BLG 312E Operating Systems Homework 3

Due: Monday, June 5, 2023

Lecturer: Prof. Kemal Bıçakçı CRN: 23148

Fatih Baskın
150210710

Contents

Introduction

In this homework, we were asked to implement the Bankers' Algorithm, in which resource allocations to processes are determined without generating a deadlock. As a result, a work queue is generated for processes to prevent a deadlock. It might not be possible to prevent a deadlock because there might not be enough resources to do so, but still, the algorithm provides a way to run processes that don't cause a deadlock.

My implementation of such an algorithm works in an iterative fashion. It is `bankers.c` and using the `makefile`, writing `make bankers` will generate the executable file `bankers`. It asks for the number of processes and the number of resources from the console and prints the information about each process. Then it calculates the execution queue and prints the execution order and if there is a deadlock, it prints the processes causing deadlocks.

It reads from three text files, `allocations.txt` holds the allocation info of each process, `requests.txt` holds the additional requests of each process and `resources.txt` holds the total amount of resources in the system.

Implementation

My implementation runs in an iterative fashion using arrays. It has $O(n^2 \times m)$ complexity, n is the number of processes and m is the number of resources. It is possible to solve this problem using graph theory by finding a directed acyclic graph that has a lower complexity but for the sake of simplicity, I preferred an iterative array-based solution.

After reading all the necessary information from the text files and printing them, the remaining resources (total resources - allocated resources) are calculated from the processes. After that, the iterative algorithm works to calculate the working order (execution queue).

Inside a while loop, all processes are iteratively checked to whether there are enough resources to run them. The first process that can be run is marked as ran and put into the execution queue. After a process is marked as ran, it is not checked again. If all processes are run or there is no suitable process to run, then the loop is broken. After that, the process queue is printed and if there are any unmarked processes left, they are causing a deadlock and they are printed as deadlock processes.

Calculations

Process	R1	R2	R3	R4	R5
P1	3,0	0,1	1,7	1,0	0,1
P2	1,0	1,0	0,1	0,0	0,3
P3	0,2	3,2	0,0	0,0	0,1
P4	1,1	0,0	0,1	0,0	0,2
P5	0,3	1,1	4,0	0,1	0,1

Table 1: Each entry holds pre-allocated resources and resource requests

Calculations continued on next page...

Resource	Total	Available
R1	5	0
R2	7	2
R3	10	5
R4	2	1
R5	6	6

Table 2: Total and available resources at the beginning

1. **Iteration 1**
   (a) Process 1: Requests 7 of R3, there are only 5, skip.
   (b) Process 2: Process 2 can run, there are enough of R3 (requested: 1, available: 5) and R5 (requested: 3, available: 6). Process 2 executes and releases its resources. Newest available resources: **R1:1, R2:3, R3:5, R4:1, R5:6**.
2. **Iteration 2**
   (a) Process 1: Requests 7 of R3, there are only 5, skip.
   (b) Process 3: Requests 2 of R1, there are only 1, skip.
   (c) Process 4: Process 4 can run, there are enough of R1 (requested: 1, available: 1), R3 (requested: 1, available: 5) and R5 (requested: 2, available: 6). Process 4 executes and releases its resources. Newest available resources: **R1:2, R2:3, R3:5, R4:1, R5:6**.
3. **Iteration 3**
   (a) Process 1: Requests 7 of R3, there are only 5, skip.
   (b) Process 3: Process 3 can run, there are enough of R1 (requested: 2, available: 2), R3 (requested: 2, available 3) and R5 (requested: 1, available: 6). Process 3 executes and releases its resources. Newest available resources: **R1:2, R2:6, R3:5, R4:1, R5:6**.
4. **Iteration 4**
   (a) Process 1: Requests 7 of R3, there are only 5, skip.
   (b) Process 5: Requests 3 of R3, there are only 2, skip.
   (c) End of the processes list, no process can be worked anymore, breaking out of the loop.

The working order of the processes is **P2, P4, P3**.

P1 and **P5** are causing a deadlock.