EI338 Lab Report Chapter5&7

Bao Xiaoyi 517030910306

November 21, 2019

1 Project 8 Banker's Algorithm

1.1 Environment

- VirtualBox 5.2.18
- Ubuntu 14.04 (64 bit) running on the virtual machine

1.2 Assignment

Implement the banker's algorithm with either C or Java. With interactive command line, customers request and release resources from the bank(program). The banker will grant a request only if it leaves the system in a safe state. To be more specific, the following functions should be implemented:

- Get a command from the user.
- Decode the command to see what the user asks for
- Judge whether it is a valid command, if it is, then executes it.

1.3 Decomposition and Analysis

Here, we try to decompose each of the original task into smaller parts, and solve the sub problems one by one.

1. Design a good structure to store the information about a specific task. Initialize all of them.

   Solution. *We use two-dimensional array to store maximum, allocation and need. The array of given resources is stored in one-dimensional array available.*
   At the beginning of the program, available is read from the command line input. We also open the file which stores the information of the maximum number of resources for each process, then store it in maximum. allocation is initialized to be 0, thus need = maximum - allocation = maximum.

2. Get a command from the user, and choose a branch for execution.

Solution. *First, we use fget to receive an input from the user. Then we decide what we should do according to the first word of the command. If it is QUIT, then we end the whole program. Else, the command that the user requests or releases is executed if it is valid.*

3. Judge whether the state is still safe if we choose to perform a request.

   Solution. *The pseudo code for the full algorithm is shown in the textbook(operating systems concept: 10th edition , P335 safe algorithm). Be sure that we do not actually change the shared data in the procedure. Instead, we create a copy for each of them and try to perform operation on the copy.*

4. Make modification to the data structures that are used for storage.

   Solution. *There are two kinds of operations: request or release. If we have already confirmed that the input can be performed, we just modify need allocation.*

1.4 Details

In this part, some codes are shown for better illustration.

- When getting input from the file, we need to remove the "
  n" at the end of each line, so that the function *strtok* can get the right input.

```
while(getline(&line, &len, fptr) != -1){
        line[strlen(line)-1] = '\0';   /* remove \n */
        token_num = 0;
        token = strtok(line, ",");
        while (token != NULL){
                maximum[line_num][token_num] = atoi(token);
                token = strtok(NULL, ",");
                token_num += 1;
        }
        line_num += 1;
}
fclose(fptr);
```

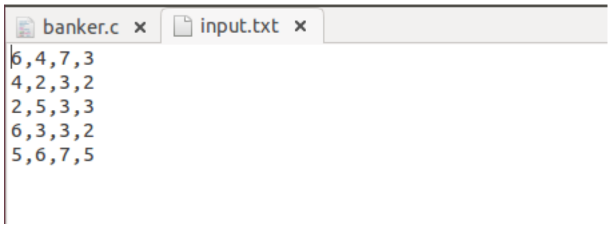
- In the part for testing request, we create a copy for the original one, and work on the copy to decide whether the state is still safe. The request should be included in allocation first:

```
/* make a copy*/
for (i=0; i<NUMBER_OF_CUSTOMERS; i++){
        for (j=0; j<NUMBER_OF_RESOURCES; j++){
                need_new[i][j] = need[i][j];
                allocation_new[i][j] = allocation[i][j];
        }
}


for (i=0; i<NUMBER_OF_RESOURCES; i++){
        work[i] = available[i] - request[i];
        need_new[customer_num][i] -= request[i];
        allocation_new[customer_num][i] += request[i];
}
```

1.5 Result

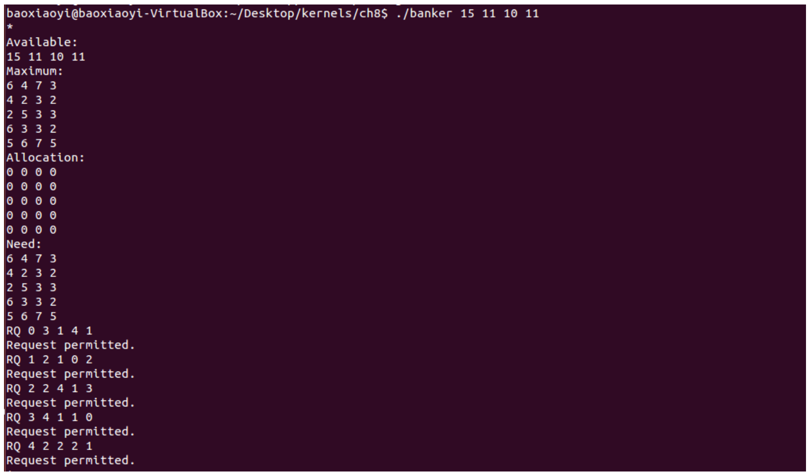
We use a toy example to test our scheduling algorithms. Here is the input file:



```
6,4,7,3
4,2,3,2
2,5,3,3
6,3,3,2
5,6,7,5
```

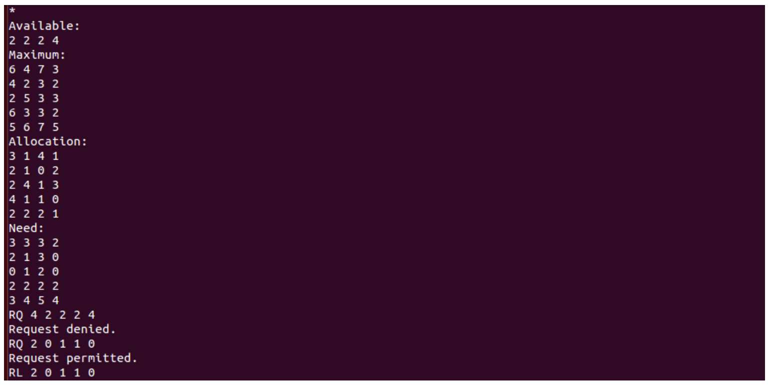
Figure 1: Toy test example: input file

Here are the results of our experiment:



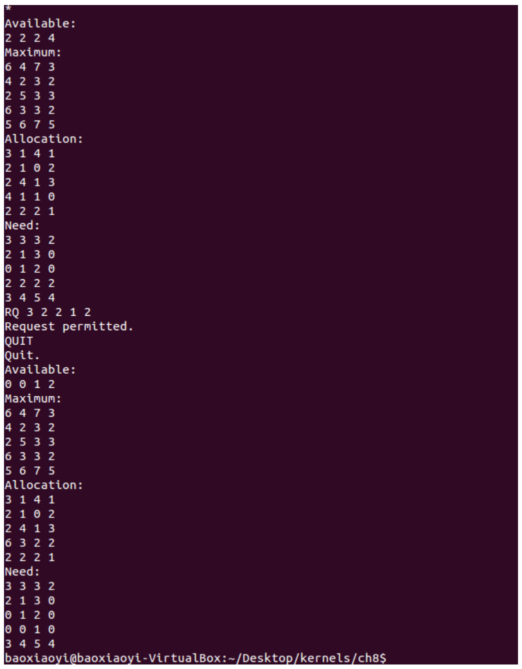
```
baoxiaoyi@baoxiaoyi-VirtualBox:~/Desktop/kernels/ch8$ ./banker 15 11 10 11
*
Available:
15 11 10 11
Maximum:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Allocation:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
Need:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
RQ 0 3 1 4 1
Request permitted.
RQ 1 2 1 0 2
Request permitted.
RQ 2 2 4 1 3
Request permitted.
RQ 3 4 1 1 0
Request permitted.
RQ 4 2 2 2 1
Request permitted.
```

Figure 2: Toy test example: result1



```
*
Available:
2 2 2 4
Maximum:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Allocation:
3 1 4 1
2 1 0 2
2 4 1 3
4 1 1 0
2 2 2 1
Need:
3 3 3 2
2 1 3 0
0 1 2 0
2 2 2 2
3 4 5 4
RQ 4 2 2 2 4
Request denied.
RQ 2 0 1 1 0
Request permitted.
RL 2 0 1 1 0
```

Figure 3: Toy test example: result2



```
*
Available:
2 2 2 4
Maximum:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Allocation:
3 1 4 1
2 1 0 2
2 4 1 3
4 1 1 0
2 2 2 1
Need:
3 3 3 2
2 1 3 0
0 1 2 0
2 2 2 2
3 4 5 4
RQ 3 2 2 1 2
Request permitted.
QUIT
Quit.
Available:
0 0 1 2
Maximum:
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
Allocation:
3 1 4 1
2 1 0 2
2 4 1 3
6 3 2 2
2 2 2 1
Need:
3 3 3 2
2 1 3 0
0 1 2 0
0 0 1 0
3 4 5 4
baoxiaoyi@baoxiaoyi-VirtualBox:~/Desktop/kernels/ch8$
```

Figure 4: Toy test example: result3

1.6 Future work

There are several small points that we may improve in the future:

- Currently, the number of customers and resources are fixed. In fact, We may ask the user for these parameters, and create storage for information with dynamic memory allocation.

- We may allow user to give the input file name as another parameter.