COMP301

Operating Systems

Section: C

Assignment#: 2

Title: Implementation of the Banker's Algorithm embedded with safety Algorithm

Dated: November 6, 2014

Submitted To: Prof. Muhammad Rauf Butt

Submitted by: Waleed Ahmed

Roll#: 16-10876

Abstract:

The purpose of this code is to simulate the Banker's algorithm which is very famous solution to the contemporary problem of deadlocks. This algorithm is implemented in UNIX based operating systems. This algorithm as implemented in this code contains a different algorithm that is called the safety algorithm whose job is to determine the possibility of the dead locks. It follows a simple process that involves the comparing of the available resources at the instant with the needed resources at that instant and determine the safety status. Following are the steps involved in the Banker's algorithm:

1. Calculate the need matrix at the present given state.
2. Determine the safety status.
3. Start allocating the resources depending upon the safety status.
4. Call safety algorithm to determine the present safety status.
5. If at any instant available are less than needed then report deadlock.

Introduction:

The code on the following page is an implementation of the Banker's Algorithm (embedded with safety algorithm). This was given as a deadlock solution for the resource sharing processes by a Dutch Computer Scientist named Edseger Dijkstra. It takes three matrices from the user to define a present system state as inputs. Then it calculates a need matrix to feed it to the safety algorithm along with the newly requested resource instances to check if granted, whether or not there will be a dead lock. It runs an imaginary sort of parallel simulation to determine whether the requested resources will exceed the available if all the safe processes have been completed whether or not there will be a dead lock. It starts with the safer processes along with recollection of the resources which were pre-granted and adding them back to the available matrix. In this way if always recalls the safety algorithm to check the safe status as per the updated availability. If all the processes are successfully granted the resources and taken back after they complete then we will have no deadlock under the present circumstances. So as a result we proceed.

Code:

```
import java.util.*;

public class Bankers {

	int proc;
	int res;
	int[][] maximum;
	int[][] need;
	int[][] allocated;
	int[][] available;
	int[][] request;
	boolean[] safetyLog;

	void display_Present_State() {
		System.out.println("-----PRESENT_SYSTEM_STATE-----");

		System.out.println("Processes    |Maximum     |Allocated");

		for (int i = 0; i < proc; i++) {
			System.out.print("P" + i + "           ");
			for (int j = 0; j < res; j++) {
				System.out.print(maximum[i][j] + "  ");
			}
			System.out.print("  ");
			for (int j = 0; j < res; j++) {
				System.out.print(+allocated[i][j] + "  ");
			}
			System.out.println();
		}

		System.out.println("\nAvailable Resources in the OS");

		for (int i = 0; i < res; i++) {
			System.out.print(available[0][i] + " ");
		}

		System.out.println();

		System.out.println("Need Matrix");

		for (int i = 0; i < proc; i++) {
			System.out.print("P" + i + "  ");
			for (int j = 0; j < res; j++) {
				System.out.print(need[i][j] + " ");
			}
			System.out.println();
		}
	}

	public void isSafe() {
		// this function determines the processes for which it is unsafe to
		// start with
		safetyLog = new boolean[proc];
```

```
		for (int i = 0; i < proc; i++) {
			safetyLog[i] = true;
		}

		for (int i = 0; i < proc; i++) {
			for (int j = 0; j < res; j++) {
				if (need[i][j] > available[0][j]) {
					safetyLog[i] = false;
					break;
				}
			}
		}

		/*System.out
				.println("Following processes are safe and unsafe to start
with");

		for (int i = 0; i < proc; i++) {
			if (safetyLog[i]) {
				System.out.println("P" + i + ": Safe");
			} else {
				System.out.println("P" + i + ": Un-Safe");
			}
		}*/

	}

	void safetyAlgo(int[][] request, int n) {

		boolean[] done = new boolean[proc];

		for (int i = 0; i < res; i++) {
			need[n][i] += request[0][i];
		}
		//changing the safety status after updating the need matrix
		isSafe();
		for (int i = 0; i < proc; i++) {
			done[i] = false;
		}

		int count = 0;
		int count_safe_alloc = 0;

		System.out.println("Processes will start in the following sequence:");

		while (count < proc) {

			for (int i = 0; i < proc; i++) {
				if (safetyLog[i] && !done[i]) { // if safe and has not
been done
					count_safe_alloc++;
					for (int j = 0; j < res; j++) {
						available[0][j] = available[0][j] +
allocated[i][j];
```

```
                                        //System.out.print("P" + i + ", ");
                                        done[i] = true;

                                        //**********this part is just to update the
safe or unsafe status

                                        isSafe();

                                        //************

                                        // must change the safety status here

                                        // safetyLog[i]=false;


                                  }

                                  System.out.print("P" + i + ", ");
                            }
                      }
                      count++;

               }

               System.out.println("\n");

               if (count_safe_alloc < proc) {

                      System.out
                                  .println("\nThere will be a dead lock. So Can't
allocate");
               } else {
                      System.out.println("There will be No dead lock.");
               }

               System.out.println("Final Available Matrix:");

               for(int i=0;i<res;i++){
                      System.out.print(available[0][i]+" ");
               }

        }

        void input() {
               Scanner scan = new Scanner(System.in);

               System.out.print("Enter the no of Resources: ");
               res = scan.nextInt();
               System.out.print("Enter the no of Processes: ");
               proc = scan.nextInt();

               // allocated_Resources
               System.out.println("Enter the allocated resources:");
               allocated = new int[proc][res];
```

```
for (int i = 0; i < proc; i++) {
	System.out.print("Process-" + i + ": ");
	for (int j = 0; j < res; j++) {
		allocated[i][j] = scan.nextInt();
	}
	System.out.println();
}

// maximum resources required
System.out.println("Enter the maximum required resources: ");
maximum = new int[proc][res];

for (int i = 0; i < proc; i++) {
	System.out.print("Process-" + i + ": ");
	for (int j = 0; j < res; j++) {
		maximum[i][j] = scan.nextInt();
	}
	System.out.println();
}

// available resources
available = new int[1][res];
System.out.println("Enter the resources present at the instance:");
for (int i = 0; i < res; i++) {
	available[0][i] = scan.nextInt();
}

// need matrix calculation::

need = new int[proc][res];

for (int i = 0; i < proc; i++) {
	for (int j = 0; j < res; j++) {
		need[i][j] = maximum[i][j] - allocated[i][j];
	}
}

display_Present_State();
isSafe();

request = new int[1][res];

System.out.print("Enter the process and request the resources: \n");
System.out.print("Process: ");
int n = scan.nextInt();
System.out.print("Enter the resources: ");
for (int i = 0; i < res; i++) {
	request[0][i] = scan.nextInt();
}

int[][] temp = new int[1][res];

for (int i = 0; i < res; i++) {
	temp[0][i] = available[0][i];
}
```

```
            safetyAlgo(request, n); // passing on the requested resources and the
                                                    // process no
        }

        public static void main(String[] args) {

            new Bankers().input();
            // input();

        }

}
```

Output Samples:

Sample-1

```
-----PRESENT_SYSTEM_STATE-----
Processes       |Maximum        |Allocated
P0              7  5  3         0  1  0
P1              3  2  2         2  0  0
P2              9  0  2         3  0  2
P3              2  2  2         2  1  1
P4              4  3  3         0  0  2

Available Resources in the OS
3 3 2
Need Matrix
P0  7 4 3
P1  1 2 2
P2  6 0 0
P3  0 1 1
P4  4 3 1
Enter the process and request the resources:
Process: 1
Enter the resources: 1 2 2
Processes will start in the following sequence:
P3, P4, P1, P2, P0,
 5
There will be No dead lock.
Final Available Matrix:
10 5 7
```

Sample-2

```
-----PRESENT_SYSTEM_STATE-----
Processes       |Maximum        |Allocated
P0              4  7  3         1  1  0
P1              6  5  4         2  2  0
P2              4  0  3         3  0  2
P3              2  1  2         2  1  1
P4              3  6  3         0  2  2

Available Resources in the OS
3 4 4
Need Matrix
P0  3 6 3
P1  4 3 4
P2  1 0 1
P3  0 0 1
P4  3 4 1
Enter the process and request the resources:
Process: 1
Enter the resources: 0 0 0
Processes will start in the following sequence:
P2, P3, P4, P0, P1,
 5
There will be No dead lock.
Final Available Matrix:
11 10 9
```

Sample-3

```
-----PRESENT_SYSTEM_STATE-----
Processes       |Maximum        |Allocated
P0              2  2  2         1  1  1
P1              7  2  4         7  2  4
P2              1  2  3         1  2  3
P3              3  2  1         3  2  1
P4              3  1  2         3  1  2

Available Resources in the OS
0 0 0
Need Matrix
P0  1 1 1
P1  0 0 0
P2  0 0 0
P3  0 0 0
P4  0 0 0
Enter the process and request the resources:
Process: 3
Enter the resources: 1 1 1
Processes will start in the following sequence:
P1, P2, P3, P4, P0,
 5
There will be No dead lock.
Final Available Matrix:
15 8 11
```

Sample-4

```
-----PRESENT_SYSTEM_STATE-----
Processes       |Maximum        |Allocated
P0              2  3  1         2  3  1
P1              4  2  2         3  1  2
P2              7  7  7         2  1  3

Available Resources in the OS
0 1 0
Need Matrix
P0  0 0 0
P1  1 1 0
P2  5 6 4
Enter the process and request the resources:
Process: 0
Enter the resources: 0 0 0
Processes will start in the following sequence:
P0, P1,
 2

There will be a dead lock. So Can't allocate
Final Available Matrix:
5 5 3
```

Sample-5

```
-----PRESENT_SYSTEM_STATE-----
Processes       |Maximum        |Allocated
P0              3  3  3         3  3  2
P1              3  4  2         2  4  1

Available Resources in the OS
1 0 2
Need Matrix
P0  0 0 1
P1  1 0 1
Enter the process and request the resources:
Process: 0
Enter the resources: 3 5 3
Processes will start in the following sequence:
P1,
 1

There will be a dead lock. So Can't allocate
Final Available Matrix:
3 4 3
```

References:

http://rosettacode.org/wiki/Banker's_algorithm