**Program 3 Banker’s Algorithm**

**Problem Statement**

The assigned task is to write a C/C++ program which implements the banker's algorithm. The program needs to perform the banker’s algorithm on data sets to determine if a system is in a safe or unsafe state. The importance of the project is to learn about resource allocation, deadlock avoidance, and how the banker’s algorithm determines whether a set of processes can encounter a deadlock or not.

**Approach**

The approach I took for this program was to implement the Banker’s algorithm in C++ using two dimensional vectors for the claim and allocation matrices. The first step to implementing the Banker’s algorithm was to input the system data. The system data is received by the program through a formatted text file. The format of the text file is as follows:

* Line 1 contains the number of resources
* Line 2 contains the quantity for each resource
* Line 3 contains the number of processes
* Lines 4 through 3+n contain the Claim matrix, each line contains the maximum resource requirements for one process
* Lines 4+n and beyond contain the Allocation matrix, each line contains the current number of resources allocated to the process
* The Claim and Allocation matrices are in the same process order
* Sequenced numbers on the same line are separated by a single tab

I stored the number of resources on line 1 and the number of processes on line 3 as integer values because theses inputs are just a single value. The number of resources translated to the number of columns in the claim and allocation matrices, the number of processes translated to the number of rows in the claim and allocation matrices. Theses numbers were useful for traversing the matrices.

I stored the quantity for each resource as an array with a size specified by the first line of the text file. To create the array, I used my splitString() function to tokenize the integer values present on the text line inputted from the file.

To create the claim and allocation matrices I made a function that returns a two-dimensional vector of the data from the text file. The reason I chose to implement the matrices as two-dimensional vectors was because the vector data structure does not need a size at declaration. My matrix creation function is called makeProcessVector() and takes the arguments ifstream\*, int, and int. The pointer to the ifstream is where the text file is opened. The two integer values are the number of resources and processes, which are also the dimensions of the matrices. The makeProcessVector() function creates a two-dimensional vector to be filled with the data from the text file. Then while the function is iterating through each column of each row the elements of each row are loaded into a temporary one-dimensional vector. When the loop finishes a row, the temporary vector is pushed onto the two-dimensional vector and the next row begins generating.

The last matrix needed in the computation of the Banker’s algorithm is a need matrix that indicates the remaining resource need of each process. The need matrix is calculated by subtracting the allocation matrix from the claim matrix. This is done through my calcNeed() function that iterates through the claim and allocation matrices and pushes the claims minus the allocation value to the corresponding position of the matrix. It returns a two-dimensional vector which is the need matrix.

Finally, I implemented the banker’s algorithm in my safe() function that is called at the end of main. I pass the number of processes and resources to the function as well the claim, allocation, and need matrices. My function first makes a copy of my resource array to create an array containing the available resources for the system. The available resources for the system are calculated by taking the total available resources, defined in the resources array, minus the current resources allocated from the allocation matrix. Next, I loop through the processes and determine if there are enough available resources for one of the processes to execute. If there are enough resources for a specific process to execute then the process returns its resources to the pool of available resources and the process is added to the safe sequence of execution. If the algorithm iterates through all the processes and every process is granted sufficient resources, then the system is in a safe state and the function prints out the process execution sequence.

**Solution**

The assigned task was to write a C/C++ program which implements the banker's algorithm. To implement a solution, I used C++. I implemented array and vector data structures to store process data.

When running my program, you can either manually change the text file name for the data within the main function or run the executable with the name of a text file as an argument.

Figure 1 shows that I used the g++ compiler on my Ubuntu 20.04 LTS environment to compile my program.



Figure 1

Figure 2 shows the execution of the attached data file. My program determined that the attached file resulted in an unsafe system. Figure 2 also shows the use of an argument when running the executable.

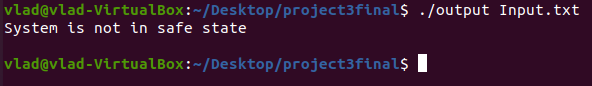


Figure 2

Figure 3 shows the execution of the banker’s algorithm on the data set figure 6.7a from the textbook. The algorithm determines that the data set results in a safe system with the process execution sequence of process 2, process 1, process 3, and finally process 4.

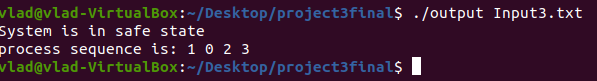


Figure 3

Figure 4 shows the execution of the banker’s algorithm on the data set figure 6.8b from the textbook. The algorithm determines that the data set results in an unsafe system because each process will need at least one additional unit of resource 1, and there are none available.

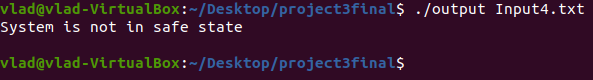


Figure 4

A problem that I encountered while trying to solve the assigned task was that it was challenging to the allocation and claim matrices as array data structures. To solve this problem, I decided to use the vector data structure to implement the needed matrices because vectors do not need a size at definition but vector positions can still be accessed in a similar manner as arrays.