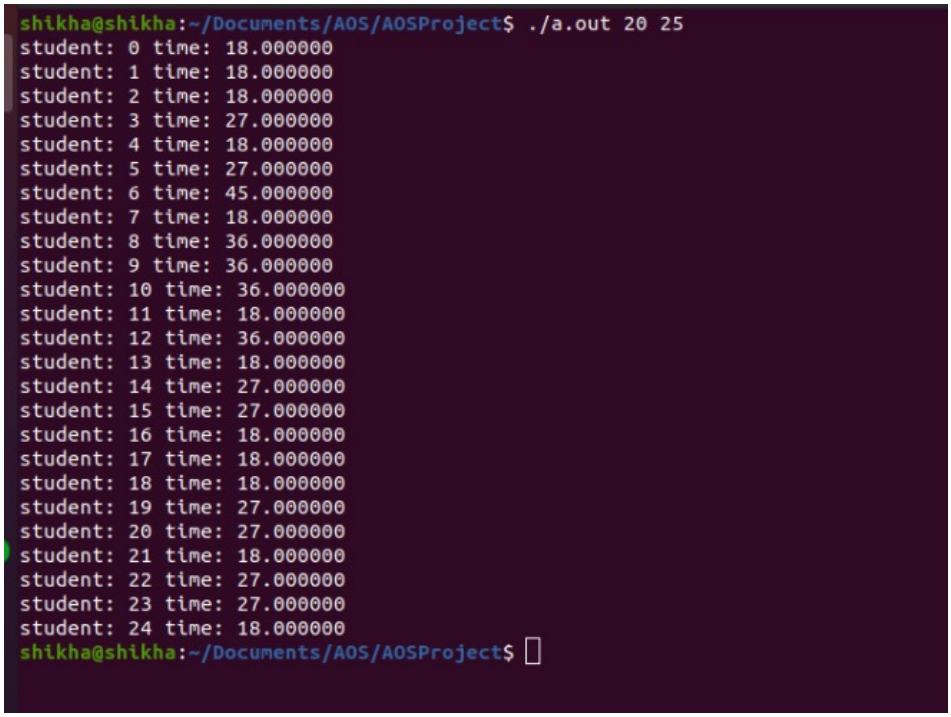
- Copying vector for left and right thread may have a greater overhead than copying arrays
- CFS scheduler groups all threads of a process together and uses their combined execution time for scheduling
- Shared memory for parent and child process in multi process method
- Due to this, multi threaded approach for merge sort performs worse than the other approaches

Result and Conclusion

1. Editor-Reporters Problem - There are n output log files for n number of reporters. Each file has accepted items and rejected items.



2. Dish Washing Problem - Student ID with the completion time respective of each student



Time for large values is significantly large. We tried to get some to virtually simulate things instead of sleep but were not able to perform soon.

3. Concurrent Merge Sort - Time taken for sorting the array of size 100,1000,10000 elements

No of elements	Single process	Multi thread	Multi process
100	0.000112	0.006318	0.000253
1000	0.000803	0.050867	0.000254
10000	0.060979	0.880375	0.000634