BLG 312E Homework 3 Report

Mohamad Chahadeh - 150220901

30 - 05 - 2023

1 Introduction

In this Homework, We implemented Bankers' Algorithm, which is a resource allocation and deadlock avoidance algorithm used in operating systems. It is designed to prevent deadlocks by determining whether a particular resource allocation request from processes can be granted safely without causing deadlock.

1.1 Program Workflow

Three input files are given:

- **resources.txt**: contains the **total** number of resources available in the system for each resource.
- **allocations.txt**: contains the resources **already allocated** to each process
- **requests.txt**: a.k.a **needs** table, contains the **extra resources required** by each process in order for it to **finish execution**.

At the start of the program we are supposed to **read all three files**, and import the data into array variables, after that, the **Max table**, which is a table containing the maximum resources needed for process to finish executing, is calculated along the **available table**, which is a table containing the available resources for use in the system.
After that, **the bankers' algorithm is ran** on the arrays to determine if there is a deadlock or not, if there is, the processes causing the **deadlock are identified** and outputed into the console.

1.2 The Bankers' Algorithm

The following steps are taken in order:

1. the following tables need to be either available or calculated from available data:
   (a) Needs (Requests) table: Max - Allocated
   (b) Available Resources table: Max Resources - sum(Allocated)
2. set $P = 0$
3. if $P < Num\ of\ Processes$: **Increment P**, Else: Jump to step 6.
4. **if** $Needs[Process\ P] \leq Available\ Resources$, Then:
   (a) execute P
   (b) Mark P as finished
   (c) $Available\ Resource = Needs[ProcessP] + Available\ Resources$

(d) Jump to step 3.

5. **else:** Jump to step 3.
6. **if all processes finished**, Then there are **no deadlocks**.
7. **if not all processes finished**, Then:
   (a) if no processes finished in the previous cycle, **Then there is Deadlock with the unfinished processes**.
   (b) if some processes finished in previous cycle, **Jump to Step 2.**

2 Implementation

2.1 File reading

The files' contents were read using the $fopen()$ and $fscanf()$ functions, in the case of the resources file, one for loop was used to append the data to the array, while for the other files, nested for loops were used because they are 2-D array files.

2.2 Algorithm

The Algorithm was implemented based on the steps mentioned in the **Algorithm section** in the introduction in C language as follows:

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

int seq_counter = 0;
while(true) {
    bool found_process = false;

    for (int i = 0; i < MAX_PROCESS; i++) {

        if (finish[i]) continue;
        bool can_execute = true;

        for(int j = 0; j< MAX_RESOURCE; j++) {
            if (need[i][j] > work[j]) {
                can_execute = false;
                break;
            }
        }

        if(can_execute) {

            for(int j = 0; j< MAX_RESOURCE; j++) work[j] += allocation[i][j];

            safe_sequence[seq_counter++] = i;
            finish[i] = true;
            found_process = true;
```

```
            }
        }
        if(!found_process) break;
    }

    for (int i = 0; i < MAX_PROCESS; i++) {
        if (!finish[i]) {
            safe = false;
            break;
        }
    }
```

3 Conclusion

3.1 Output

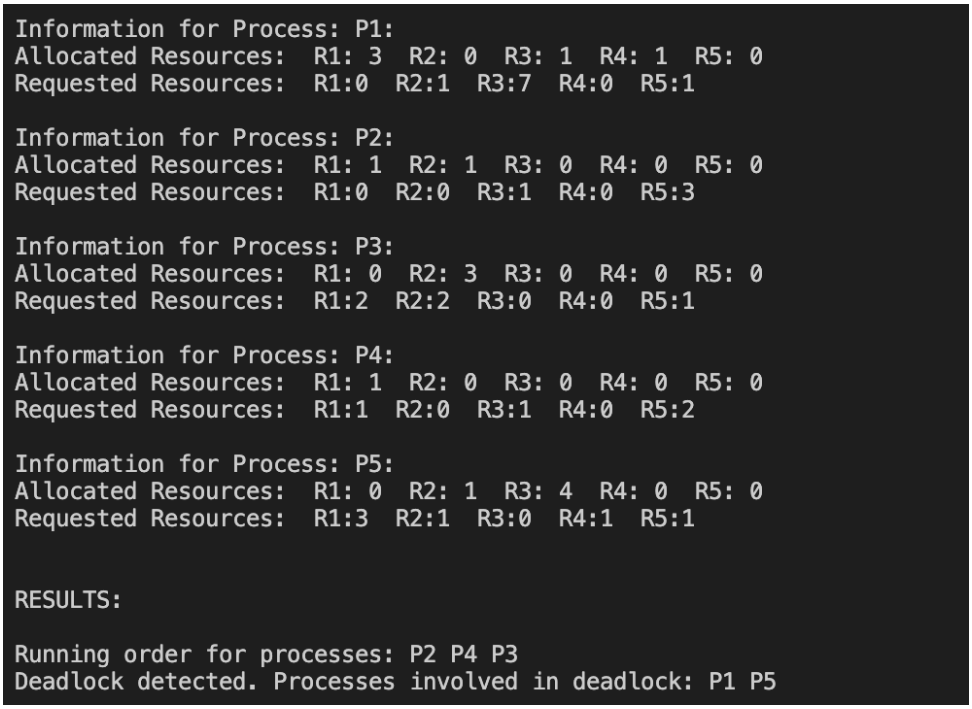

Figure 1: Output for the given input files

3.2 Discussion

The Bankers' Algorithm is a helpful way to detect deadlocks before they happen, and it can ensure that a system remains concurrent and without any issues, and in this experiment, I was able to implement it in C language without problems and got the correct output as shown above.