Marbo Cheng, Sherry Feng
Professor Andrew Segal
Operating Systems 326

March 8th, 2017

Program 2 - Banker's algorithm

Files: bankers.c, WriteUp.pdf

Compile:
gcc bankers.c -o bankers -lpthread

Run:
./bankers 10 5 7
(the 3 numbers after the file name is the number to input, 10, 5 ,7 is an example but any other 3 numbers would suffice)

This code is a multithreaded implementation of the banker's algorithm, each thread will request and keep running the banker's algorithm until the user stops the program. It will also show if it has insufficient resources, and also show the allocation when the thread releases the resources.

We first began writing the bankers algorithm, once we finished that, we tested with some input values. One issue we faced with the banker's algorithm is our search array, which we implemented to check if the processes has been looked at or not. It was a bit difficult of keeping track what each number meant, for example in our search array 0 = untested, 1 = tested, and it got confusing while we tried to debug. Instead we created new variables named 'safe' and 'execute' which played the role of what we needed and made the code looked cleaner and also debugged it. Then we wrote a random number generator and the request function. The request function takes in the request of the threads (request is randomly generated) and approves if the request plus allocation is less or equal to the maximum. We then tested it with threads, but we faced another issue which was that we neglected putting 'pthread_join' in the main, neglecting that caused the program to only stop after 1 thread. But after some guidance from the professor we had fixed it.
The next step we wrote the release function. Finally we put it altogether in a while(1) loop for the program to constantly run, but the values that were printed of request and allocation did not seem quite right, we then looked at the location of the print statements and moved it in order to print the accurate information.
Lastly, to delay time we did not use the sleep() function because it terminates the threads instead of letting the threads wait. So instead we wrote a function waitFor() that simply creates a delay time used for releasing resources.

Figure 1. Output of bankers.c

```
Customer 1 | Request   0 2 0 | Available = 9 2 7  | Allocation = 0 2 0
Customer 2 | Request   0 3 1 | Available = 9 2 7 | INSUFFICIENT RESOURCES
Customer 3 | Request   0 1 0 | Available = 9 1 7  | Allocation = 0 1 0
Customer 4 | Request   0 1 2 | Available = 9 0 5  | Allocation = 0 1 2
Customer 1 | Releasing 0 2 0 | Available = 9 2 5 | Allocated = 0 0 0
Customer 4 | Releasing 0 1 2 | Available = 9 3 7 | Allocated = 0 0 0
Customer 1 | Request   1 2 0 | Available = 8 1 7  | Allocation = 1 2 0
Customer 4 | Request   2 1 1 | Available = 6 0 6  | Allocation = 2 1 1
Customer 4 | Releasing 2 1 1 | Available = 8 1 7 | Allocated = 0 0 0
Customer 4 | Request   0 0 1 | Available = 8 1 6  | Allocation = 0 0 1
Customer 4 | Releasing 0 0 1 | Available = 8 1 7 | Allocated = 0 0 0
Customer 4 | Request   2 2 1 | Available = 8 1 7 | INSUFFICIENT RESOURCES
Customer 4 | Request   1 0 2 | Available = 7 1 5  | Allocation = 1 0 2
Customer 4 | Releasing 1 0 2 | Available = 8 1 7 | Allocated = 0 0 0
Customer 3 | Releasing 0 1 0 | Available = 8 2 7 | Allocated = 0 0 0
Customer 4 | Request   0 1 0 | Available = 8 1 7  | Allocation = 0 1 0
Customer 3 | Request   1 4 1 | Available = 8 1 7 | INSUFFICIENT RESOURCES
```