Josh Romisher
Operating Systems
Banker's Algorithm Documentation
11/19/22

Project Implementation

I began this project with a few different ideas on how I was going to attempt to implement the task we were given. My final program used a source file and an input text file, using the fstream library to read input from bankersAlgInput.txt for the necessary data structures. First, I declared a series of different arrays, both one and two dimensional, to hold the data from the table given. These arrays were declared in bankersAlgSource.cpp and then initialized using ifstream. The arrays held data that was important for keeping track of process 'names' (proc[]), current number of resource instances allocated (2d array allocated[]), maximum number of resource instances a process can request (2d array max[]), number of resources currently available to be requested (2d array available[]). These arrays were filled based on values given as parameters for the assignment.

I used print statements after filling the arrays via file I/O to ensure that the correct values were assigned to the correct array indices. After confirming the file I/O filled the arrays correctly, I began working on the resource request and safe state test function, safetyTest(). The function safetyTest() uses the difference in the values of the previously declared arrays max[] and allocated[] in order to initialize the array need[]. Need[] represents the resources still needed by process P_i to complete its execution. Also within safetyTest(), three more arrays are declared and initialized to help keep track of the data.

Work[] is a copy of available[], it holds the current number of available resources, and its values are incremented with the appropriate values as processes complete execution and release their resources. Next is finish[], which keeps track of which process, P_0-P_4, has completed its execution. A value of 0 in each index shows the P_i has not completed its execution and released its resources. Once the process succeeds in passing the conditions of the loops, its index value in finish[] is changed to 1, so that on the following iterations, it will be passed over due to the initial condition if(finish[p] == 0). Last is correctSeq[], an array that holds the correct sequence of process execution to maintain a safe system state.

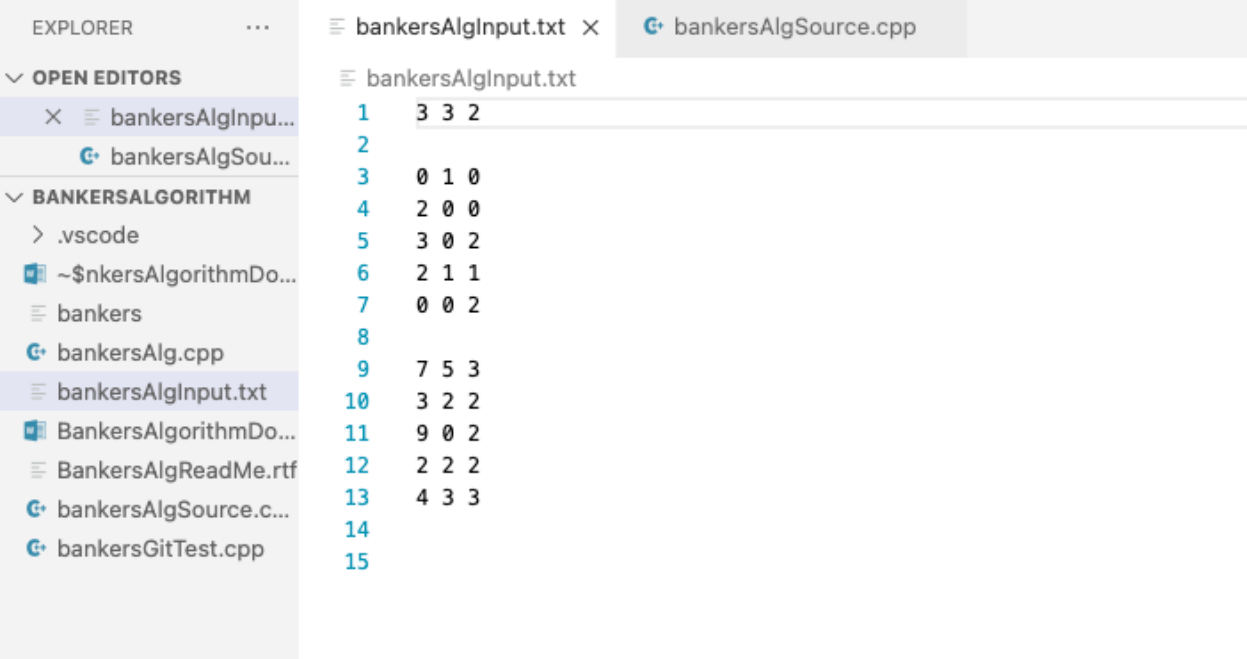
The function safetyCheck() works as follows. A tracking variable safeCount is initialized to 0, signifying that at the beginning of the function, no processes have been declared in a safe state in their current order. A while loop keeps track of iterations using safeCount, and iterates while it remains less than the constant numProc, representing the number of processes. A Boolean variable foundSafe is initialized to false, and will return an error that the system is not in a safe state with the current order of execution unless its value is returned as true.

The function then checks for the following conditions using for loops and if statements. First, it checks if finish[] is 0 for the current index position. Finish must be 0 for that particular index to pass the condition because if finish[p] = 1, then that process has already been added to correctSeq[]. The next condition to check is whether the current processes need exceeds the current amount of available resources in work[]. If its need[] exceeds the number of resources available, the function breaks and continues on to the next iteration.

Once the need[] > work[] condition is passed, this means that the current process in the current iteration's need[] does not exceed the available amount of resources in work[] and it can complete its execution. The process's resources are then released to work[], which is incremented based on its current value plus the values in allocated[] for the current process in that iteration. That process is then added to the correctSeq[] array, its finish[] value is set to 1 for the correct index, and foundSafe is set to true. This process continues until all processes pass the test. The correct order or process execution is then printed out.

Visual Walkthrough

The following pages show a visual representation of the implementation of the banker's algorithm project, along with the final execution of the project. The visual walkthrough starts by showing the text input file, opening the input text file, and adding the values from this file to the correct arrays. The print statements that follow are to ensure that file I/O was done correctly, and the correct values are in the right spot in the appropriate arrays. Next is a visual overview of safetyCheck(), and finally a printout of the correct program execution.



bankersAlgInput.txt used to fill arrays available[], allocated, and max[]

Declaring allocated[], max[], and available[], and opening bankersAlgInput.txt

```
int main(){

    /*Declaring array data structures*/
    int allocated[numProc][numRes];
    int max[numProc][numRes];
    int available[numRes];

    /*Opening File Stream*/
    ifstream arrayIn;
    arrayIn.open("bankersAlgInput.txt");
    if(arrayIn.fail()){
        cout << "Error, file failed to open";
        exit(1);
    }

    for(int i = 0; i < numRes; ++i){
        arrayIn >> available[i];
    }

    for(int i = 0; i < numProc; ++i)
        for(int j = 0; j < numRes; ++j){
            arrayIn >> allocated[i][j];
        }

    for(int i = 0; i < numProc; ++i)
        for(int j = 0; j < numRes; ++j){
            arrayIn >> max[i][j];
        }

    arrayIn.close();
```

Filling arrays available[], allocated[], and max[]

```
    /*Process array for bool safetyTest parameter*/
    int proc[] = {0, 1 ,2 ,3 ,4};

    /*Available Vector Printout Test*/
    for(int i = 0; i < numRes; ++i)
        cout << "Resource " << i << " contains " <<  available[i] << " available instances of the resource" << endl;

   /*Allocated Vector, initializing process allocated vectors (Printout Test)*/
        for(int i = 0; i < numProc; ++i)
            for(int j = 0; j < numRes; ++j){
                cout << "Process P" << i << " is currently allocated " << allocated[i][j] << " instance(s) of resource " << j << endl;
            }

    /*Max instances of reso const int numProc = 5 :(Printout Test)*/
        for(int i = 0; i < numProc; ++i)
            for(int j = 0; j < numRes; ++j){
                cout << "Process P" << i << " may request at most " << max[i][j]<< " instances of resource " << j << endl;
            }
```

Print statements to show that array values assigned by file I/O were correct

```
bool safetyTest(int proc[], int available[], int max[][numRes], int allocated[][numRes]){
    /*Need vector initialization*/

    int need[numProc][numRes];

        for(int i = 0; i < numProc; ++i)
            for(int j = 0; j < numRes; ++j){
                need[i][j] = max[i][j] - allocated[i][j];
            }
    /*Finish vector keeps track of which processes are complete, initialized to false for all indeces*/
    int finish[numProc] = {0};

    /*correctSeq vector holds the Process numbers in order of safe sequence*/
    int correctSeq[numProc] = {0};

    /*Work vector, a copy of available resources to track available resources*/
    int work[numRes];
    for(int i = 0; i < numRes; ++i){
        work[i] = available[i];
        cout << "Work vector contains value " << work[i] << " at index " << i << endl;
    }

```

safetyTest() initializing need[], finish[], correctSeq[] and work[]

```cpp
    /*Keeping track of number of unfinished processes or processes in safe state*/
    int safeCount = 0;

    while(safeCount < numProc){

        bool foundSafe = false;
        for(int p = 0; p < numProc; ++p){
            if(finish[p] == 0){
                int j;
                for(j = 0; j < numRes; ++j)
                    if(need[p][j] > work[j])
                        break;


                if(j == numRes){
                    for(int k = 0; k < numRes; ++k){
                        work[k] += allocated[p][k];
                    }
                    correctSeq[safeCount++] = p;
                    finish[p] = 1;

                    foundSafe = true;
                }
            }
        }

        if(foundSafe == false){
            cout << "Error, system is not currently in a safe state" << endl;
            return false;
        }

    }

    cout << "System is currently in a safe state, and the correct sequence of process execution is: \n";
    for(int i = 0; i < numProc; ++i){
        cout << "P" << correctSeq[i] << "->";
    }
    return true;
}
```

Resource allocation and safe state check algorithm

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                                                                    bash

Joshs-MacBook-Air:bankersAlgorithm j_romisher$ ls
Archived                                Images                                  bankersAlgInput.txt             ~$nkersAlgorithmDocumentation.docx
Documentation                           bankers                                 bankersAlgSource.cpp
Joshs-MacBook-Air:bankersAlgorithm j_romisher$ rm bankers
Joshs-MacBook-Air:bankersAlgorithm j_romisher$ g++ bankersAlgSource.cpp -o bankers
Joshs-MacBook-Air:bankersAlgorithm j_romisher$ ./bankers
Reading array input from file is complete, file closed
Resource 0 contains 3 available instances of the resource
Resource 1 contains 3 available instances of the resource
Resource 2 contains 2 available instances of the resource
Process P0 is currently allocated 0 instance(s) of resource 0
Process P0 is currently allocated 1 instance(s) of resource 1
Process P0 is currently allocated 0 instance(s) of resource 2
Process P1 is currently allocated 2 instance(s) of resource 0
Process P1 is currently allocated 0 instance(s) of resource 1
Process P1 is currently allocated 0 instance(s) of resource 2
Process P2 is currently allocated 3 instance(s) of resource 0
Process P2 is currently allocated 0 instance(s) of resource 1
Process P2 is currently allocated 2 instance(s) of resource 2
Process P3 is currently allocated 2 instance(s) of resource 0
Process P3 is currently allocated 1 instance(s) of resource 1
Process P3 is currently allocated 1 instance(s) of resource 2
Process P4 is currently allocated 0 instance(s) of resource 0
Process P4 is currently allocated 0 instance(s) of resource 1
Process P4 is currently allocated 2 instance(s) of resource 2
Process P0 may request at most 7 instances of resource 0
Process P0 may request at most 5 instances of resource 1
Process P0 may request at most 3 instances of resource 2
Process P1 may request at most 3 instances of resource 0
Process P1 may request at most 2 instances of resource 1
Process P1 may request at most 2 instances of resource 2
Process P2 may request at most 9 instances of resource 0
Process P2 may request at most 0 instances of resource 1
Process P2 may request at most 2 instances of resource 2
Process P3 may request at most 2 instances of resource 0
Process P3 may request at most 2 instances of resource 1
Process P3 may request at most 2 instances of resource 2
Process P4 may request at most 4 instances of resource 0
Process P4 may request at most 3 instances of resource 1
Process P4 may request at most 3 instances of resource 2
Work vector contains value 3 at index 0
Work vector contains value 3 at index 1
Work vector contains value 2 at index 2
System is currently in a safe state, and the correct sequence of process execution is:
P1->P3->P4->P0->P2->Joshs-MacBook-Air:bankersAlgorithm j_romisher$
```

bankersAlgSource.cpp execution