

Banker's algorithm

The **Banker algorithm**, sometimes referred to as the **detection algorithm**, is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes an "s-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.

The algorithm was developed in the design process for the THE operating system and originally described (in Dutch) in EWD108.^[1] When a new process enters a system, it must declare the maximum number of instances of each resource type that it may ever claim; clearly, that number may not exceed the total number of resources in the system. Also, when a process gets all its requested resources it must return them in a finite amount of time.

Contents

Resources

- Example

- Safe and Unsafe States

- Requests

- Example

Limitations

References

Further reading

Resources

For the Banker's algorithm to work, it needs to know three things:

- How much of each resource each process could possibly request[**MAX**]
- How much of each resource each process is currently holding[**ALLOCATED**]
- How much of each resource the system currently has available[**AVAILABLE**]

Resources may be allocated to a process only if the amount of resources requested is less than or equal to the amount available; otherwise, the process waits until resources are available.

Some of the resources that are tracked in real systems are memory, semaphores and interface access.

The Banker's Algorithm derives its name from the fact that this algorithm could be used in a banking system to ensure that the bank does not run out of resources, because the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers^[2]. By using the Banker's algorithm, the bank ensures that when customers request money the bank never leaves a safe state. If the customer's request does not cause the bank to leave a safe state, the cash will be allocated, otherwise the customer must wait until some other customer deposits enough.

Basic data structures to be maintained to implement the Banker's Algorithm:

Let n be the number of processes in the system and m be the number of resource types. Then we need the following data structures:

- Available: A vector of length m indicates the number of available resources of each type. If $\text{Available}[j] = k$, there are k instances of resource type R_j available.
- Max: An $n \times m$ matrix defines the maximum demand of each process. If $\text{Max}[i,j] = k$, then P_i may request at most k instances of resource type R_j .
- Allocation: An $n \times m$ matrix defines the number of resources of each type currently allocated to each process. If $\text{Allocation}[i,j] = k$, then process P_i is currently allocated k instances of resource type R_j .
- Need: An $n \times m$ matrix indicates the remaining resource need of each process. If $\text{Need}[i,j] = k$, then P_i may need k more instances of resource type R_j to complete the task.

Note: $\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j]$. $n=m-a$.

Example

Total system resources are:

A	B	C	D
6	5	7	6

Available system resources are:

A	B	C	D
3	1	1	2

Processes (currently allocated resources):

	A	B	C	D
P1	1	2	2	1
P2	1	0	3	3
P3	1	2	1	0

Processes (maximum resources):

	A	B	C	D
P1	3	3	2	2
P2	1	2	3	4
P3	1	3	5	0

Need = maximum resources - currently allocated resources

Processes (possibly needed resources):

	A	B	C	D
P1	2	1	0	1
P2	0	2	0	1
P3	0	1	4	0

Safe and Unsafe States

A state (as in the above example) is considered safe if it is possible for all processes to finish executing (terminate). Since the system cannot know when a process will terminate, or how many resources it will have requested by then, the system assumes that all processes will eventually attempt to acquire their stated maximum resources and terminate soon afterward. This is a reasonable assumption in most cases since the system is not particularly concerned with how long each process runs (at least not from a deadlock

avoidance perspective). Also, if a process terminates without acquiring its maximum resource it only makes it easier on the system. A safe state is considered to be the decision maker if it's going to process ready queue.

Given that assumption, the algorithm determines if a state is **safe** by trying to find a hypothetical set of requests by the processes that would allow each to acquire its maximum resources and then terminate (returning its resources to the system). Any state where no such set exists is an **unsafe** state.

We can show that the state given in the previous example is a safe state by showing that it is possible for each process to acquire its maximum resources and then terminate.

1. P1 needs 2 A, 1 B and 1 D more resources, achieving its maximum
 - [available resource: $\langle 3 \ 1 \ 1 \ 2 \rangle - \langle 2 \ 1 \ 0 \ 1 \rangle = \langle 1 \ 0 \ 1 \ 1 \rangle$]
 - The system now still has 1 A, no B, 1 C and 1 D resource available
2. P1 terminates, returning 3 A, 3 B, 2 C and 2 D resources to the system
 - [available resource: $\langle 1 \ 0 \ 1 \ 1 \rangle + \langle 3 \ 3 \ 2 \ 2 \rangle = \langle 4 \ 3 \ 3 \ 3 \rangle$]
 - The system now has 4 A, 3 B, 3 C and 3 D resources available
3. P2 acquires 2 B and 1 D extra resources, then terminates, returning all its resources
 - [available resource: $\langle 4 \ 3 \ 3 \ 3 \rangle - \langle 0 \ 2 \ 0 \ 1 \rangle + \langle 1 \ 2 \ 3 \ 4 \rangle = \langle 5 \ 3 \ 6 \ 6 \rangle$]
 - The system now has 5 A, 3 B, 6 C and 6 D resources
4. P3 acquires 1 B and 4 C resources and terminates.
 - [available resource: $\langle 5 \ 3 \ 6 \ 6 \rangle - \langle 0 \ 1 \ 4 \ 0 \rangle + \langle 1 \ 3 \ 5 \ 0 \rangle = \langle 6 \ 5 \ 7 \ 6 \rangle$]
 - The system now has all resources: 6 A, 5 B, 7 C and 6 D
5. Because all processes were able to terminate, this state is safe

For an example of an unsafe state, consider what would happen if process 2 was holding 2 units of resource B at the beginning.

Requests

When the system receives a request for resources, it runs the Banker's algorithm to determine if it is safe to grant the request. The algorithm is fairly straightforward once the distinction between safe and unsafe states is understood.

1. Can the request be granted?
 - If not, the request is impossible and must either be denied or put on a waiting list
2. Assume that the request is granted
3. Is the new state safe?
 - If so grant the request
 - If not, either deny the request or put it on a waiting list

Whether the system denies or postpones an impossible or unsafe request is a decision specific to the operating system.

Example

Starting in the same state as the previous example started in, assume process 3 requests 2 units of resource C.

1. There is not enough of resource C available to grant the request
2. The request is denied

On the other hand, assume process 3 requests 1 unit of resource C.

1. There are enough resources to grant the request
2. Assume the request is granted
 - The new state of the system would be:

```
Available system resources
  A B C D
Free 3 1 0 2
```

```
Processes (currently allocated resources):
  A B C D
P1  1 2 2 1
P2  1 0 3 3
P3  1 2 2 0
```

```
Processes (maximum resources):
  A B C D
P1  3 3 2 2
P2  1 2 3 4
P3  1 3 5 0
```

1. Determine if this new state is safe
 1. P1 can acquire 2 A, 1 B and 1 D resources and terminate
 2. Then, P2 can acquire 2 B and 1 D resources and terminate
 3. Finally, P3 can acquire 1 B and 3 C resources and terminate
 4. Therefore, this new state is safe
2. Since the new state is safe, grant the request

Final example: from the state we started at, assume that process 2 requests 1 unit of resource B.

1. There are enough resources
2. Assuming the request is granted, the new state would be:

```
Available system resources:
  A B C D
Free 3 0 1 2
```

```
Processes (currently allocated resources):
  A B C D
P1  1 2 5 1
P2  1 1 3 3
P3  1 2 1 0
```

```
Processes (maximum resources):
  A B C D
P1  3 3 2 2
```

```
P2  1 2 3 4
P3  1 3 5 0
```

1. Is this state safe? Assuming P1, P2, and P3 request more of resource B and C.
 - P1 is unable to acquire enough B resources
 - P2 is unable to acquire enough B resources
 - P3 is unable to acquire enough B resources
 - No process can acquire enough resources to terminate, so this state is not safe
2. Since the state is unsafe, deny the request

```
import numpy as np

n_processes = int(input('Number of processes? '))
n_resources = int(input('Number of resources? '))

available_resources = [int(x) for x in input('Claim vector? ').split(' ')]

currently_allocated = np.array([[int(x) for x in input('Currently allocated for process ' + str(i + 1) + '? ').split(' ')] for i in range(n_processes)])
max_demand = np.array([[int(x) for x in input('Maximum demand from process ' + str(i + 1) + '? ').split(' ')] for i in range(n_processes)])

total_available = available_resources - np.sum(currently_allocated, axis=0)

running = np.ones(n_processes) # An array with n_processes 1's to indicate if process is yet to run

while np.count_nonzero(running) > 0:
    at_least_one_allocated = False
    for p in range(n_processes):
        if running[p]:
            if all(i >= 0 for i in total_available - (max_demand[p] - currently_allocated[p])):
                at_least_one_allocated = True
                print(str(p) + ' is running')
                running[p] = 0
                total_available += currently_allocated[p]
    if not at_least_one_allocated:
        print('Unsafe')
        exit()

print('Safe')
```

Limitations

Like the other algorithms, the Banker's algorithm has some limitations when implemented. Specifically, it needs to know how much of each resource a process could possibly request. In most systems, this information is unavailable, making it impossible to implement the Banker's algorithm. Also, it is unrealistic to assume that the number of processes is static since in most systems the number of processes varies dynamically. Moreover, the requirement that a process will eventually release all its resources (when the process terminates) is sufficient for the correctness of the algorithm, however it is not sufficient for a practical system. Waiting for hours (or even days) for resources to be released is usually not acceptable.

References

1. Dijkstra, Edsger W. *Een algorithmie ter voorkoming van de dodelijke omarming (EWD-108)* (<http://www.cs.utexas.edu/users/EWD/ewd01xx/EWD108.PDF>) (PDF). E.W. Dijkstra Archive. Center for American History, University of Texas at Austin. (transcription (<http://www.cs.utexas.edu/users/EWD/transcriptions/EWD01xx/EWD108.html>)) (in Dutch; *An algorithm for the prevention of the deadly embrace*)
2. Silberschatz, Galvin, & Gagne (2013). *Operating System Concepts, 9th Edition*. Wiley. p. 330. ISBN 978-1-118-06333-0.

Further reading

- "Operating System Concepts (<https://web.archive.org/web/20101013000653/http://codex.cs.yale.edu/avi/os-book/OS7/os7c/index.html>)" by Silberschatz, Galvin, and Gagne (pages 259-261 of the 7th edition)
 - "Operating System Concepts (<http://codex.cs.yale.edu/avi/os-book/OS8/os8c/index.html>)" by Silberschatz, Galvin, and Gagne (pages 298-300 of the 8th edition)
 - Dijkstra, Edsger W. *The mathematics behind the Banker's Algorithm (EWD-623)* (<http://www.cs.utexas.edu/users/EWD/ewd06xx/EWD623.PDF>) (PDF). E.W. Dijkstra Archive. Center for American History, University of Texas at Austin. (transcription (<http://www.cs.utexas.edu/users/EWD/transcriptions/EWD06xx/EWD623.html>)) (1977), published as pages 308–312 of Edsger W. Dijkstra, *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag, 1982. ISBN 0-387-90652-5
-

Retrieved from "https://en.wikipedia.org/w/index.php?title=Banker%27s_algorithm&oldid=951205787"

This page was last edited on 16 April 2020, at 00:47 (UTC).

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.