CSCE 4600 Project 2

Members: Robert Martinez, Isaac Thomas, Thomas Truong, Clint Wyatt

Algorithm Description for Approach A

Using the expedient characteristic, we allocate the available resources to all processes that need allocation. This is done by decrementing the given available vector by the values in the lower left quadrant of the matrix, referred to as the assignment edge quadrant.

If a graph is in a deadlock free state, then processes can request resources without going over the number available units of a resource block. The algorithm checks this by checking whether a request for resources is less than or equal to the available resources. If the request by a process is less than or equal to what is available, then the process can make the request. Once the process is done executing, it gives its resources back to the system, incrementing the number of resources available.

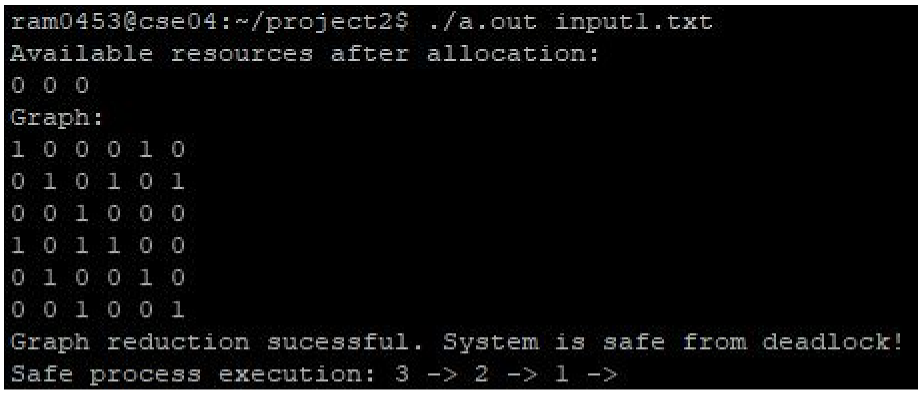
The function that performs this task by analyzing the upper right quadrant of the matrix, referred to as the request edge quadrant. If an entry in the request edge quadrant equals one, this represents a process wanting a unit of resource from a specific resource block. If there are not enough units of resources for that resource in the available vector, the process is blocked and the next process is checked. If the process is clear to request a resource, then the process is reduced from the graph by changing its allocation column entries to zero and the process is appended to the safe sequence list. The last step before checking the remaining processes is to increment the available vector by the process's requests. The function constantly checks the processes until either there are no remaining processes or none of the processes can be granted resources. If there are no remaining processes, then function returns true and shows the safe sequence. If none of the requests can be granted, then the function returns false and shows that there is a deadlock in the system.

Note: project2-algorithm.pdf has a visual representation for how the algorithm works.

Experimental Results

The following are results from executing our code with 3 different input files:

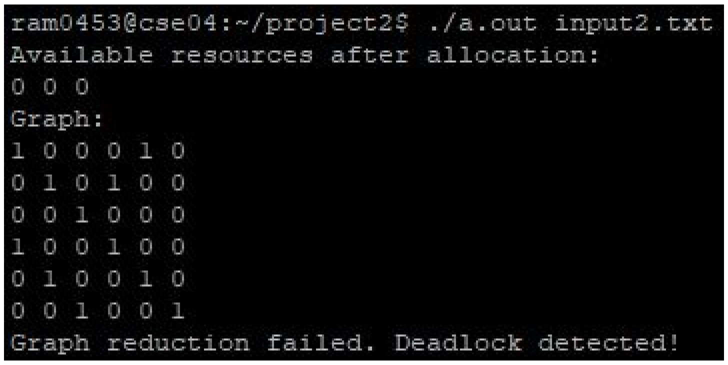
input1.txt:



```
ram0453@cse04:~/project2$ ./a.out input1.txt
Available resources after allocation:
0 0 0
Graph:
1 0 0 0 1 0
0 1 0 1 0 1
0 0 1 0 0 0
1 0 1 1 0 0
0 1 0 0 1 0
0 0 1 0 0 1
Graph reduction sucessful. System is safe from deadlock!
Safe process execution: 3 -> 2 -> 1 ->
```

Input1.txt is the same as the example given in the project 2 template. With the available vector being (2,1,1), the system is deadlock free because resource 1 has 2 available resources. When Process 3 is done executing, Resource 1 will have 1 unit of allocation from Process 3. Since Resource 1 and Resource 3 have a unit of a resource after P3 exits the system, Process 2 can go next since its requests for resource 3 and resource 1 can be granted.

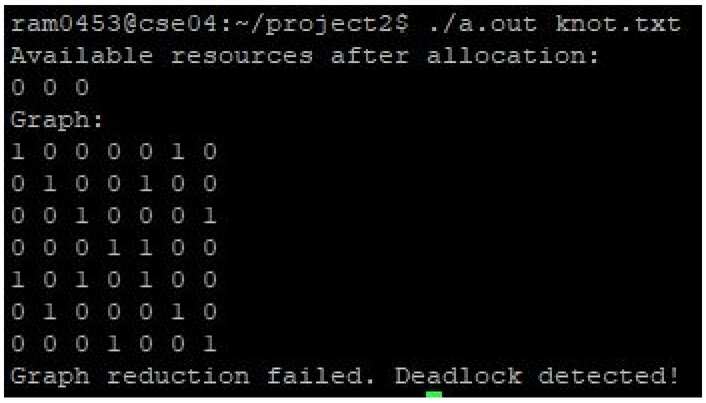
input2.txt



```
ram0453@cse04:~/project2$ ./a.out input2.txt
Available resources after allocation:
0 0 0
Graph:
1 0 0 0 1 0
0 1 0 1 0 0
0 0 1 0 0 0
1 0 0 1 0 0
0 1 0 0 1 0
0 0 1 0 0 1
Graph reduction failed. Deadlock detected!
```

Input2.txt is identical to input1.txt, the difference being the allocation vector is (1,1,1) instead of (2,1,1). This system deadlocks because there are not enough units for resource 1. After Process 3 exits the system, Process 2 cannot execute because there are no available units for resource 1, process 1 is holding resource 1. Process 1 cannot execute because process 2 is holding resource 2. Therefore, the kind of deadlock between process 1 and process 2 is “hold and wait”. In order for the deadlock to be eliminated, one of the processes must be killed.

knot.txt



```
ram0453@cse04:~/project2$ ./a.out knot.txt
Available resources after allocation:
0 0 0
Graph:
1 0 0 0 0 1 0
0 1 0 0 1 0 0
0 0 1 0 0 0 1
0 0 0 1 1 0 0
1 0 1 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 0 0 1
Graph reduction failed. Deadlock detected!
```

Knot.txt cannot be reduced because it is expedient and there is a knot in the graph. Since a knot and an expedient system will always produce a deadlock, the knot in the system cannot be reduced, hence the deadlock was detected.