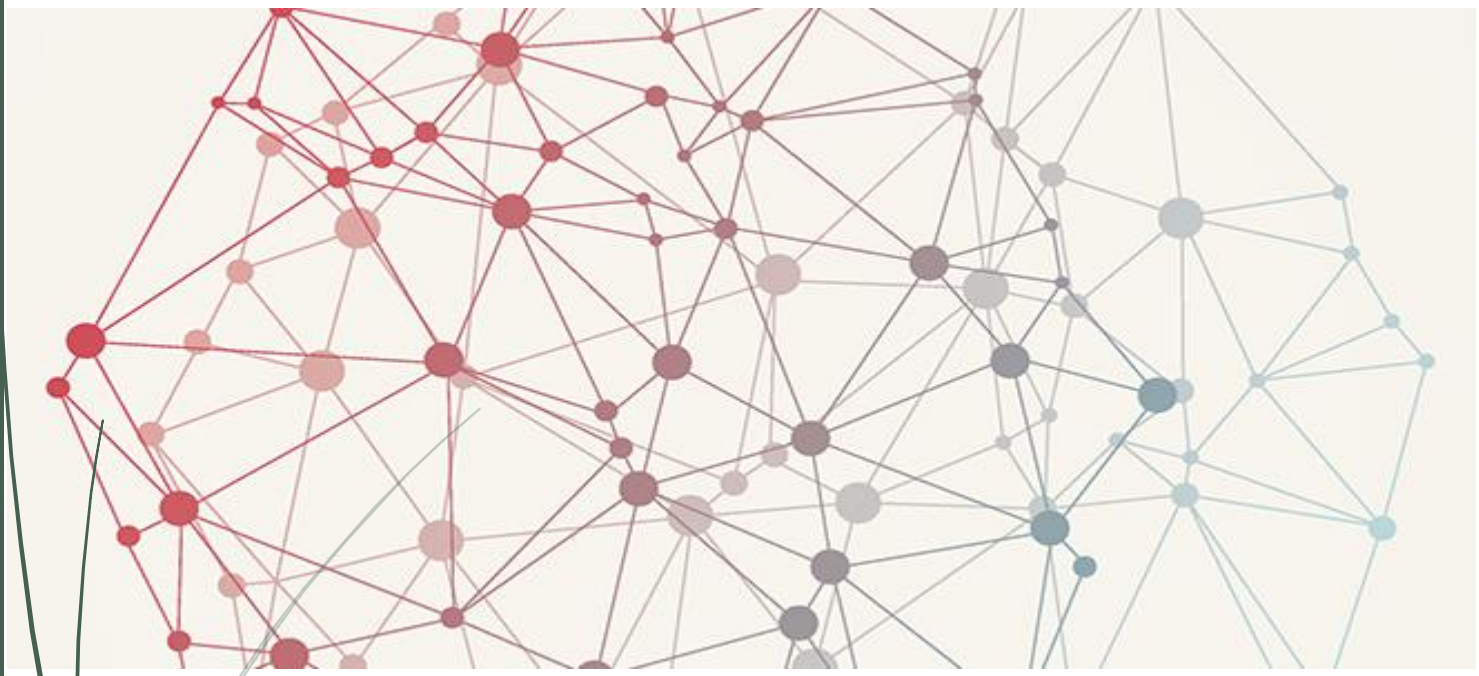
17-3-2017

Least Slack Algorithm

OSM-S HAN Nijmegen



Luke Nooteboom (568157) & Lieven Plasmans (562227)
HOGESCHOOL VAN ARNHEM EN NIJMEGEN 2017

1. Fair vs. Priority scheduling.

The most fundamental difference between fair and priority scheduling is actually quite accurately described by their corresponding names. A fair scheduling algorithm gives every participating member an equal chance to ‘win’ the ‘lottery’. Every participant has an equal chance to be the result of the algorithm. However, in a priority scheduling algorithm, certain members are prioritized over others. For example: there could be several participants in the algorithm with different assigned time values. In the algorithm it is possible that participants with higher time values could be prioritized, so they have a higher chance to be chosen by the algorithm than those with lower time values.

2. Scheduling policy descriptions

Fair Scheduling

Round-robin scheduling

Round-robin scheduling employs time-sharing. This means that each job is given a certain amount of CPU time. If the job isn’t completed when the CPU time runs out, the job is preempted. The algorithm thus in fact iterates between all jobs, assigning them all a fixed amount of CPU time. So when a job is preempted, it is pushed to the back of the queue and will be continued later. When a process is terminated, the algorithm will start handling the next task in the queue and allocate a full portion of CPU time to the next job. This sequence repeats until the job is finished and no more CPU time is required. The advantage of this algorithm is that it is very simple. However the main disadvantage is that it isn’t priority based, this makes it unsuitable for real-time jobs. This scheduling algorithm is used in simpler operating systems and AmigaOS.

Work-conserving scheduling

A work conserving scheduler works similarly to the round-robin scheduler. The main difference in this scheduler is that this scheduler automatically tries to keep scheduled resources busy. If there is a submitted job which is ready to be scheduled, the work-conserving scheduler will try to schedule those jobs, even while the job isn’t set to be scheduled yet. This makes sure all the work is always done.

Priority scheduling

Earliest deadline first scheduling

This is a way of scheduling most students are probably very familiar with. In this kind of scheduling the queue of jobs is searched for the process closest to its deadline. This is the process that will be executed first. The main advantage of this is that it is very suitable for real-time jobs, however it is vulnerable to process starvation if preemption isn’t implemented.

Shortest job first scheduling

In shortest job first scheduling, the queue of jobs is searched for the job that takes the least CPU time to schedule. The job which shows to need the least time will then be scheduled. When the scheduling is finished, the queue is searched again to find the next shortest job. This sequence is repeated until all of the jobs are finished.

3. Systematic comparison

All of the described scheduling methods obviously have the same end goal; all jobs should be scheduled and the queue should be empty when all the jobs are finished. The way in which this is approached however, is fairly different from one algorithm to another. The way in which the jobs are scheduled might not differ all that much, the way in which is determined which job to schedule first however, is. In the fair scheduling methods, this is still fairly equal. All jobs have the same chance to be scheduled first.

This, however is a really short comparison between the aforementioned scheduling methods. When we get deeper into the theory, there are a few keywords we need to clarify before we can conclude how the algorithms chosen to be examined in this article rank on these points; stability, optimality, responsiveness and robustness.

Stability

To quote: *"A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted. Some sorting algorithms are stable by nature (…), other sorting algorithms are not (…)."* (Adams, 2009)

This obviously needs to be clarified. When we would take a list with multiple inserts, for example:

- Bicycle
- Pineapple
- Wizard
- Potato
- Thread

Using a stable sorting algorithm to sort this list only on the first letter, this list would result in:

- Bicycle
- Pineapple
- Potato
- Thread
- Wizard

We can only say this with certainty while using a stable sorting algorithm. When using an unstable sorting algorithm, the words "pineapple" and "potato" might well be swapped around. This is a great example to describe what is meant by "keeping items with the same sorting key in order."

Now this does not explain the subject of stability in its entirety. When using an unstable sorting algorithm on the aforementioned list of words, the algorithm may well result in the same list as a stable algorithm. This would be different when we would ask both algorithms to first sort a list of first and second names on first name and then to sort all names on the last names. With the stable sorting algorithm, we would end up with a list first sorted by last name, but the first names would also be in order, with the unstable sorting algorithm however, the list would be neatly sorted by last name, but all the first names would be all swapped around.

We can thus conclude that most fair scheduling/sorting algorithms, including the ones that were examined in this article, are stable algorithms. The priority scheduling/sorting algorithms are not.

Optimality

An algorithm is considered asymptotically optimal when it is the fastest known algorithm to solve a certain problem. (Brodnik, Carlsson, Demaine, Munro, & Sedgewick, 1999)

This is a very difficult subject. First of all, it is extremely hard to prove a certain algorithm is the most efficient way to solve a certain problem, it is nearly impossible to try all the available algorithms the whole wide world has to offer. Secondly, optimal is a relative concept. Whatever may be the optimal solution for the one case, may actually be the worst possible solution for the other.

These reflections led to the conclusion that it is really hard to point towards the optimal algorithm for solving a problem. All of the discussed algorithms are quite possibly the optimal solution for a certain use case scenario.

Responsiveness

Responsiveness is the word that describes the amount of latency or delay in an algorithm. An algorithm is considered responsive when the latency of the entire loop is as low as possible. This topic again is very much related to the problem the algorithm is confronted with. Some algorithms may work more responsive on certain problems than on others.

In perspective to the scheduling problem this whole article turns around, the priority scheduling algorithms will probably work in a more responsive way than the fair scheduling systems. In the priority scheduling algorithms, the CPU won't have to change around between tasks while still busy scheduling the current task because CPU time has run out.

Robustness

Robustness in computer science is regarded as the ability of a system to handle errors while executing a task. (Dl.ifip.org, 2016)

Robustness relates to many different aspects of computer science. A few examples are robust programming, robust machine learning and a robust security network.

There are techniques (e.g. fuzz testing) which are vital to prove the robustness of a system.

Again, in this category it is extremely hard to rank the examined algorithms. There are many forms and shapes in which errors can take place. It is therefore practically impossible to point at the most or least robust algorithm. Every algorithm is created with (a) certain use case scenario(s) in mind, they are thus created as robust as possible for the pre-calculated risks corresponding to these specific scenarios.

4. Unit testing

Assignment example
2 3
0 30 1 30 2 10
0 60 1 15 2 10

2 3
0 12 1 27 2 31
0 35 1 12 2 15

	Task 0	Task 1	Task 2	End Time
Job 0	0, 30	1, 30	2, 10	
E.S.	0	30	60	70
L.S.	15	45	75	85
Job 1	0, 60	1, 15	2, 10	
E.S.	0	60	75	85
L.S.	0	60	75	85
With Resource Management				
Job 0	60	90	120	130
Job 1	0	60	75	85
	Task 0	Task 1	Task 2	End Time
Job 0	0, 12	1, 27	2, 31	
E.S.	0	12	39	70
L.S.	0	12	39	70
Job 1	0, 35	1, 12	2, 15	
E.S.	0	35	47	62
L.S.	8	43	55	70
With Resource Management				
Job 0	0	12	39	70

Job 1	12	47	70	85

As seen above, these two test configurations were worked out by hand. When compared to the results our algorithm returned, they were exactly the same.
This means our algorithm does work in the way it was intended to.

The manual calculations were carried out while keeping the explanation in the documentation to the side, the exact same steps were carried out in the right sequence.

This indicates the algorithm was successful. However there is still a possibility for errors.

References

Adams, J. (2009, 10 05). *www.stackoverflow.com*. Retrieved from www.stackoverflow.com: http://stackoverflow.com/questions/1517793/stability-in-sorting-algorithms

Brodnik, A., Carlsson, S., Demaine, E. D., Munro, I. J., & Sedgewick, R. (1999, 09 01). *Cs.uwaterloo.ca.* Retrieved from Cs.uwaterloo.ca: https://cs.uwaterloo.ca/research/tr/1999/09/CS-99-09.pdf

Dl.ifip.org. (2016, 11 13). *http://dl.ifip.org/db/conf/pts/testcom2005/FernandezMP05.pdf.* Retrieved from http://dl.ifip.org/db/conf/pts/testcom2005/FernandezMP05.pdf: http://dl.ifip.org/db/conf/pts/testcom2005/FernandezMP05.pdf