```
1  //james roesemann
2  //CSCi 375
3  //Project #2
4  //Due 7/10/18
5
6  //!!!!This program does not work!!!!
7
8
9
10 //Despite my efforts, i wwas not able to coreclty implement this program.
11 //under this implementation, readers and writers are frequently in the critical
   section together.
12 //I never got around to implementing any shared buffer
13 //I think i'm doing something wrong with the semaphores? but i'm not sure.
14 //all in all i'm very disapointed.
15
16
17 /*
18 program goals:
19 -up to two reader processes can be inside the critical section without any
   writer processes.
20 -inorder for a writer process to go into the critical section, it must first
   check wether or there is a process int the critical section.
21
22 -the critical section reads shared data buffer form the reader. updates shared
   data buffer for the wrriter process.
23 i must implemnt any shared data for readers/writers but you must clearly specify
   the following in output.
24
25     -when reader or writer enters the critical section, it must report whetehер
   there are any readers/reiters other than itself.
26     -(optional) may print out the data you read write when implementing the real
   buffer.
27     -print out a "panic message" when the semaphore rules are not observed.
28
29
30 You need to keep track of program counter of each process to resume at the right
   place once it will be chosen to run by keeping global counter variable per
   process. (**i don;t entirly understand this***).
31
32
33 */
34
35
36
37
38
39 #include <iostream>
40 #include <cstdlib>
41 #include <ctime>
42 #include <cmath>
43 #include <thread>
44 using namespace std;
45
46
47
48
49 class countingSemaphore{
50         private:
51                 bool keyIsAvailable;
52                 int availableTime;
53         public:
```

```
54                  //default construction is to set lock position to 0. i think
    over time this needs to increase somehow? probably goes back to 0 once it
    finishes running. i think.
55                  countingSemaphore(){
56                          keyIsAvailable=true;
57                          availableTime=100;
58                  }
59                  bool getKeyIsAvailable(){return keyIsAvailable;}
60                  void setKeyIsAvailable(bool x){keyIsAvailable=x;}
61                  int getAvailableTime(){return availableTime;}
62                  void decreaseAvailableTime(){availableTime--;}
63                  // reset the availableTime to 100
64                  void resetAvailableTime(){availableTime=100;}
65
66  };
67
68  //simpler to implement. start here.
69  class binarySemaphore{
70          private:
71                  bool keyIsAvailable;
72
73          public:
74                  binarySemaphore(){keyIsAvailable=true;}
75                  bool getKeyIsAvailable(){return keyIsAvailable;}
76                  void setKeyIsAvailable(bool x){keyIsAvailable=x;}
77  };
78
79
80  class processObject{
81          private:
82                  //process type of 0=reader. 1=writer
83                  int processType;
84                  bool hasKey;
85          public:
86                  //constructor
87                  processObject(int x){
88                          if(x !=0 && x != 1){
89                                  //error message. look up how to do this properly
    again.
90                                  std::cout << "You have entered a process number
    that does not match the available range of 0 to 1.\n";
91                          }
92                          else{processType=x;}
93                          hasKey=false;
94                  }
95                  //not puting a set for processType, should not change once
    created.
96                  int getProcessType(){return processType;}
97                  bool getHasKey(){return hasKey;}
98                  void setHasKey(bool x){hasKey=x;}
99
100 };
101
102
103 //reports if any writers in the critical section
104 void checkCriticalSection(bool writerKey);
105
106 //reports if any readers are in the critical section
107 void checkCriticalSection(bool readerKeys[]);
108
109 //reports if any writers or readers are in the critical section
110 void checkCriticalSection(bool readerKeys[], bool writerKey);
```

```
111
112 //subwait is used to implement the counting semaphore. while availbleTime>0,
    checks if keyIsAvailable from the passedSemaphore = true. if it is returns true.
    if time has expired it resets availbleTime and returns false.
113 bool subWait(countingSemaphore * passedSemaphore);
114
115 //countingSemaphore wait. checks if there any available counting semaphores
    (where available time =100). if not it returns false to allow another process to
    enter the critical section. if there are it calls subWait for that semaphore.
    returns the result.
116 bool wait( countingSemaphore passedSemaphores[]);
117
118 //binarySemaphore wait. while the semaphores keyIsAvailable is false, break to
    allow another process to enter critical section. if it is true. change it to
    false, and allow the process to continue.
119 bool wait( binarySemaphore * passedSemaphore);
120
121 //signal for the binarySemaphore, changes the value of keyIsAvailable
122 void signal(binarySemaphore * passedSemaphore);
123
124 //signal for the countingSemaphore. compares the availableTime of both
    semaphores. changes the value of keyIsAvailable for the one that is lowest. (the
    one that has been waiting the longest)
125 //i might run itno a problem here. its an array of pointers os maybe it will
    change it, but i'm not entirely sure.
126 void signal(countingSemaphore passedSemaphore[]);
127
128 //checks to see if there are any writers in the critical section(for reader
    processes) if there are any in the critical section, return true. true causes
    the case to break. does not wait and does not change any variables.
129 //i feel like this should be modified. maybe allow it to wait endlessly untill
    it can enter? probably a bad idea
130 bool checkForWriters(binarySemaphore * writerSemaphores);
131
132 //checks to see if there are any reader in the critical section(for writer
    processes) if there are any in the critical section, return true. true causes
    the case to break. does not wait and does not change any variables
133 bool checkForReaders(countingSemaphore  readerSemaphores[]);
134
135 //passed the readerKeys array and processID(the current case number).checks
    which readerKey is available to be locked and locks it.
136 //i don't like how i implemented this. i have to check which key is available
    first before entering swaping it. i know that can lead to some problems but i
    don't know how to get around it. may need to rewrite this.
137 void lock(processObject * processId, bool readerKeys[]);
138
139 //passed the writerKey and processId( the current case number). locks the
    binarySemaphore
140 void lock(bool &writerKey,processObject * processId);
141
142 //passed the readerKeys array and processID(the current case number). unlocks
    the first available spot in the array
143 void unlock(processObject * processId, bool readerKeys[]);
144
145 //passed the writerKey. unlocks it
146 void unlock(bool &writerKey,processObject * processId);
147
148
149
150
151
152 int main(){
```

```
153
154         //if array of the available processes. if this array were any larger i
    would have probably written a function to assign assign them.
155         processObject *avaiableProcesses[5]={new processObject(0), new
    processObject(0), new processObject(0), new processObject(1), new processObject
    (1)};
156
157         bool *readerKeys[2]{new bool(true), new bool(true)};
158         bool writerKey=true;
159         binarySemaphore writerLock;
160         countingSemaphore *readerLock[2]={new countingSemaphore(), new
    countingSemaphore()};
161
162         //random intilization
163         srand(time(NULL));
164
165         //for testing purposes this programs end after 20 "processes" have been
    called.
166         int maxProcesses=20;
167         int processCount=0;
168
169         while(processCount<maxProcesses){
170                 //case 0-2 represent reader processes. csse 3-4 represent writer
    processes.
171                 switch(rand()%5){
172                         case 0:
173                                 //wait untill you can enter the critical
    section. if no available semaphores or time expires, break
174                                 if(wait(*readerLock)==false){break;}
175                                 //check to see if thhere is a writer process in
    the critical section. if there is, signal the countingSemaphres, break and
    return to ready state to allow another process to enter the critical section.
176                                 if(checkForWriters(& writerLock)==true){
177                                         signal(*readerLock);
178                                         break;
179                                 }
180                                 //lock the critical section. position on
    availableProcesses array is the came ase case number
181                                 lock(avaiableProcesses[0], *readerKeys);
182                                 //report current processes report processes
    curently in critical section
183                                 std::cout << "reader case 0: entering critical
    section\n";
184                                 checkCriticalSection(*readerKeys, writerKey);
185                                 //***critical section***
186                                 while(avaiableProcesses[0]->getHasKey()==true){
187                                         //optional, do somthing with a shared
    buffer here
188                                         //***end critical section***
189                                         //unlock critical section
190                                         unlock(avaiableProcesses[0],
    *readerKeys);
191                                 }
192                                 //signal semaphore
193                                 signal(*readerLock);
194                                 break;
195                         case 1:
196                                 //wait untill you can enter the critical
    section. if no available semaphores or time expires, break
197                                 if(wait(*readerLock)==false){break;}
198                                 //check to see if thhere is a writer process in
    the critical section. if there is, signal the countingSemaphres, break and
```

```
    return to ready state to allow another process to enter the critical section.
199                                 if(checkForWriters(& writerLock)==true){
200                                         signal(*readerLock);
201                                         break;
202                                 }
203                                 //lock the critical section. position on
    availableProcesses array is the came ase case number
204                                 lock(avaiableProcesses[1], *readerKeys);
205                                 //report current processes report processes
    curently in critical section
206                                 std::cout << "reader case 1: entering critical
    section\n";
207                                 checkCriticalSection(*readerKeys, writerKey);
208                                 //***critical section***
209                                 while(avaiableProcesses[1]->getHasKey()==true){
210                                         //optional, do somthing with a shared
    buffer here
211                                         //***end critical section***
212                                         //unlock critical section
213                                         unlock(avaiableProcesses[1],
    *readerKeys);
214                                 }
215                                 //signal semaphore
216                                 signal(*readerLock);
217                                 break;
218                         case 2:
219                                 //wait untill you can enter the critical
    section. if no available semaphores or time expires, break
220                                 if(wait(*readerLock)==false){break;}
221                                 //check to see if thhere is a writer process in
    the critical section. if there is, signal the countingSemaphres, break and
    return to ready state to allow another process to enter the critical section.
222                                 if(checkForWriters(& writerLock)==true){
223                                         signal(*readerLock);
224                                         break;
225                                 }
226                                 //lock the critical section. position on
    availableProcesses array is the came ase case number
227                                 lock(avaiableProcesses[2], *readerKeys);
228                                 //report current processes report processes
    curently in critical section
229                                 std::cout << "reader case 2: entering critical
    section\n";
230                                 checkCriticalSection(*readerKeys, writerKey);
231                                 //***critical section***
232                                 while(avaiableProcesses[2]->getHasKey()==true){
233                                         //optional, do somthing with a shared
    buffer here
234                                         //***end critical section***
235                                         //unlock critical section
236                                         unlock(avaiableProcesses[2],
    *readerKeys);
237                                 }
238                                 //signal semaphore
239                                 signal(*readerLock);
240                                 break;
241                         case 3:
242                                 //check to see if there are any readers in the
    critical section. if so, break. if not continue.
243                                 if(checkForWriters( &writerLock)==true){break;}
244                                 //if another writer is in the critical section,
    break. if not, set keyIsAvailable to false and continue
```

```
245                                     if(wait(&writerLock)==false){break;}
246
247                                     //lock the critical section
248                                     lock(writerKey, avaiableProcesses[3]);
249                                     //report on what is in the critical section
    before entering.
250                                     std::cout << "writer case 3: entering critical
    section\n";
251                                     checkCriticalSection(*readerKeys, writerKey);
252                                     ///***critical section ***
253                                     while(avaiableProcesses[3]->getHasKey()==true){
254                                     //do something with the buffer(optional)
255
256
257                                     //***end critical section***
258                                     //unlock
259                                             unlock(writerKey, avaiableProcesses[3]);
260                                     //signal
261                                     signal(&writerLock);
262                                     }
263                                     break;
264                             case 4:
265                                     //check to see if there are any readers in the
    critical section. if so, break. if not continue.
266                                     if(checkForWriters( &writerLock)==true){break;}
267                                     //if another writer is in the critical section,
    break. if not, set keyIsAvailable to false and continue
268                                     if(wait(&writerLock)==false){break;}
269
270                                     //lock the critical section
271                                     lock(writerKey, avaiableProcesses[4]);
272                                     //report on what is in the critical section
    before entering.
273                                     std::cout << "writer case 4: entering critical
    section\n";
274                                     checkCriticalSection(*readerKeys, writerKey);
275                                     ///***critical section ***
276                                     while(avaiableProcesses[4]->getHasKey()==true){
277                                     //do something with the buffer(optional)
278                                     //***end critical section***
279                                     //unlock
280                                             unlock(writerKey, avaiableProcesses[4]);
281                                     }
282                                     //signal
283                                     signal(&writerLock);
284
285                                     break;
286                     }
287                     processCount++;
288             }
289
290
291             return 0;
292     }
293
294
295
296
297
298     //checks if there are any writers in the critical section before atempting to
    enter.
299     void checkCriticalSection(bool writerKey){
```

```
300         if(writerKey==false){std::cout << "0 writer processes in the critical
    section\n";}
301         else{std::cout << "1 writer process in the critical section.\n";}
302 };
303
304 //checks if there are any readers in the critical section before attempting to
    enter.
305 void checkCriticalSection(bool readerKeys[]){
306         int total=0;
307         for(int i=0; i<2; i++){
308                 if(readerKeys[i]==false){total++;}
309         }
310         std::cout << total << " reader processes in the critical section\n";
311 };
312
313 //checks bolth if there are any writes or readers in the critical section before
    attemptin to enter.
314 void checkCriticalSection(bool readerKeys[], bool writerKey){
315         checkCriticalSection(readerKeys);
316         checkCriticalSection(writerKey);
317 };
318
319
320
321 //subwait is used to implement the counting semaphore. while availbleTime>0,
    checks if keyIsAvailable from the passedSemaphore = true. if it is returns true.
    if time has expired it resets availbleTime and returns false.
322 bool subWait(countingSemaphore * passedSemaphore){
323         while(passedSemaphore->getAvailableTime()>0){
324                 if(passedSemaphore->getKeyIsAvailable()==true){return true;}
325                 passedSemaphore->decreaseAvailableTime();
326         }
327         passedSemaphore->resetAvailableTime();
328         return false;
329 };
330
331
332
333 //countingSemaphore wait. checks if there any available counting semaphores
    (where available time =100). if not it returns false to allow another process to
    enter the critical section. if there are it calls subWait for that semaphore.
    returns the result.
334 bool wait( countingSemaphore passedSemaphores[]){
335         //asume an array size of 2.
336         for(int i=0; i<2; i++){
337                 if(passedSemaphores[i].getAvailableTime()==100){
338                         return subWait(& passedSemaphores[i]);
339                 }
340         }
341         return false;
342 };
343
344
345
346
347
348 //binarySemaphore wait. while the semaphores keyIsAvailable is false, break to
    allow another process to enter critical section. if it is true. change it to
    false, and allow the process to continue.
349 bool wait( binarySemaphore * passedSemaphore){
350         if(passedSemaphore->getKeyIsAvailable()==false){return false;}
351         else{
```

```
352                     passedSemaphore->setKeyIsAvailable(false);
353                     return true;
354             }
355     };
356
357     //signal for the binarySemaphore, changes the value of keyIsAvailable
358     void signal(binarySemaphore * passedSemaphore){
359             passedSemaphore->setKeyIsAvailable(true);
360     };
361
362     //signal for the countingSemaphore. compares the availableTime of both
        semaphores. changes the value of keyIsAvailable for the one that is lowest. (the
        one that has been waiting the longest)
363     //i might run itno a problem here. its an array of pointers os maybe it will
        change it, but i'm not entirely sure.
364     void signal(countingSemaphore passedSemaphore[]){
365             //assumed that the array is of size 2.
366             //im pretty sure that any process that is able to call signal will
        beable to signa; atleast once.
367
368             int oldest;
369             if(passedSemaphore[0].getAvailableTime()<=passedSemaphore
        [1].getAvailableTime()){oldest=0;}
370             else{oldest=1;}
371             passedSemaphore[oldest].setKeyIsAvailable(true);
372             passedSemaphore[oldest].resetAvailableTime();
373             return;
374     };
375
376     //checks to see if there are any readers in the critical section(for writer
        processes) if there are any in the critical section, return true. true causes
        the case to break. does not wait and does not change any variables
377     bool checkForReaders(countingSemaphore  readerSemaphores[]){
378             for(int i=0; i<2; i++){
379                     if(readerSemaphores[i].getKeyIsAvailable()==false){return true;}
380             }
381             return false;
382     };
383
384
385     //checks to see if there are any writers in the critical section(for reader
        processes) if there are any in the critical section, return true. true causes
        the case to break. does not wait and does not change any variables
386     bool checkForWriters(binarySemaphore * writerSemaphores){
387             if(writerSemaphores->getKeyIsAvailable()==false){return true;}
388             else{return false;}
389     };
390
391     //passed the readerKeys array and the current processObject.checks which
        readerKey is available to be locked and locks it.
392     //i don't like how i implemented this. i have to check which key is available
        first before entering swaping it. i know that can lead to some problems but i
        don't know how to get around it. may need to rewrite this.
393     void lock(processObject * processId, bool readerKeys[]){
394             for(int i=0; i<2; i++){
395                     if(readerKeys[i]==true){
396                             bool temp;
397                             temp=readerKeys[i];
398                             readerKeys[i]=processId->getHasKey();
399                             processId->setHasKey(temp);
400                             return;
401                     }
```

```
402             }
403     };
404
405     //passed the writerKey and processObject. locks the binarySemaphore
406     void lock(bool & writerKey,processObject * processId){
407             bool temp;
408             temp=writerKey;
409             writerKey=processId->getHasKey();
410             processId->setHasKey(temp);
411     };
412
413     //passed the readerKeys array and processObject. unlocks the first available
        spot in the array
414     void unlock(processObject * processId, bool readerKeys[]){
415             for(int i=0; i<2; i++){
416                     if(readerKeys[i]==false){
417                             bool temp;
418                             temp=readerKeys[i];
419                             readerKeys[i]=processId->getHasKey();
420                             processId->setHasKey(temp);
421                             return;
422                     }
423             }
424
425     };
426
427     //passed the writerKey and processObject. unlocks it
428     void unlock(bool &writerKey,processObject *processId){
429             bool temp;
430             temp=writerKey;
431             writerKey=processId->getHasKey();
432             processId->setHasKey(temp);
433     };
```