CS 343 Fall 2017 – Assignment 4
Instructor: Peter Buhr
Due Date: Monday, November 6, 2017 at 22:00
Late Date: Wednesday, November 8, 2017 at 22:00

October 25, 2017

This assignment introduces complex locks in μC++ and continues examining synchronization and mutual exclusion. Use it to become familiar with these new facilities, and ensure you use these concepts in your assignment solution. **(Tasks may *not* have public members except for constructors and/or destructors.)**

1. Given the C++ program in Figure 1, compare buffering using internal-data versus external-data format. Redirect the program output to /dev/null to discard the output; otherwise, the output distorts the performance measurements.

   (a) Compare the two versions of the program with respect to performance by doing the following:

   - Run the program with and without the preprocessor option -DSTR.
   - Time the executions using the time command:

```
$ /usr/bin/time -f "%Uu %Ss %E" ./a.out > /dev/null  # ignore program output
3.21u 0.02s 0:03.32
```

```
#include <iostream>
using namespace std;

int main( int argc, char *argv[] ) {
    int times = 1000000, size = 40;                       // defaults

    try {
        switch ( argc ) {
          case 3: size = stoi( argv[2] ); if ( size <= 0 ) throw 1;
          case 2: times = stoi( argv[1] ); if ( times <= 0 ) throw 1;
          case 1: break;                                   // use defaults
          default: throw 1;
        } // switch
    } catch( ... ) {
        cout << "Usage: " << argv[0] << " [ times (> 0) [ size (> 0) ] ]" << endl;
        exit( 1 );
    } // try

    for ( int i = 0; i < times; i += 1 ) {
#ifndef STR
        int intbuf[size];                                  // internal-data buffer
        for ( int i = 0; i < size; i += 1 ) intbuf[i] = i;
        for ( int i = 0; i < size; i += 1 ) cout << intbuf[i] << ' '; // internal buffering
        cout << endl;
#else
        string strbuf;                                     // external-data buffer
        for ( int i = 0; i < size; i += 1 ) strbuf += to_string( i ) + ' '; // external buffering
        cout << strbuf << endl;
#endif
    } // for
}
```

Figure 1: Internal versus External Buffering

(Output from time differs depending on the shell, so use the system time command.) Compare the *user* time (3.21u) only, which is the CPU time consumed solely by the execution of user code (versus system and real time).

- Use the program command-line arguments (as necessary) to adjust program execution into the range 1 to 100 seconds. (Timing results below 1 second are inaccurate.) Use the same command-line values for all experiments, if possible; otherwise, increase/decrease the arguments as necessary and scale the difference in the answer.
- Run both the experiments again after recompiling the programs with compiler optimization turned on (i.e., compiler flag -O2).
- Include 4 timing results to validate the experiments.

(b) State the performance difference (larger/smaller/by how much) between the two versions of the program, and what caused the difference.

(c) State the performance difference (larger/smaller/by how much) between the original and transformed programs when compiler optimization is used.

(d) For interest, change endl to `'\n'` to see if there is any performance difference.

2. Consider the following situation involving a tour group of *V* tourists. The tourists arrive at the Louvre Museum for a tour. However, a tour can only be composed of *G* people at a time, otherwise the tourists cannot hear what the guide is saying. As well, there are 3 kinds of tours available at the Louvre: pictures, statues and gift shop. Therefore, each group of *G* tourists must vote among themselves to select the kind of tour to take. Voting is a *ranked ballot*, where each tourist ranks the 3 tours with values 0, 1, 2, where 2 is the highest rank. Tallying the votes sums the ranks for each kind of tour and selects the highest ranking. If tie votes occur among rankings, prioritize the results by gift shop, pictures, and then statues. During voting, tasks block until all votes are cast, i.e., assume a secret ballot. Once a decision is made, the tourists in that group proceed on the specified tour.

   To simplify the problem, the program only has to handle cases where the number of tourists is evenly divisible by the tour-group size so all groups contain the same number of tourists.

   Implement a vote-tallier for G-way voting as a class using *only*:

   a) an uOwnerLock and uCondLocks to provide mutual exclusion and synchronization (eliminate busy waiting using a signaller flag and extra condition lock).
   b) a single uBarrier to provide mutual exclusion and synchronization. Note, a uBarrier has implicit mutual exclusion so it is only necessary to manage the synchronization. As well, only the basic aspects of the uBarrier are needed to solve this problem.
   c) uSemaphores used as binary rather than counting to provide mutual exclusion and synchronization.

   No busy waiting is allowed and barging tasks can spoil an election and must be prevented. Figure 2 shows the different forms for each μC++ vote-tallier implementation (you may add only a public destructor and private members), where the preprocessor is used to conditionally compile a specific interface. This form of header file removes duplicate code. When the vote-tallier is created, it is passed the size of a group and a printer for printing state transitions. There is only one vote-tallying object created for all of the voters, who share a reference to it. Each tourist task calls the vote method with their id and a ranked vote, indicating their desire for a picture, statue, or gift-shop tour. The vote routine does not return until group votes are cast; after which, the majority result of the voting (Picture, Statue or GiftShop) is returned to each voter. The groups are formed based on voter arrival; e.g., for a group of 3, if voters 2, 5, 8 cast their votes first, they form the first group, etc. Hence, all voting is serialized. An appropriate preprocessor variable is defined on the compilation command using the following syntax:

   ```
   u++ -DIMPLTYPE_SEM -c TallyVotesSEM.cc
   ```

   The interface for the voting task is (you may add only a public destructor and private members):

```
#if defined( IMPLTYPE_MC )          // mutex/condition solution
// includes for this kind of vote-tallier
class TallyVotes {
    // private declarations for this kind of vote-tallier
#elif defined( IMPLTYPE_BAR )       // barrier solution
// includes for this kind of vote-tallier
_Cormonitor TallyVotes : public uBarrier {
    // private declarations for this kind of vote-tallier
#elif defined( IMPLTYPE_SEM )       // semaphore solution
// includes for this kind of vote-tallier
class TallyVotes {
    // private declarations for this kind of vote-tallier
#else
    #error unsupported voter type
#endif
    // common declarations
  public:                           // common interface
    TallyVotes( unsigned int group, Printer & printer );
    struct Ballot { unsigned int picture, statue, giftshop; };
    enum Tour { Picture = 'p', Statue = 's', GiftShop = 'g' };
    Tour vote( unsigned int id, Ballot ballot );
};
```

Figure 2: Tally Voter Interfaces

```
_Task Voter {
    // Choose ranking of picture tour, then relationship of statue to gift shop.
    TallyVotes::Ballot cast() {         // cast 3-way vote
        static unsigned int voting[3][2][2] = { { {2,1}, {1,2} }, { {0,2}, {2,0} }, { {0,1}, {1,0} } };
        unsigned int picture = mprng( 2 ), statue = mprng( 1 );
        return (TallyVotes::Ballot){ picture, voting[picture][statue][0], voting[picture][statue][1] };
    }
  public:
    enum States { Start = 'S', Vote = 'V', Block = 'B', Unblock = 'U', Barging = 'b',
                  Complete = 'C', Finished = 'F' };
    Voter( unsigned int id, TallyVotes & voteTallier, Printer & printer );
};
```

The task main of a voting task looks like:

- yield a random number of times, between 0 and 19 inclusive, so all tasks do not start simultaneously
- print start message
- yield once
- vote
- yield once
- print finish message

Note that each task votes only once. Casting a vote is accomplished by calling cast. Yielding is accomplished by calling yield(times) to give up a task's CPU time-slice a number of times.

All output from the program is generated by calls to a printer, excluding error messages. The interface for the printer is (you may add only a public destructor and private members).

```
_Monitor / _Cormonitor Printer {    // chose one of the two kinds of type constructor
  public:
    Printer( unsigned int voters );
    void print( unsigned int id, Voter::States state );
    void print( unsigned int id, Voter::States state, TallyVotes::Tour tour );
    void print( unsigned int id, Voter::States state, TallyVotes::Ballot ballot );
    void print( unsigned int id, Voter::States state, unsigned int numBlocked );
};
```

(Monitors are discussed shortly, and are classes with public methods that implicitly provide mutual exclusion.) The printer attempts to reduce output by storing information for each voter until one of the stored elements is

```
 1  $ vote 3 1 103
 2  V0      V1      V2
 3  ******* ******* *******
 4  S               S
 5  V 0,2,1
 6  C               V 0,2,1
 7  F s     S       C
 8          V 1,0,2 F s
 9          C
10          F g
11  *****************
12  All tours started
```

```
$ vote 6 3 16219
V0      V1      V2      V3      V4      V5
******* ******* ******* ******* ******* *******
                                S
                                V 0,2,1
                        S       B 1
                        V 1,2,0
S                       B 2             S
V 2,0,1
C                               U 1
F s     S       S       U 0     F s     b
                b       F s             V 1,2,0
        b       V 0,2,1                 B 1
        V 2,1,0 B 2
        C                               U 1
        F s     U 0                     F s
                F s
*****************
All tours started
```

Figure 3: Voters: Example Output

overwritten. When information is going to be overwritten, all the stored information is flushed and storing starts again. Output must look like that in Figure 3.

Each column is assigned to a voter with an appropriate title, “V$_i$”, and a column entry indicates its current status:

State	Meaning
S	starting
V p,s,g	voting with ballot containing 3 rankings
B n	blocking during voting, n voters waiting (including self)
U n	unblocking after group reached, n voters still waiting (not including self)
b	barging into voter and having to wait for signalled tasks
C	group is complete and voting result is computed
F t	finished voting and selected tour is t (p/s/g)

Information is buffered until a column is overwritten for a particular entry, which causes the buffered data to be flushed. If there is no new stored information for a column since the last buffer flush, an empty column is printed. After a task has finished, no further output appears in that column. All output spacing can be accomplished using the standard 8-space tabbing. Buffer any information necessary for printing in its internal representation; **do not build and store strings of text for output**. Calls to perform printing may be performed from the vote-tallier and/or a voter task (you decide where to print).

For example, in line 4 of the left hand example in Figure 3, V0 has the value “S” in its buffer slot, V1 is empty, and V2 has value “S”. When V0 attempts to print “V 0,2,1”, which overwrites its current buffer value of “S”, the buffer must be flushed generating line 4. V0’s new value of “V 0,2,1” is then inserted into its buffer slot. When V0 attempts to print “C”, which overwrites its current buffer value of “V 0,2,1”, the buffer must be flushed generating line 5, and no other values are printed on the line because the print is consecutive (i.e., no intervening call from another object). Then V0 inserts value “C” and V2 inserts value “V 0,2,1” into the buffer. When V2 attempts to print “C”, which overwrites its current buffer value of “V 0,2,1”, the buffer must be flushed generating line 6, and so on. Note, a group size of 1 means a voter never has to block/unblock.

The executable program is named vote and has the following shell interface:

```
vote [ V [ G [ Seed ] ] ]
```

(Square brackets indicate optional command line parameters, and do not appear on the actual command line.) V is the size of a tour, i.e., the number of voters (tasks) to be started (multiple of G); if V is not present, assume a value of 6. G is the size of a tour group (odd number); if G is not present, assume a value of 3. Seed is the seed for the random-number generator and must be greater than 0. If the seed is unspecified, use a random value like the process identifier (getpid) or current time (time), so each run of the program generates different output. Use the monitor MPRNG to safely generate random values (monitors will be discussed shortly). Note, because of the non-deterministic execution of concurrent programs, multiple runs with a common seed may not generate the same output. Nevertheless, shorts runs are often the same so the seed can be useful for testing. Check all command arguments for correct form (integers) and range; print an appropriate usage message and terminate the program if a value is missing or invalid.

Submission Guidelines

Please follow these guidelines carefully. Review the Assignment Guidelines and C++ Coding Guidelines *before* starting each assignment. **Each text file, i.e., *.*txt file, must be ASCII text and not exceed 500 lines in length, where a line is a maximum of 120 characters.** Programs should be divided into separate compilation units, i.e., *.{h,cc,C,cpp} files, where applicable. Use the submit command to electronically copy the following files to the course account.

1. q1*.txt – contains the information required by question 1, p. 1.
2. MPRNG.h – random number generator (provided)
3. q2tallyVotes.h, q2*.{h,cc,C,cpp} – code for question question 2, p. 2. **Program documentation must be present in your submitted code. No user or system documentation is to be submitted for this question.**
4. q2*.testdoc – test documentation for question 2, p. 2, which includes the input and output of your tests. **Poor documentation of how and/or what is tested can results in a loss of all marks allocated to testing.**
5. Modify the following Makefile to compile the programs for questions 1, p. 1 and 2, p. 2 by inserting the object-file names matching your source-file names.

```
TYPE:=MC
OPT:=-O2

CXX = u++                                        # compiler
CXXFLAGS = -g -Wall ${OPT} -multi -MMD -std=c++11 -DIMPLTYPE_${TYPE} # compiler flags
MAKEFILE_NAME = ${firstword ${MAKEFILE_LIST}} # makefile name

OBJECTS2 = q2tallyVotes${TYPE}.o # list of object files for question 1 prefixed with "q2"
EXEC2 = vote

OBJECTS = ${OBJECTS2}                            # all object files
DEPENDS = ${OBJECTS:.o=.d}                       # substitute ".o" with ".d"
EXECS = ${EXEC2}                                 # all executables

#############################################################

.PHONY : all clean

all : ${EXECS}                                   # build all executables

-include ImplType

ifeq (${IMPLTYPE},${TYPE})                       # same implementation type as last time ?
${EXEC2} : ${OBJECTS2}
	${CXX} ${CXXFLAGS} $^ -o $@
else
ifeq (${TYPE},)                                  # no implementation type specified ?
# set type to previous type
TYPE=${IMPLTYPE}
${EXEC2} : ${OBJECTS2}
	${CXX} ${CXXFLAGS} $^ -o $@
else                                             # implementation type has changed
```

```
.PHONY : ${EXEC2}
${EXEC2} :
    rm -f ImplType
    touch q2tallyVotes.h
    sleep 1
    ${MAKE} ${EXEC2} TYPE="${TYPE}"
endif
endif

ImplType :
    echo "IMPLTYPE=${TYPE}" > ImplType
    sleep 1

#############################################################

${OBJECTS} : ${MAKEFILE_NAME}               # OPTIONAL : changes to this file => recompile

-include ${DEPENDS}                         # include *.d files containing program dependences

clean :                                     # remove files that can be regenerated
    rm -f *.d *.o ${EXECS} ImplType
```

This makefile is invoked as follows:

```
$ make vote TYPE=MC
$ vote ...
$ make vote TYPE=SEM
$ vote ...
$ make vote TYPE=BAR
```

Put this Makefile in the directory with the programs, name the source files as specified above, and enter the appropriate make to compile a specific version of the programs. This Makefile must be submitted with the assignment to build the program, so it must be correct. Use the web tool Request Test Compilation to ensure you have submitted the appropriate files, your makefile is correct, and your code compiles in the testing environment.

Follow these guidelines. Your grade depends on it!