# Programming Assignment 1: Multi-user Book Management Systems

**Announce Day: Wednesday October 17, 2012.**
**Due Day: 24:00 Tuesday October 30, 2012.**

## GOAL:

In this assignment, you will learn how to conduct atomic operations on file I/O. You will write four programs: **`query`**, **`borrow`**, **`query_borrow`**, and **`return`**. The users will execute any of these programs simultaneously to access the books in library. Your responsibility is to assure that the users can query the information of the books, borrow books from the library, and return books to the library.

The processes share two files: *books* and *book.log*. Each file consists of records of book information: each record in file *books* contains book name, borrower's ID and a list of reservation requests; each record in file *book.log* stores the result of "borrow", "return" and "query_borrow". In this assignment, you need to pay attention to data consistency of the file. In short, you need to assure all the writes to the file are atomic and synchronize the file access when necessary. You will learn how to conduct atomic operation, file locks, and pay attention to the performance tradeoff of using file locks.

## Format of Library Database File, *books*

File ''books'' is a binary file which consists of numbers of fixed-sized records. Every record consists of the information for one book.

The following is the format of one book record:

| 32 bytes | 4 bytes | 16*4 bytes |
|---|---|---|
| Title | number_of_resv_req | userID_for_resv_request |

*title (string)*
book title

*number_of_resv_req (integer)*
the number of reservation requests for this book and is always a non-negative number. When it is zero, no users request for the book. Its maximum value is 15.

*userID_for_resv_request (array of integers)*
There are at most sixteen 4-bytes user_id in this field. The first one[1] is the user ID for the current borrower of the book; the second is that for the first reservation request; the third one is that for the second reservation request and so on. If the book is not borrowed by anyone, the first entry of the userID_for_resv_request is marked by a constant NO_USER(-1). The *book_idx* of each book represents the ID of book in file *books* but is NOT stored in the file. If the ID of a book is *n*, the beginnig of the book record is located at sizeof(BookInfo)*n bytes of the file.

## Format of Library Log File, *book.log*

File book.log is a file which logs the operations conducted by "borrow", "return" and "query_borrow". The details will be described in specification of above programs.

## Format of Transaction File, *Tx*

We also provide a file named "Tx"[2], which is related to "books"[3]. You may use it to implement atomic operations. The file structure of *Tx*'s is similar to that of file *books*, but it has only one field, named userID, for each book. Every book's userID in file *Tx* is set to NO_USER(-1) initially.

## Supplied Header File: library.h

Header file *library.h* defines the data structure for book record, the data structure for "Tx" record and the constant used in the program. You can download the header file from the CEIBA

[1] userID_for_resv_request[0]
[2] "Tx" means "Transaction".
[3] The number of entries in "Tx" is same as the number of entries in "books".

and should not modify this header file. Samples of file *books* and *Tx* would be provided too, you can use them to test your program.

library.h

```
#define MAX_TITLE_LENGTH (32)
#define MAX_RESERVE (16)

struct BookInfo{
        char title[MAX_TITLE_LENGTH];
        int number_of_resv_req;/*>=0, <MAX_RESERVE*/
        int userID_for_resv_request[MAX_RESERVE];
};
struct BookTx{
       int userID;
};
#define NO_USER (-1)
/*userID_for_resv_request[0] is either NO_USER or integer >=0*/
```

## Programs Semantics

Every program interacts with users by command-line arguments. If the programs received invalid arguments, the programs append a line to the ‘‘book.log’’ file and terminate the programs. Last but not the least, the programs should terminate within a reasonable time.

When a user would like to borrow a book from the library, he/she could executes program “query” to get the status of the book, then executes program “borrow” and his/her user ID will be inserted into the userID_for_resv_request when the request succeeds. User could execute program “query_borrow” if he/she wants to perform these two operations atomically.
When the student returns a borrowed book, the first four bytes of userID_for_resv_request will be set to NO_USER(-1).
In the following, we describe the semantic and requirement of the four programs.

[NAME]
`query`
[SYNOPSIS]
`query book_idx`
[DESCRIPTION]
“query” program prints information of the record on the terminal. book_idx is a nonnegative integer which is the index of book records in “books”.
[OUTPUT FORMAT]
After every execution, “query” program must output three lines to the terminal. The first line is the title and number_of_resv_req of the book[4]; the second and third line contains eight user_ids of userID_for_resv_request, and each user_id is separated by a space in a line. The “#” represents unused entries.[5]

If the book_idx is less than 0 or greater the size of the file, the output is “invalid book index book_idx”.

[Sample execution 1]
$ query 1245

[Sample ouput 1]
UnixProgramming 1
452 584 # # # # # #
# # # # # # # #

[4] printf(“%s %d\n”,title, number_of_resv_req)
[5] You must output -1 if the userID_for_resv_request[0]==NO_USER

| [Sample excution 2]<br>$ query 154 |
| --- |
| [Sample output 2](no borrower)<br>LinuxKernel 0<br>-1 # # # # # # #<br># # # # # # # |

| [Sample execution 3:invalid book index.]<br>$ query -1 |
| --- |
| [content of ‘‘book.log’’]<br>…<br>invalid book index -1 |

[NAME]

`borrow`

[SYNOPSIS]

`Borrow book_idx user_id`

[DESCRIPTION]

To borrow books from the library, a user execute the “borrow” program with “book_idx” and “user_id” as command-line arguments. book_idx is a nonnegative integer which is the index of the book record in file “books”, and user_id is a nonnegative integer which represents the borrower.

Process *borrow* updates two files to keep track of the books and user operations.

In file *books*, process *borrow* update the book record in three cases:

1. Write the user_id into the first entry in the userID_for_resv_request. The first entry is updated if one of the following conditions is true:

   a. number_of_resv_req is zero and the first user_id in the userID_for_resv_request is NO_USER (-1);

   b. number_of_resv_req is between 1 ~ 15, the first user_id is NO_USER, and the second element in userID_for_resv_request array is same as the given *user_id*.

2. Append the given *user_id* into the userID_for_resv_request[6] and increase number_of_resv_req by 1. The given *user_id* is inserted if one of the following conditions is true:

   a. number_of_resv_req is zero and the first user_id in the userID_for_resv_request is not NO_USER (-1);

   b. number_of_resv_req is between 1 ~ 14 and the second element in userID_for_resv_request array is different from the given *user_id*.

3. Shift the user IDs in number_of_resv_req to the left for 4 bytes, and decrease number_of_resv_req by 1. When the second element in userID_for_resv_request array is the same with the input user_id, the process shifts left all user_ids for one entry.

4. You should not write “books” file if one of the following conditions is true:

   a. number_of_resv_req is 15 and the second user_id is different from the input user_id;

[6] userID_for_resv_request[++number_of_resv_req]= given_user_id.

b. the input user_id has reserved[7] or borrowed the book;

c. invalid book_idx or user_id

[OUTPUT FORMAT]

After every execution, "borrow" program appends the output to the "book.log" file, not the terminal:

1. If number_of_resv_req is 0 and userID_for_resv_request[0] is NO_USER, result is "user_id borrowed book_idx[8]";

2. If number_of_resv_req is 0 and userID_for_resv_request[0] is not NO_USER, result is "user_id reserved book_idx";

3. If number_of_resv_req is between 1~15, userID_for_resv_request[0] is NO_USER, and userID_for_resv_request[1] is the same as input user_id, the output is "user_id borrowed book_idx";

4. If number_of_resv_req is between 1~14 and input user_id doesn"t appear in the userID_for_resv_request array, result is "user_id reserved book_idx";

5. If number_of_resv_req is 15 or the input user_id already reserved or borrowed the book, "user_id couldn"t borrow book_idx".

6. If the book_idx is less than 0 or greater than the size of the file, the output is "invalid book index book_idx".

7. When someone is performing "query_borrow" to same book and not finished yet, no matter what status of book user got, the output is "user_id failed to borrow book_idx".

| [Sample execution 1: before execution, number_of_resv_req is zero]<br>$ borrow 1245 452 |
|---|
| [content of "book.log"]<br>…<br>452 borrowed 1245 |

| [Sample execution 2: before execution, number_of_resv_req is one]<br>$ borrow 1245 584 |
|---|
| [content of "book.log"]<br>…<br>584 reserved 1245 |

| [Sample execution 3: before execution, userID_for_resv_request is full]<br>$ borrow 1245 455 |
|---|
| [content of "book.log"]<br>…<br>455 couldn't borrow 1245 |

| [Sample excution 4: invalid book index or user_id]<br>$ borrow -1 455<br>$ borrow 3 -1<br>$ borrow -1 -1 |
|---|

[7] Prerequisite is that userID_for_resv_request[1] is not given user_id, or the book will be borrowed.

[8] sprintf(str, "%d %s %d\n",user_id, "borrowed", book_idx);

| |
|---|
| [content of “book.log”]<br>…<br>invalid book index-1<br>invalid user ID -1<br>invalid user ID -1 |

| |
|---|
| [Sample execution 5: When someone is performing “query_borrow” to book 1245 and has not made his decision yet]<br>$ borrow 455 1245 |
| [content of “book.log”]<br>…<br>455 failed to borrow 1245 |

[NAME]

`query_borrow` – query book information and check out the book from library if available

[SYNOPSIS]

```
query_borrow book_idx user_id
```

[DESCRIPTION]

To perform “query” and “borrow” atomically, a user execute the ‘‘query_borrow’’ program. In short, it’s a combination of “query” and “borrow”.

Process query_borrow has two phases: query and borrow if available. In query phase, it displays if the book is available for checking out; in borrow phase, it assists the users to check out the book from the library if the book is available and the user would like to check it out.

Book ID ‘‘book_idx’’is a nonnegative integer which is also the index of the book recordsin file ‘‘books’’.

In the first phase, the process displays the information of the queried book.

In the second phase, the process asks if the user would like to check out or reserve the book. If the file is available for check out, the process shows

```
Check out the book (y/n)?
```

If the user answer ‘y’, the book should be checked out. If the book is not available for check out, the process shows

```
Reserve the book (y/n)?
```

If the user answer ‘y’, the book should be reserved for the user. If the answer of the above questions are no, the process terminates and does not change the status of the book.

These above operations must be **atomic**[9]. For example, there are three users execute program “query_borrow”, named A, B and C. A queries a book first, but A has not decided to borrow it yet. Then B and C are querying the same book simultaneously. If they input “y”, they will fail to borrow this book and their processes will be terminated, even if A has decided not to borrow it before B and C decide to borrow.
But if B has already borrowed the book, B could return it while A is performing “query_borrow” .

The details of query phase and borrow phase, including error messages, are as same as the details

[9] You could get help by using file “Tx” or finish it in your own way. By writing userID to the book entry in “Tx”, you can determine who has the right to borrow it now.

described in the specification of program “query” and “borrow”.

[OUTPUT FORMAT]
If there are multiple users who performed “query_borrow” simultaneously and all of them haven’t finished their operation yet, only the one who performed “query_borrow” first has right to borrow. Others will output “user_id failed to borrow book_idx” and terminates the program, no matter what status of book others got.[10]

| [Sample execution 1]<br>$ query_borrow 1245 1 |
|---|
| [Sample ouput 1](no borrower)<br>UnixProgramming 0<br>-1 # # # # # # #<br># # # # # # # #<br>Check out the book (y/n)?<br>y<br>[content of “book.log”]<br>…<br>1 borrowed 1245 |

| [Sample execution 2]<br>$ query_borrow 1245 -1 |
|---|
| [content of “book.log”]<br>…<br>invalid user ID -1 |

| [Sample execution 3]<br>$ query_borrow 1245 1 |
|---|
| [Sample ouput 3]<br>UnixProgramming 1<br>452 584 # # # # # #<br># # # # # # # #<br>Reserve the book (y/n)?<br>y<br>[content of “book.log”]<br>…<br>1 reserved 1245 |

| [Sample execution 4: the book is returned while performing “query_borrow”]<br>$ query_borrow 1245 584 |
|---|
| UnixProgramming 1<br>452 584 # # # # # #<br># # # # # # # #<br>Reserve the book (y/n)?<br>[at this time, book 1245 is returned. Book 1245 is not held by 452 now.]<br>y<br>[content of “book.log”]<br>…<br>584 borrowed 1245 |

[10] If others perform “borrow” operation, they will fail too and output “usrt_id failed to borrow book_idx” to book.log, no matter what status of book others got.

[Sample execution 5: before execution, userID_for_resv_request is full]
$ query_borrow 1245 17

[Sample ouput 5]
UnixProgramming 1
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16
Reserve the book (y/n)?
y
[content of "book.log"]
…
17 couldn't borrow 1245

[Sample execution 6: before execution, user 17 has already borrowed or reserved book 1245]
$ query_borrow 1245 17

[Sample ouput 6]
UnixProgramming 1
452 584 17 # # # # #
# # # # # # # #
Reserve the book (y/n)?
y
[content of "book.log"]
…
17 couldn't borrow 1245

[Sample execution 7: before execution, user 456 is querying book 1245 by "query_borrow" and has not made his/her decision yet]
$ query_borrow 1245 1

[Sample ouput 7]
UnixProgramming 1
452 584 # # # # #
# # # # # # # #
Reserve the book (y/n)?
y
[content of "book.log"]
…
1 failed to borrow 1245

[Sample execution 8]
$ query_borrow -1 1
$ query_borrow 1 -1
$ query_borrow -1 -1

[content of "book.log"]
…
invalid book index-1
invalid user ID -1
invalid user ID -1

[NAME]
*return*
[SYNOPSIS]
return book_idx
[DESCRIPTION]

To return books, a user executes program “return” with “book_idx” as command-line parameters. Book_idx is a nonnegative integer which is the index of book records in the “books”.

“Return” sets the first entry of userID_for_resv_request with NO_USER(-1) if success.

The program fails if the book is not held by anyone or the book_idx is not valid.

[OUTPUT FORMAT]
After every execution, “return” program must append the line “user_id returned book_idx” result to the “book.log” file[11], not the terminal.

| [Sample execution 1: before execution, book 154 is held by User 124] |
| --- |
| $ return 154 |
| [content of “book.log”] |
| … |
| 124 returned 154 |

| [Sample execution 2: book 154 is not held by any user] |
| --- |
| $ return 154 |
| [content of “book.log”] |
| … |
| 154 is not held by anyone |

| [Sample execution 3: there is no book 888 in the file] |
| --- |
| $ return 888 |
| [content of “book.log”] |
| … |
| invalid book index 888 |

## Submission

You should package all files into an archive file named by your student number and submit it to the CEIBA. There are at least six files in the archive file: Makefile, README.txt, query.c, borrow.c, query_borrow.c, and return.c. The Makefile states how to compile your source codes into four executable file, namely query , borrow, query_borrow, and return. All files including “library.h”, “books”, “Tx” and “book.log” are be stored in the working directory when we test your program.

```
$ mkdir STUDENT_ID
$ cp Makefile README.txt query.c borrow.c query_borrow.c return.c STUDENT_ID
$ tar -zcvf SPHW1_STUDENT_ID.tar.gz STUDNET_ID
#You may want to peek your tarball with this command
$ tar -ztvf SPHW1_STUDENT_ID.tar.gz
#Don’t use any other archive format such as .rar or .zip
```

To submit homework, upload your files to CEIBA.

## Grading

Plagiarism is prohibited. Your source codes will be checked by moss (http://theory.stanford.edu/~aiken/moss/).
For submission after deadline, 5% of credits will be deducted for every single day delay.
There are 3 subtasks in this assignment, you can get full 7 points by finishing them.

NOTE: you are responsible for following the requirements of the specification. You may receive no credit because of wrong filename and output format.

1. Compilation & execution (Point 2): compile the program by `make` and execute the four programs when each of the programs executes sequentially.

[11] sprintf(str, “%d %s %d”, user_id, “returned”, book_idx)

2. Interleaved `query, borrow` and `return` (2 Points): TAs will interleave the execution of the three programs and check the correctness of file “books‟ and file “book.log”.
3. Interleaved `query`, `borrow`, `query_borrow` and `return` (3 points): TAs will interleave the execution of the four programs check the correctness of file “books‟ and file “book.log”.