Felicia Kuan

COEN 177

Final Lab Report

For the final lab, I intend to test how well the Banker’s Algorithm performs in resource allocation against an algorithm that allocates resources to random requests. In a situation involving resource allocation where the said resources are limited, deadlock is caused when the algorithm haphazardly hands out the limited resources to requesting processes until the resources have run out, and the involved processes, in order to finish and release the resources they acquired, wait forever for the others to give up the resources they acquired. Deadlocks in these situations are bad because the processes cannot continue to be run and must be killed, so an algorithm that attempts to avoid deadlock, the Banker’s Algorithm, is used to allow processes to complete.

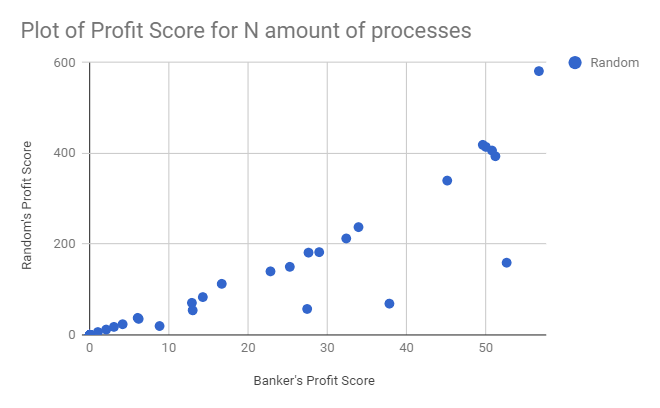
However, the Banker’s Algorithm takes much longer to run than an algorithm that haphazardly hands out the resources-- with the Banker’s Algorithm running at O(n^2) time. Thus, in this experiment, I intend to test whether the benefits of avoiding deadlock using the Banker’s Algorithm outweigh the costs of how long it takes for the algorithm to run. The performance of the two algorithms will be measured by a quantity I call Profit Score: (number of processes successfully allocated resources/ time it takes for the program to run) \* 100 so that the value wouldn’t have too many decimal places. Profit Score is an arbitrary metric made up by me that is supposed to represent the rate of processes that were successfully allocated resources per unit of time (microseconds). Therefore, mathematically, the higher the profit score, the faster processes are allocated resources. I hypothesize that the Profit Score of the random allocation of resources will be the square of the Banker’s Algorithm, because the Banker’s Algorithm runs at O(n^2).

To make things easier for testing, I wrote my Banker’s Algorithm to only allocate one type of resource. Therefore, each array would be requesting a random number n of a resource, and would currently be holding (n - another random number) amount of the resource. The algorithm goes through and searches for the process with the minimum difference of resources between what the process requests and what it currently has and hands the resources over. Then, the algorithm checks if the amount of resources it is currently holding dipped below 0. If it did, then the algorithm failed and allocated too much to the processes. Deadlock occurs. If the resources is currently greater than 0, it takes back all of the resources of the process the program just allocated resources to and takes that process out of the list of needy processes. Once satisfied, the processes cannot ask for more resources, and once all processes are allocated resources, the program has successfully finished and we are returned a time (in microseconds) it took to run the Banker’s Algorithm.

For the random allocation algorithm, it simply goes through sequentially through all the processes that require resources and allocate to them. This is random because both the starting available resources the algorithm starts with and the amount each process needs is randomly chosen for each time I run the program. There is no need to add another layer of randomization for the selection process of which process to allocate to first. After all, theoretically, a program could start with 1 available resource and the first process’ difference between what it has and what it needs is 2. Banker’s Algorithm could trump this if the difference of second process between what it has and what it wants is 1, because Banker would allocate to the second process first, free up the resources, and allocate to the first process.

To test this, I ran the two algorithms against each other 100 times, starting with 1 process to allocate resources to, all the way to 100 different processes. Each time, the starting amount of available resources to allocate is randomly chosen, but the range is dependent upon the number of different processes need to be allocated resources. The range of available resources scales linearly for every 20 process bracket. For example, for one process that needs to be allocated resources, the range for available resources is from 1 - 100. This range is for all processes from 1 -20. For 21 - 40 processes that need to be allocated resources, the range of starting available resources is 1-200. For 81-100 processes that need resources, the range available is from 1 - 500.

With the given amount of processes and randomly selected amount of resources to work with, I ran the algorithms to calculate the Profit Score and plotted the results on a scatter plot. (See Fig 1: Scatterplot below) And, the results were messy. When there was a deadlock and not all resources could be allocated, the Profit Score for an algorithm would be 0. There were many 0’s, especially when the number of processes were larger. From 72 - 93 processes, both random and Banker’s method of allocation of resources led to deadlock and produced a Profit Score of 0. This might be caused by the way I scaled the amount of available resources, but since the resources were chosen randomly, it is possible that in all of these instances, the initial amount of available resources was too little for the number of processes that request resources. ( See “Chart Displaying Profit Score for Number of Processes 1 - 100” for more details)



Observing the Profit Scores of the Banker’s Algorithm versus the Random allocation, it would seem that the Profit Scores of the Random Allocation is on average, actually less than the square of the Profit Scores of the Banker’s Algorithm. However, the Profit Score for random allocation is often at least 3, even 5 times the Profit Score of the Banker’s Algorithm. This difference between the Profit Scores of the Banker’s Algorithm and my random allocation algorithm was smaller than I thought. However, it is clear that the Banker’s Algorithm is much more costly, and these costs will scale even further when there are more processes involved.

Additionally, there are no instances where the Banker’s Algorithm avoided deadlock and Random Allocation did not. Perhaps, due to randomization and chance, I simply did not get this instance of the data. However, with the available information it seems that the Banker’s Algorithm can take up to 5 times longer to run than with no added benefits of avoiding deadlock. This means that with this Profit Score metric, that is focused more on a short execution time, Random Allocation would trump Banker’s Algorithm. The only way Banker’s Algorithm could have defended itself and demonstrated itself as a useful algorithm was to show an instance of where it managed to avoid deadlock while the Random Allocation didn’t. This would show that the costs of the Banker’s Algorithm taking much longer to run would be rewarded by avoiding deadlock. However, this did not occur, and the Random Allocation algorithm performed better than the Banker’s Algorithm, with this metric.

Chart Displaying Profit Score for Number of Processes 1 - 100

|  |  |  |
| --- | --- | --- |
|  | Banker's | Random |
| 1 | 1.03 | 6.25 |
| 2 | 2.08 | 11.76 |
| 3 | 3.06 | 17.64 |
| 4 | 4.16 | 23.52 |
| 5 | 6.18 | 35.29 |
| 6 | 6.06 | 37.5 |
| 7 | 0 | 0 |
| 8 | 0 | 0 |
| 9 | 8.82 | 19.56 |
| 10 | 0 | 0 |
| 11 | 0 | 0 |
| 12 | 12.9 | 70.58 |
| 13 | 13 | 54.16 |
| 14 | 0 | 0 |
| 15 | 14.28 | 83.3 |
| 16 | 0 | 0 |
| 17 | 0 | 0 |
| 18 | 16.67 | 112.5 |
| 19 | 0 | 0 |
| 20 | 0 | 0 |
| 21 | 22.82 | 140 |
| 22 | 0 | 0 |
| 23 | 0 | 0 |
| 24 | 25.26 | 150 |
| 25 | 0 | 0 |
| 26 | 0 | 0 |
| 27 | 0 | 0 |
| 28 | 27.45 | 57.14 |
| 29 | 27.62 | 181.25 |
| 30 | 0 | 0 |
| 31 | 28.97 | 182.35 |
| 32 | 0 | 0 |
| 33 | 0 | 0 |
| 34 | 32.381 | 212.5 |
| 35 | 0 | 0 |
| 36 | 0 | 0 |
| 37 | 0 | 0 |
| 38 | 33.93 | 237.5 |
| 39 | 0 | 0 |
| 40 | 0 | 0 |
| 41 | 0 | 0 |
| 42 | 37.83 | 68.85 |
| 43 | 0 | 0 |
| 44 | 0 | 0 |
| 45 | 0 | 0 |
| 46 | 0 | 0 |
| 47 | 0 | 0 |
| 48 | 0 | 0 |
| 49 | 0 | 0 |
| 50 | 0 | 0 |
| 51 | 45.13 | 340 |
| 52 | 0 | 0 |
| 53 | 0 | 0 |
| 54 | 0 | 0 |
| 55 | 0 | 0 |
| 56 | 0 | 0 |
| 57 | 0 | 0 |
| 58 | 50 | 414.28 |
| 59 | 0 | 0 |
| 60 | 0 | 0 |
| 61 | 0 | 0 |
| 62 | 51.21 | 393.75 |
| 63 | 0 | 0 |
| 64 | 0 | 0 |
| 65 | 0 | 0 |
| 66 | 50.78 | 406.25 |
| 67 | 0 | 0 |
| 68 | 49.62 | 418.75 |
| 69 | 0 | 0 |
| 70 | 52.63 | 159.09 |
| 71 | 0 | 0 |
| 72 | 0 | 0 |
| 73 | 0 | 0 |
| 74 | 0 | 0 |
| 75 | 0 | 0 |
| 76 | 0 | 0 |
| 77 | 0 | 0 |
| 78 | 0 | 0 |
| 79 | 0 | 0 |
| 80 | 0 | 0 |
| 81 | 0 | 0 |
| 82 | 0 | 0 |
| 83 | 0 | 0 |
| 84 | 0 | 0 |
| 85 | 0 | 0 |
| 86 | 0 | 0 |
| 87 | 0 | 0 |
| 88 | 0 | 0 |
| 89 | 0 | 0 |
| 90 | 0 | 0 |
| 91 | 0 | 0 |
| 92 | 0 | 0 |
| 93 | 56.7 | 581.25 |
| 94 | 0 | 0 |
| 95 | 0 | 0 |
| 96 | 0 | 0 |
| 97 | 0 | 0 |
| 98 | 0 | 0 |
| 99 | 0 | 0 |
| 100 | 0 | 0 |