NATIONAL UNIVERSITY OF COMPUTER

AND EMERGING SCIENCES

# OPERATING SYSTEMS CS2006

# INSTRUCTOR-MR. Farooq Zaidi

A\* Search Algorithm: Multithreaded Approach

**BY**

**Member 1:** [ *Ayan Bin Rizwan: 21K-3058* ]

**Member 2:** [ *Ismail Qayyum: 21K-3118* ]

**[PROLOGUE]**

## What was the inspiration behind choosing a mapping algorithm?

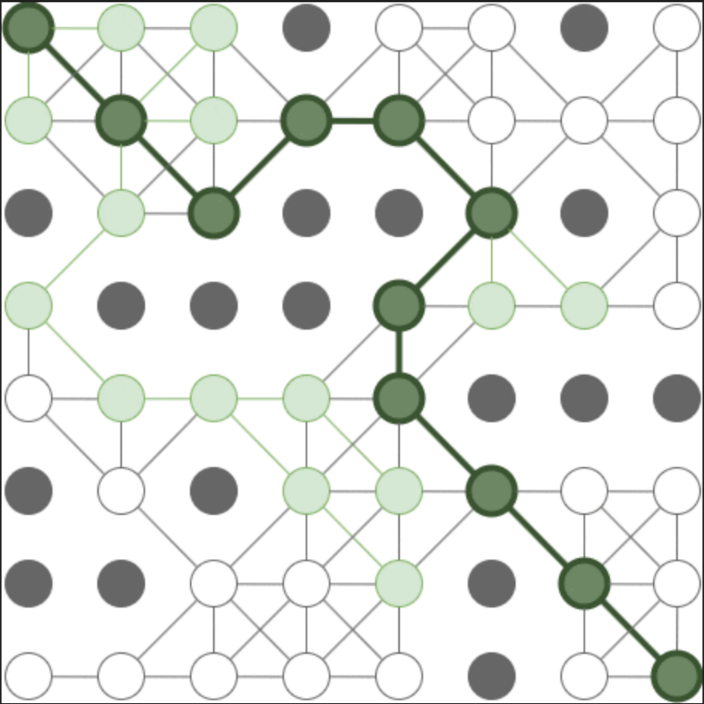
Having studied Data Structures the last semester and learned about time complexity and the issues that arrive with it as well as learning A\* search algorithm in the Artificial Intelligence course this semester , when we learned about the use of Parallel Processing and how it can be used to complete tasks faster in OS, we decided to put the 2 together and create a pathfinding algorithm that would work on large graphs while keeping the time complexity of the solution relatively low.

## How did this project use Operating System concepts learnt in the course?

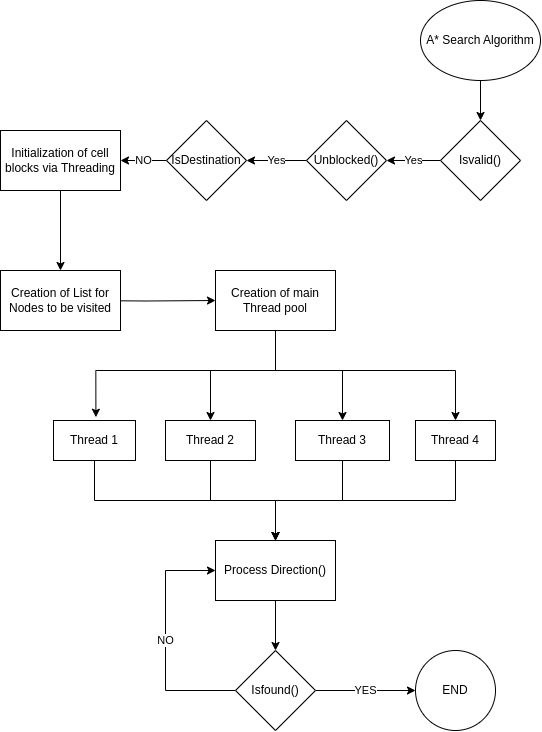
The Algorithm makes use of 2 primary concepts we have learned during, those being that of Threading and Synchronization. The code aims to use the concept of Multithreading in order to split the distance calculation of each node in order to make it so that rather than waiting for a serial number of calculations to be done, these calculations can take place simultaneously. The concept of Mutexs is also used to Synchronize the procedure, since with the code we have tends to ignore the smallest path if it finds that 2 paths to a destination node exist.

## What is the A\* Search Algorithm?

The A\* search Algorithm is a method of searching for the most optimal route in a given graph. Take a look at Dijkstra’s Algorithm, wherein the cost of reaching a node is considered when trying to find out the shortest possible route. Here that cost is added to a Heuristic. A heuristic is an estimation by means of a formula for how far the node currently being processed is from the destination node of the given graph. As such the algorithm can then use this combined calculation of cost and heuristic to guide itself down the most optimal nodes for consideration



# Flowchart showing Functionality of the Algorithm.



**Where has OS been implemented into the Project?**

1. The concept of threading has been implemented to do the task of cell initialization and the task of path finding. The cells are initialized as the process is started via normal threads. A thread pool of 4 threads is then used to make the calculations for the shortest path in all 8 directions around whatever cell involves the function. Calculations for all 8 functions are queued onto the pool and as they get complete previous tasks, they pick up new ones as intended. Regarding thread pooling, since it was a concept learned outside of the course, our attempt to implement it did not go as easily as we imagine it to. While we did attempt to create a thread pooling function, its inconsistency lead to us simply using a thread pooling library for our uses.
2. Additionally, mutexes have also been implemented to prevent a single thread that has reached the destination to declare it has arrived and finish all other threads. This is done since it is not necessary that the direction which computes the fastest is also the shortest. As such only 1 thread is allowed to access the destination at a time so that the reaching of the destination block and its subsequent returning is not interrupted.

## What is the Future prospect of the project?

The implementation of A\* we presented today is one of many others available online. The multithreaded approach may be modified and used to develop new algorithms such as ones that are able to be integrated into OS for the sake of process scheduling or a simpler task of file finding. The algorithm, if further refined can also be used in the urban planning, transport and video game fields wherein it can serve as a optimal route finder whenever needed.

## [EPILOGUE]

1. **What is the ultimate project about?**

The project aims to create a searching algorithm that can produce efficient results when challenged with larger map grid sizes. The utilization of Heuristics to speed up the basic technique of Dijkstra and then the further use of threading to make the code run parallel further helps achieve this goal. This results in the creation of a Algorithm that optimizes itself as it goes on, being able to outdo other algorithms like BFS and Dijkstra especially on Larger Grid Sizes.

## What Problems have been encountered?

During the process of implementing the idea, we encountered multiple problems that hindered the project’s progress at various instances. Following are the problems experienced and their solutions met:

### Overhead and its effect on the time complexity.

Having done threading in our theory and lab courses of Operating Systems, it presented itself as a easy solution to efficiency of codes. Being able to reduce execution time by splitting up tasks to complete them faster, its benefits made us forget about an important issue that comes along with it. Overhead, i.e., the additional resources that a task will require once beyond its minimum necessities. In this case, the creation and deletion of threads as the algorithm moves forward and adjusts itself resulted in a major issue with the first few times we ran the project, our threaded version giving us an exponentially higher execution time that the unthreaded version.

**Solution:** Pooling of threads. By pooling threads, we are able to eliminate the major overhead created as a result of the creation and destruction of multiple threads.

### Integrating the Search Algorithm to a Module in OS.

When we initially picked up the project, we did a quick chat gpt search to see if the algorithm could be used in any module of an OS. To our delight we got the answer saying it could be used in Virtual Memory Management with page replacement and in Process Scheduling to optimize the order of loading/execution in each case. It was only after we started the project and did our full research that we came to the realization that the implementations we earlier saw were not possible, at least at the level we could create. The main issue we came across was that was starting and ending points. Our algorithm requires a definite start and end point in order to work. When it comes to processes or pages, that definitive start and end point doesn’t exist. In other words, the scheduling done would be greatly affected by whatever process we would set as the destination because the algorithm would prioritize reaching it rather than any other process.

**Solution:** No Solution was found for the issue. While we consulted multiple codes and studies available online, we were quickly humbled and realized that the code before us was still a bit beyond our reach to the point where it would be obvious if we lifted it. As such we left creating a module and worked on the main algorithm and optimizing it to give better results than its competitors.

## Additional Concepts Learned as a result of the Project.

As a result of the overhead encountered earlier, we were forced to look into methods that would drastically decrease the overhead of the constant creation and deletion of threads. As such we learned the concept of **Thread Pooling**, i.e., the management of a fixed group of worker threads that perform a certain task. The size of this group is fixed on its creation. The threads become reusable, taking any available task given to it as they become free. In this way, instead of multiple threads being created and destroyed over and over again, a single group of threads is created once and continues to do tasks as they become free. At the end of their service, the group is deleted altogether, significantly reducing the overhead.