CSC 456 Operating Systems
Dr. Christer Karlsson
Benjamin Kaiser
Programming Assignment 2
Part 1 Banker's Algorithm Simulator

Description of Program

This program is a simulation of a banker's algorithm process to avoid deadlocks. The assignment requirements specified that we needed to use mutex locks around the arrays

Description of algorithms and libraries used

The only special library that was used in this program was the pthread.h library. This library provides the functionality and everything for using POSIX threads.

A description of the algorithm can be found in the description of functions and program structure.

Description of functions and program structure

This program is a procedurally designed program. It consists of seven individual functions including main. These functions are spread out over two files.

The first file is called bank.c. This file contains the main function. This function simply does most of the initialization and creation of threads and structures. It takes arguments from the command line which are then converted into integers to be the number of initial available resources. These are automatically set into the available array. The threads are then created and joined into the main thread.

The second function is entitled init_arrays. This is called by the main function and it bases all of the initializations off of the original available values. The allocations are originally set to 0. The maximum for each customer is the original available divided by half of the number of customers. The need is then calculated by doing the maximum minus the currently allocated (0).

The third function is then the safety_test function which is required per the assignment documentation. This function actually implements the banker's algorithm. This means that the first thing that needs to happen is that we copy our currently available resources into a test array to compute the banker's algorithm. We then loop through each process (customer) and we originally assume that all of the needed resource count for each process are less than the available. We check to make sure this is true and if it is true, we then do a test run on this, and pretend the process terminates so that add the resources that it had to the currently available test resource count. We mark the process as finished (take it out of the set of processes) and then proceed again. As long as for each iteration we remove at least one process, we are good

and can grant the request and continue to next iteration. However if we go all the way through and find that there was at least one process that didn't finish then we return an unsafe state.

The other file is called customer.c. The first function is called print_state and this just takes care of printing out the tables for each request that occurs. It prints out the table headers and then for each table it prints out the column headers and then prints out the rows for each process.

The next function on this file is the customer_function. It is the function that each thread is running and it essentially consists of a while loop that runs until it has gone through all of the resources that it required. It requests a random amount of resources bounded by the amount of resources that the process still needs. The return value for this is set to a flag which is used to decide whether or not a process can release resources. If the process was not allocated resources, it isn't going to release any. We then check to see if the process is done by looking at all of the resource need values. If all these are zero, the process is done and can terminate. Then if the flag for releasing was set, the process releases its resources.

The next function is the request_resources function. This was also required per the assignment documentation. The function tries to acquire a mutex which locks all the arrays for the bank. When this happens, it prints out the request to the screen. It then checks to make sure that the number of resources requested were less than the currently available. This automatically returns and unsafe state. If this isn't the case, it proceeds into the banker's algorithm. If this is a safe state, the allocation to the process is made and all arrays are updated accordingly. In the event that the need array goes negative (shouldn't ever happen with the above check) then the need array is clipped. The function then returns safe to the main thread function after unlocking the mutex.. If the safety test returns an unsafe state, then the function prints out a DENIED status with an unsafe reason. The function then unlocks and returns.

The final function is the release_resources function which was also required by the assignment description. It locks a mutex so that no race conditions occur. A release statement is printed that we are releasing resources. The resources are then released and the appropriate array values are updated. The function the unlocks the mutex and if this is successful, it returns safe to the thread.

Description of testing and verification process

The way I tested this program was two run it several different times and examine the output by hand. It appeared to be right. I did notice that my output is interleaved and this is because the printf function is not safe. I attempted to put a lock around the print statements so that the printing wouldn't happen, however I found that I wasn't ever getting denied requests after several runs so I left this in and dealt with the poorly formatted output. Since the timing on this program was not entirely relevant, I did no timing.

Data

There was not really any data collected for this project. The only data that the program produces is its output which can be redirected to a file.

Analysis of Data

The only analysis of the data that I did was a manual observation of the file.

Description of Submission

The contents of the submission of this program are as follows:

- bank.c -- This file contains the main function, the initialization functions and the actual banker's algorithm
- customer.c This file contains all functions required by the customer. These include the table printing, the release_resources, the request_resources and the main loop function for each customer thread.
- bank.h This file contains the globals and #defines for the values required for the program. It also contains a function prototype for the safety_test function.
- customer.h This file only contains the function prototypes for the functions that are implemented in customer.c