Contents

1 Basic Test Results

```
============== Tar Content Test ==============
found README
found Makefile
tar content test PASSED!

============== logins ==============
login names mentioned in file:  kawabanga,michal.maayan
Please make sure that these are the correct login names.

============== make Command Test ==============
g++ -Wall -std=c++11 -g  -I.   -c -o MapReduceFramework.o MapReduceFramework.cpp
g++ -Wall -std=c++11 -g  -I.   -c -o Barrier.o Barrier.cpp
ar rv libMapReduceFramework.a MapReduceFramework.o Barrier.o Barrier.h
a - MapReduceFramework.o
a - Barrier.o
a - Barrier.h
ranlib libMapReduceFramework.a

MapReduceFramework.cpp: In function void* threadLogic(void*):
MapReduceFramework.cpp:111:21: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
     while (oldValue < tc->inputVec->size()) {
             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MapReduceFramework.cpp:131:41: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
             for (int i = FIRSTTHREAD; i < tc->MT; ++i) {
                                       ~~~~~~~~~~
MapReduceFramework.cpp:159:37: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
         for (int i = FIRSTTHREAD; i < tc->MT; ++i) {
                                     ~~~~~~~~~~

MapReduceFramework.cpp:172:19: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
         if (index < tc->Queue->size()) {
             ~~~~~~~~~~~~~~~~~~~~~~~~~~
MapReduceFramework.cpp: In function void runMapReduceFramework(const MapReduceClient&, const InputVec&, OutputVec&, int):
MapReduceFramework.cpp:218:55: warning: narrowing conversion of i from int to unsigned int inside { } [-Wnarrowing]
                        &Queue, &mutexQueue, &fillCount};
                                                       ^
ar: creating libMapReduceFramework.a

stderr: b'MapReduceFramework.cpp: In function \xe2\x80\x98void* threadLogic(void*)\xe2\x80\x99:\nMapReduceFramework.cpp:111:
make command test FAILED!

Pre-submission Test FAILED!
Check info above.
```

2 README

```
kawabanga, michal.maayan
Yaron Scherf (305758211), Michal Maayan (203348776)
EX: 3

FILES:
MapReduceFramework.c -- Our main code file, an implementation of the
                        required code.
Barrier.h -- school implementation
Barrier.cpp -- school implementation

REMARKS:
Hope you have enjoyed from our implementation, we have done our best :)
```

3 Barrier.h

```
#ifndef BARRIER_H
#define BARRIER_H
#include <pthread.h>

// a multiple use barrier

class Barrier {
public:
    Barrier(int numThreads);
    ~Barrier();
    void barrier();

private:
    pthread_mutex_t mutex;
    pthread_cond_t cv;
    int count;
    int numThreads;
};

#endif //BARRIER_H
```

4 Barrier.cpp

```cpp
#include "Barrier.h"
#include <cstdlib>
#include <cstdio>

Barrier::Barrier(int numThreads)
 : mutex(PTHREAD_MUTEX_INITIALIZER)
 , cv(PTHREAD_COND_INITIALIZER)
 , count(0)
 , numThreads(numThreads)
{ }


Barrier::~Barrier()
{
    if (pthread_mutex_destroy(&mutex) != 0) {
        fprintf(stderr, "[[Barrier]] error on pthread_mutex_destroy");
        exit(1);
    }
    if (pthread_cond_destroy(&cv) != 0){
        fprintf(stderr, "[[Barrier]] error on pthread_cond_destroy");
        exit(1);
    }
}


void Barrier::barrier()
{
    if (pthread_mutex_lock(&mutex) != 0){
        fprintf(stderr, "[[Barrier]] error on pthread_mutex_lock");
        exit(1);
    }
    if (++count < numThreads) {
        if (pthread_cond_wait(&cv, &mutex) != 0){
            fprintf(stderr, "[[Barrier]] error on pthread_cond_wait");
            exit(1);
        }
    } else {
        count = 0;
        if (pthread_cond_broadcast(&cv) != 0) {
            fprintf(stderr, "[[Barrier]] error on pthread_cond_broadcast");
            exit(1);
        }
    }
    if (pthread_mutex_unlock(&mutex) != 0) {
        fprintf(stderr, "[[Barrier]] error on pthread_mutex_unlock");
        exit(1);
    }
}
```

5 Makefile

```
CC=g++
CXX=g++
RANLIB=ranlib

LIBSRC=MapReduceFramework.cpp Barrier.cpp Barrier.h
LIBOBJ=$(LIBSRC:.cpp=.o)

INCS=-I.
CFLAGS = -Wall -std=c++11 -g $(INCS)
CXXFLAGS = -Wall -std=c++11 -g  $(INCS)

OSMLIB = libMapReduceFramework.a
TARGETS = $(OSMLIB)

TAR=tar
TARFLAGS=-cvf
TARNAME=ex3.tar
TARSRCS=$(LIBSRC) Makefile README

all: $(TARGETS)

$(TARGETS): $(LIBOBJ)
	$(AR) $(ARFLAGS) $@ $^
	$(RANLIB) $@

clean:
	$(RM) $(TARGETS) $(OSMLIB) $(OBJ) $(LIBOBJ) *~ *core

depend:
	makedepend -- $(CFLAGS) -- $(SRC) $(LIBSRC)

tar:
	$(TAR) $(TARFLAGS) $(TARNAME) $(TARSRCS)
```

6 MapReduceFramework.cpp

```cpp
#include <cstdlib>
#include <cstdio>
#include "Barrier.h"
#include "MapReduceClient.h"
#include <atomic>
#include <iostream>
#include <algorithm>    // std::sort
#include <semaphore.h>

#define FIRSTTHREAD 0

typedef struct ThreadContext{
    unsigned int threadId;
    unsigned int MT;
    Barrier * barrier;
    std::atomic<int>* atomicIndex;
    std::atomic<int> *outAtomicIndex;
    std::atomic<int> *reduceAtomic;
    const InputVec* inputVec;
    OutputVec* outputVec;
    std::vector<IntermediateVec> *arrayOfInterVec;
    const MapReduceClient* client;
    std::vector <IntermediateVec> *Queue;
    sem_t *mutexQueue;
    sem_t *fillCount;
}ThreadContext;

typedef struct MapContext{
    IntermediateVec *interVector;
}MapContext;

typedef struct ReduceContext{
    OutputVec *outVector;
    std::atomic<int> *outAtomicIndex;
}ReduceContext;

void safeExit(ThreadContext* tc)
{
    if(tc == nullptr){
        exit(-1);
    }
    tc->Queue->clear();
}

void printErr(std::string msg, ThreadContext *tc){
    std::cerr<<msg<<std::endl;
    safeExit(tc);
}

//for debug
//void printInterVector(IntermediateVec** array, int numOfThreads){
//    printf("++++++++++++\n");
//    for (int i = 1; i < numOfThreads; ++i){
//        printf("thread id: %d:\n",i);
//        for (unsigned long j = 0; j < (*(array[i])).size() ; ++j){
//            //printIntermediatePair((&(*(array[i])).at(j)));
//        }
//        printf("~~~~~~~~\n");
//    }
```

```
//}

void emit2 (K2* key, V2* value, void* context){
    auto* tc = (MapContext*) context;
    tc->interVector->emplace_back(key, value);
}
void emit3 (K3* key, V3* value, void* context){
    auto* tc = (ReduceContext*) context;
    int oldValue = (*(tc->outAtomicIndex))++ ;
    auto iterator = (*(tc->outVector)).begin();
    (*(tc->outVector)).emplace(iterator+(oldValue), key, value);
}

bool comperator(IntermediatePair &p1, IntermediatePair &p2){
    return (*p1.first < *p2.first);
}

// check the IntermediateVec isn't empty, in case it doesn't find the next max key
bool check_empty_find_max(std::vector<IntermediateVec> *arr, int MT, K2 **max){
    bool isEmpty = true;
    for (int i = FIRSTTHREAD; i < MT; ++i) {
        if (not((arr)->at(i)).empty()) {
            isEmpty = false;
            if(*max == nullptr){
                *max = ((arr)->at(i)).back().first;
            }
            else{
                if((**max) < *((arr)->at(i)).back().first)
                {
                    *max = ((arr)->at(i)).back().first;
                }
            }
        }
    }
    return isEmpty;
}

bool is_eq(K2 *max, IntermediatePair &p){
    if(not(*max < *p.first)){
        if(not(*p.first < *max)){
            return true;
        }
    }
    return false;
}

void* threadLogic (void* context) {
    auto *tc = (ThreadContext *) context;
    int oldValue = (*(tc->atomicIndex))++;

    //map logic
    while (oldValue < tc->inputVec->size()) {
        auto k1 = tc->inputVec->at(oldValue).first;
        auto v1 = tc->inputVec->at(oldValue).second;
        MapContext mapContext = {(&tc->arrayOfInterVec->at(tc->threadId))};
        tc->client->map(k1, v1, &mapContext);
        oldValue = (*(tc->atomicIndex))++;
    }

    //sort
    auto tempVec = &(tc->arrayOfInterVec->at(tc->threadId));//[tc->threadId];
    std::sort(tempVec->begin(), tempVec->end(), comperator);
    tc->barrier->barrier();

    //shuffle
    if(tc->threadId == FIRSTTHREAD) {
        // initial K2
        K2* max = nullptr;
```

```
128         bool isEmpty = check_empty_find_max(tc->arrayOfInterVec, tc->MT, &max);
129         while(not isEmpty){
130             IntermediateVec sameKey = {};
131             for (int i = FIRSTTHREAD; i < tc->MT; ++i) {
132                 // in case the vector isn't empty
133                 if (not(tc->arrayOfInterVec->at(i).empty())) {
134                     while(is_eq(max, (tc->arrayOfInterVec->at(i).back()))) {
135                         (sameKey).emplace_back(((tc->arrayOfInterVec->at(i))).back());
136                         ((tc->arrayOfInterVec->at(i))).pop_back();
137                         //check emptyness again
138                         if (((tc->arrayOfInterVec->at(i))).empty()){
139                             break;
140                         }
141                     }
142                  }
143             }
144             max = nullptr;
145             isEmpty = check_empty_find_max(tc->arrayOfInterVec, tc->MT, &max);
146             //add to queue using semaphore
147             if (sem_wait(tc->mutexQueue) != 0){
148                 printErr("sem_wait err",tc);
149             }
150             tc->Queue->push_back(sameKey);
151             if (sem_post(tc->mutexQueue) != 0){
152                 printErr("sem_post err",tc);
153             }
154             if (sem_post(tc->fillCount) != 0){
155                 printErr("sem_post err",tc);
156             }
157         }
158         //wakeup all the threads who went down
159         for (int i = FIRSTTHREAD; i < tc->MT; ++i) {
160             if (sem_post(tc->fillCount) != 0){
161                 printErr("sem_post err",tc);
162             }
163         }
164     }// end of shuffle
165
166     //reduce
167     while(true){
168         if (sem_wait(tc->fillCount) != 0){
169             printErr("sem_wait err",tc);
170         }
171         auto index = (*(tc->reduceAtomic))++ ;
172         if (index < tc->Queue->size()) {
173             if (sem_wait(tc->mutexQueue) != 0){
174                 printErr("sem_wait err",tc);
175             }
176             auto pairs = &(tc->Queue->at(index));
177             ReduceContext reduceContext = {tc->outputVec, tc->outAtomicIndex};
178             tc->client->reduce(pairs, &reduceContext);
179             if (sem_post(tc->mutexQueue) != 0){
180                 printErr("sem_post err",tc);
181             }
182         } else{
183             break;
184         }
185     }
186     return nullptr;
187 }
188
189
190 void runMapReduceFramework(const MapReduceClient& client,
191                            const InputVec& inputVec, OutputVec& outputVec,
192                            int multiThreadLevel){
193     if(multiThreadLevel < 1){
194         printErr("multiThreadLevel isn't legal", nullptr);
195     }
```

```
196     pthread_t threads[multiThreadLevel];
197     ThreadContext contexts[multiThreadLevel];
198     Barrier barrier(multiThreadLevel);
199     std::atomic<int> atomicIndex(0);
200     std::atomic<int> outAtomicIndex(0);
201     std::atomic<int> reducetAtomic(0);
202     std::vector<IntermediateVec> arrayOfInterVec = {};
203     std::vector <IntermediateVec> Queue;
204     sem_t mutexQueue;
205     sem_t  fillCount;
206     if (sem_init(&mutexQueue, 0, 1) != 0){
207         printErr("sem_init failed\n", nullptr);
208     }
209     if (sem_init(&fillCount, 0, 0) != 0){
210         printErr("sem_init failed\n", nullptr);
211     }
212     for (int i = FIRSTTHREAD; i < multiThreadLevel; ++i) {
213         arrayOfInterVec.push_back({});
214     }
215     for (int i = FIRSTTHREAD; i < multiThreadLevel; ++i) {
216         contexts[i] = {i, (unsigned int)multiThreadLevel, &barrier, &atomicIndex, &outAtomicIndex, &reducetAtomic,
217                        &inputVec, &outputVec, &arrayOfInterVec, &client,
218                        &Queue, &mutexQueue, &fillCount};
219     }
220     for (int i = FIRSTTHREAD+1; i < multiThreadLevel; ++i) {
221         if (pthread_create(threads + i, NULL, threadLogic, contexts + i) != 0){
222             printErr("pthread_create failed\n",contexts + i);
223         }
224     }
225     threadLogic(contexts);
226     for (int i = FIRSTTHREAD+1; i < multiThreadLevel; ++i) {
227         if (pthread_join(threads[i], NULL) != 0){
228             printErr("pthread_join failed\n",contexts + i);
229         }
230     }
231
232 }
```