# Concurrency Control in Transactional Systems: Spring 2023
# **Programming Assignment 1: Implementing BOCC and FOCC algorithms**
## Submission Date: Schedule Given Below

**Goal:** The goal of this assignment is to implement BOCC and FOCC algorithms studied in the class. Implement both these optimistic algorithms in C++.

**Details.** As shown in the book, you have to implement both the optimistic concurrency algorithms **in C++**: BOCC & FOCC. Since, you are using optimistic concurrency control approach, all writes become visible only after commit. Thus on abort of a transaction, no rollback is necessary as none of the writes of the transactions will ever be visible.

You have to implement the following methods for both the algorithms:

- $begin\_trans()$: It begins a transactions and returns a unique transaction id, say $i$
- $read(i, x, l)$: Transaction $t_i$ reads data-item $x$ into the local value $l$.
- $write(i, x, l)$: Transaction $t_i$ writes to data-item $x$ with local value of $l$.
- $tryC(i)$: Transaction $t_i$ wants to commit. The return of this function is either $a$ for abort or $c$ for commit.

For FOCC algorithm, please implement the two variants: (1) the validating transaction gets aborted on detection of a conflict. Let us call this as *current-transaction-abort* or *CTA*. (2) the transactions conflicting with validating transaction gets aborted. We call this *other-transaction-abort* or *OTA*. We call these variants as: (1) FOCC-CTA (2) FOCC-OTA.

There are no such variants for BOCC. Hence, there is only variant for BOCC. Thus there are three variants to implement for both the algorithms.

To test the performance of both the algorithms, develop an application, opt-test is as follows. Once, the program starts, it creates $n$ threads and an array of $m$ shared variable. Each of these threads, will update the shared array randomly. Since the threads could simultaneously update the shared variables of the array, the access to shared variables have to be synchronized. This synchornization is performed using the above mentioned methods of BOCC, FOCC and the variants.

To pseudocode opt-test given is as follows explains the idea better:

Listing 1: main thread

```
1 void main()
2 {
3     ...
```

```
4       ...
5       // create a shared array of size m
6       shared[] = new SharedArray[m];
7       ...
8       ...
9       create n updtMem threads;
10  }
```

Listing 2: updtMem thread

```
1
2   void updtMem()
3   {
4       int status = abort;        // declare status variable
5       int abortCnt = 0;          // keeps track of abort count
6
7       long critStartTime, critEndTime;
8
9       // Each thread invokes numTrans transactions
10      for (int curTrans=0; curTrans<numTrans; curTrans++)
11      {
12          abortCnt = 0;                       // Reset the abort count
13          critStartTime = getSysTime();       // keep track of critical section start time
14
15          // getRand(k) function used in this loop generates a random number in the range 0 .. k
16          do
17          {
18              id = begin_trans(); // begins a new transaction id
19              randIters = getRand(m); // gets the number of iterations to be updated
20
21              int locVal;
22              for (int i=0; i<randIters; i++)
23              {
24                  // gets the next random index to be updated
25                  randInd = getRand(m);
26
27                  // gets a random value using the constant constVal
28                  randVal = getRand(constVal);
29
30                  // reads the shared value at index randInd into locVal
31                  read(id, shared[randInd], locVal);
32
33                  logFile << "Thread id " << pthread_self() << "Transaction " << id <<
34                  " reads from" << randInd << " a value " << locVal << " at time " <<
35                  getSysTime;
36
37                  // update the value
38                  locVal += randVal;
39
40                  // request to write back to the shared memory
41                  write(id, shared[randInd], locVal);
42
43                  logFile << "Thread id " << pthread_self() << "Transaction " << id <<
44                  " writes to " << randInd << " a value " << locVal << " at time " <<
45                  getSysTime;
46
```

```
47                  // sleep for a random amount of time which simulates some complex computation
48                  randTime = getExpRand(λ);
49                  sleep(randTime);
50              }
51
52              status = tryCommit(id);       // try to commit the transaction
53              logFile ≪ "Transaction " ≪ id ≪ " tryCommits with result "
54              ≪ status ≪ " at time " << getSysTime;
55              abortCnt++;                   // Increment the abort count
56          }
57          while (status! = commit);
58
59          critEndTime = getSysTime(); // keep track of critical section end time
60          record commitDelay & abortCnt;        // Record these values for collecting statistics
61      }   // End numTrans for
62  }
```

Here $randTime$ is an exponentially distributed with an average of $\lambda$ mill-seconds. The objective of having this time delays is to simulate that these threads are performing some complicated time consuming tasks. It can be seen that the time taken by a transaction to commit, *commitDelay* is defined as $critEndTime - critStartTime$.

**Input:** The input to the program will be a file, named inp-params.txt, consisting of all the parameters described above:$n, m, numTrans, constVal, \lambda$. A sample input file is: 10 10 50 100 20.

**Output:** Your program should output two files in the format given in the pseudocode for each algorithm: (1) BOCC-log.txt (2) FOCC-CTA-log.txt (3) FOCC-OTA-log.txt. A sample output is as follows:

The Output of your program (for any of the variants mentioned):
Thread 1 Transaction 1 reads 5 a value 0 at time 10:00
Thread 2 Transaction 2 reads 7 a value 0 at time 10:02
Thread 1 Transaction 1 writes 5 a value 15 at time 10:05
Thread 2 Transaction 2 tryCommits with result abort at time 10:10

.
.
.

The output is essentially a history. By inspecting the output one should be able to verify the serializability of your implementations.

**Report:** You have to submit a report for this assignment. This report should contain a comparison of the performance of the various variants of BOCC & FOCC. The comparison must consist of two graphs:

- A graph comparing the average time taken by a transaction to successfully commit, i.e. commitDelay in the variants of BOCC & FOCC. The x-axis should be the number of transactions varying from 500 to 1000 in the increments of 100; the y-axis is the average commitDelay of all the transactions for each of the algorithms. This graph will have four curves, one for each of the algorithms: (1) BOCC (2) FOCC-CTA (3) FOCC-OTA.
- A graph comparing the average abort count, i.e. the number of times a transaction abort before it can successfully commit . The x-axis should be the number of transactions varying from 500 to 1000 in the increments of 100; while the y-axis should be abort count.

Thus, this graph similar to the previous graph will have two curves: one representing BOCC and the other representing FOCC.

You must run both these algorithms multiple times to obtain performances values. You can vary have the number of threads fixed in all these experiments. Please have $m$ to be fixed at 10 in all these experiments. Finally in your report, you must also give an analysis of the results while explaining any anomalies observed.

**Deliverables:** You have to submit two kinds of deliverables:

**Pseudocode:** As discussed in the class, first you will have to submit the Pseudocode. The pseudocode of BOCC & FOCC. Name them as: BOCC-<rollno>.pcode, FOCC-CTA-<rollno>.pcode, FOCC-OTA-<rollno>.pcode.
Zip all the three files and name it as ProgAssn1_PCode-<rollno>.zip. Then upload it on the google classroom page of this course. Submit by the deadline shown below.

**Actual Code:** Next, you will have to submit the actual code. The details are as follows:

- The source files of BOCC & FOCC coded in C++. Name them as: BOCC-<rollno>.cpp, FOCC-CTA-<rollno>.cpp, FOCC-OTA-<rollno>.cpp.
- A readme.txt that explains how to execute the program.
- The report as explained above

Zip all the three files and name it as ProgAssn1_Code-<rollno>.zip. Then upload it on the google classroom page of this course by 22nd October 2021, 9:00 pm. In summary, the submission schedule is as follows:

1. **Pseudocode**: ~~6th March~~ 9th March 2023, 9:00 pm
2. **Actual running code in C++**: ~~13th March~~ 15th March 2023, 9:00 pm