CSC 407: Computer Systems II: 2019 Spring,

Assignment #3

Purpose:

To practice using threads (and mutexes and conditions), and to practice safe C memory programming.

1. **Thread programming (50 Points)**
    a. Please cut, paste and finish the following program composed of 4 separate files below.
    b. `main()` should:
        - Create `2` child threads, both that run `evaluate()`. The `evaluate()` function should be passed the address of `nodeBuffer`.
        - Run `makeNode()` `NUM_PROBLEMS` times, and put the returned node address in `nodeBuffer`.
        - Wait for both child threads to finish
    c. `evaluate()` should `NUM_PROBLEM/2` times do:
        - Save the value returned by the `pullOut()` method into a `Node*` variable.
        - Print the iteration number, the node expression (obtainable as a C-string with the expression: `nodePtr->toString().c_str()`) and the value the node evaluates to (`nodePtr->eval()`).
        - `delete` the node pointer variable

    After the loop, the function should just `return(NULL)`.

2. **Stop and try your program now. It is multi-threaded, but not thread-safe. It should *not* work properly.**
    a. Make `NodeBuffer` thread safe by giving it the necessary mutex(es) and condition(s). Where is/are the critical sections? *Be sure to destroy your variables in `~NodeBuffer()`!*
3. `/*----------*`

```
4.  *---
     ---*
5.  *---            mathSolverHeader.h
     ---*
6.  *---
     ---*
7.  *---        This file defines constants and includes other
   files    ---*
8.  *---    necessary for the math generator and solver program.
     ---*
9.  *---
     ---*
10. *---    ----    ----    ----    ----    ----    ----    ----
     ----    ---*
11. *---
     ---*
12. *---    Version 1a             2019 May 6             Joseph
   Phillips ---*
13. *---
     ---*
14. *-----------------------------------------------------------
   ------------*/
15.
16.#include <cstdlib>
17.#include <cstdio>
18.#include <cstring>
19.#include <string>
20.#include <iostream>
21.#include <sstream>
22.#include <pthread.h>
23.#include <unistd.h>
24.
25.
26.//  PURPOSE:  To tell the maximum value that a tree can have.
27.const int MAX_VALUE              = 64;
28.
29.
30.//  PURPOSE:  To tell how many problems to do.
```

```
31.const int NUM_PROBLEMS           = 4096;
32.
33.
34.#include "Node.h"
35.#include "NodeBuffer.h"
36./*-------------------------------------------------------------
   ------------*
37. *---
     ---*
38. *---            Node.h
     ---*
39. *---
     ---*
40. *---        This file defines classes for nodes used to
   represent math   ---*
41. *---    expressions.
     ---*
42. *---
     ---*
43. *---    ----    ----    ----    ----    ----    ----    ----
     ----    ---*
44. *---
     ---*
45. *---    Version 1a             2019 May 6             Joseph
   Phillips ---*
46. *---
     ---*
47. *-------------------------------------------------------------
   ------------*/
48.
49.
50.//  PURPOSE:  To distinguish among the mathematical operators.
51.typedef          enum           {
52.                             ADD_OP,
53.                             SUBTRACT_OP,
54.                             MULTIPLY_OP,
55.                             DIVIDE_OP,
56.
```

```
57.                             NUM_OPS
58.                         }
59.                         operator_ty;
60.
61.
62.// PURPOSE: To serve as the base class for the Node classes.
63.class           Node
64.{
65.
66.public :
67.
68.  Node                           ()
69.                         { }
70.
71.  virtual
72.  ~Node                          ()
73.                         { }
74.
75.  virtual
76.  double eval                   ()
77.                         const
78.                         = 0;
79.
80.  virtual
81.  std::string    toString       ()
82.                                 const
83.                         = 0;
84.};
85.
86.
87.// PURPOSE: To represent a constant.
88.class           ConstNode : public Node
89.{
90.  double                 constant_;
91.
92.public :
93.
94.  ConstNode                      () :
```

```
95.                                 Node(),
96.                         constant_((double)((rand() %
   MAX_VALUE) + 1) )
97.                         { }
98.
99.  double eval             ()
100.                                const
101.                                { return(constant_); }
102.
103.   std::string  toString        ()
104.                                const
105.                                {
106.                                  std::ostringstream   stream;
107.
108.                                  stream << constant_;
109.                                  return(stream.str());
110.                                }
111.
112. };
113.
114.
115. //  PURPOSE:  To return a randomly generated Node.
116. extern
117. Node*           makeNode       ();
118.
119.
120. //  PURPOSE:  To represent an operation.
121. class           OperatorNode : public Node
122. {
123.   operator_ty                  operator_;
124.   Node*                        lhsPtr_;
125.   Node*                        rhsPtr_;
126.
127. public :
128.
129.   OperatorNode                 () :
130.                                Node(),
```

```
131.                                operator_((operator_ty)(rand()
    % NUM_OPS)),
132.                                lhsPtr_(makeNode()),
133.                                rhsPtr_(makeNode())
134.                                { }
135.
136.    ~OperatorNode               ()
137.                                {
138.                                  delete(rhsPtr_);
139.                                  delete(lhsPtr_);
140.                                }
141.
142.    double       eval           ()
143.                                const
144.                                {
145.                                  double       lhs    =
    lhsPtr_->eval();
146.                                  double       rhs    =
    rhsPtr_->eval();
147.                                  double       result;
148.
149.                                  switch  (operator_)
150.                                  {
151.                                  case ADD_OP :
152.                                  default :
153.                                    result     = lhs + rhs;
154.                                    break;
155.
156.                                  case SUBTRACT_OP :
157.                                    result     = lhs - rhs;
158.                                    break;
159.
160.                                  case MULTIPLY_OP :
161.                                    result     = lhs * rhs;
162.                                    break;
163.
164.                                  case DIVIDE_OP :
165.                                    result     = lhs / rhs;
```

```
166.                                   break;
167.                                 }
168.
169.                                 return(result);
170.                               }
171.
172.   std::string  toString       ()
173.                               const
174.                               {
175.                                 std::ostringstream   stream;
176.                                 const char*
    operatorNameCPtr;
177.
178.                                 switch  (operator_)
179.                                 {
180.                                 case ADD_OP :
181.                                 default :
182.                                   operatorNameCPtr   = " + ";
183.                                   break;
184.
185.                                 case SUBTRACT_OP :
186.                                   operatorNameCPtr   = " - ";
187.                                   break;
188.
189.                                 case MULTIPLY_OP :
190.                                   operatorNameCPtr   = " * ";
191.                                   break;
192.
193.                                 case DIVIDE_OP :
194.                                   operatorNameCPtr   = " / ";
195.                                   break;
196.                                 }
197.
198.                                 stream << "(" << lhsPtr_-
   >toString()
199.                                        << operatorNameCPtr
200.                                        << rhsPtr_->toString()
   << ")";
```

```
201.
202.                                   return(stream.str());
203.                                 }
204.
205. };
206. /*-------------------------------------------------------------
   --------------*
207.  *---
     ---*
208.  *---           NodeBuffer.h
     ---*
209.  *---
     ---*
210.  *---       This file defines a class that implements a
   thread-safe      ---*
211.  *---   buffer of pointers to math expressions.
             ---*
212.  *---
     ---*
213.  *---   ----    ----    ----    ----    ----    ----    ----
     ----    ---*
214.  *---
     ---*
215.  *---   Version 1a             2019 May 6             Joseph
   Phillips ---*
216.  *---
     ---*
217.  *-------------------------------------------------------------
   --------------*/
218.
219.
220. class   NodeBuffer
221. {
222.   enum { SIZE   = 16 };
223.
224.   Node*   array_[SIZE];
225.   int   inIndex_;
226.   int   outIndex_;
```

```
227.   int   numItems_;
228.
229. public :
230.
231.   NodeBuffer        ()
232.   {
233.     for  (int i = 0;  i < SIZE;  i++)
234.     {
235.       array_[i]= NULL;
236.     }
237.
238.     inIndex_ = outIndex_ = numItems_ = 0;
239.   }
240.
241.   ~NodeBuffer       ()
242.   {
243.   }
244.
245.   int   getNumItems  () const
246.   { return(numItems_); }
247.
248.   void  putIn (Node* nodePtr)
249.   {
250.     while  (getNumItems() >= SIZE)
251.     {
252.     }
253.
254.     array_[inIndex_] = nodePtr;
255.
256.     inIndex_++;
257.     numItems_++;
258.     if  (inIndex_ >= SIZE)
259.       inIndex_ = 0;
260.   }
261.
262.   Node*   pullOut ()
263.   {
264.     while  (getNumItems() <= 0)
```

```
265.     {
266.     }
267.
268.     Node* toReturn        = array_[outIndex_];
269.
270.     array_[outIndex_]    = NULL;
271.     outIndex_++;
272.     numItems_--;
273.     if  (outIndex_ >= SIZE)
274.       outIndex_ = 0;
275.
276.     return(toReturn);
277.   }
278. };
279. /*-------------------------------------------------------------
   --------------*
280.  *---
     ---*
281.  *---          mathSolver.cpp
     ---*
282.  *---
     ---*
283.  *---       This file defines the high-level functions of the
   math      ---*
284.  *---   generator and solver program.
     ---*
285.  *---
     ---*
286.  *---   ----    ----    ----    ----    ----    ----    ----
     ----    ---*
287.  *---
     ---*
288.  *---   Version 1a             2019 May 6             Joseph
   Phillips ---*
289.  *---
     ---*
290.  *-------------------------------------------------------------
   --------------*/
```

```
291.
292. //
293. //      Compile with:
294. //      $ g++ mathSolver.cpp -o mathSolver -lpthread -g
295. //
296.
297.
298. #include       "mathSolverHeader.h"
299.
300.
301. void*          evaluate       (void*         vPtr
302.                                )
303. {
304.   NodeBuffer*  nodeBufferPtr  = (NodeBuffer*)vPtr;
305.
306.   //  YOUR CODE HERE
307. }
308.
309.
310. //  PURPOSE:  To return a randomly generated Node.
311. Node*          makeNode       ()
312. {
313.   return( (rand() % 3) ? (Node*)new ConstNode() : (Node*)new
   OperatorNode() );
314. }
315.
316.
317. int            main           (int           argc,
318.                                 char*         argv[]
319.                                )
320. {
321.   NodeBuffer   nodeBuffer;
322.   pthread_t    consumer0;
323.   pthread_t    consumer1;
324.   int          toReturn       = EXIT_SUCCESS;
325.
326.   srand( (argc < 2) ? getpid() : atoi(argv[1]) );
327.
```

```
328.   //  YOUR CODE HERE
329.
330.   return(toReturn);
331. }
```

Sample output:

```
$ ./mathSolver 10
Made 0
Made 1
Made 2
Made 3
Made 4
Made 5
Made 6
Made 7
Made 8
Made 9
Made 10
Made 11
Made 12
Made 13
Made 14
Made 15
Made 16
Made 17
0 25 = 25.000000
1 54 = 54.000000
Made 18
Made 19
2 (24 - 4) = 20.000000
Made 20
3 54 = 54.000000
Made 21
4 (14 - (61 * (22 + 13))) = -2121.000000
Made 22
5 60 = 60.000000
6 55 = 55.000000
Made 23
```

```
Made 24
7 ((18 + ((33 + 36) / (47 - (1 / 22)))) / 2) = 9.734753
8 (59 - 59) = 0.000000
Made 25
...
Made 4089
2025 4 = 4.000000
Made 4090
2026 30 = 30.000000
Made 4091
Made 4092
2027 9 = 9.000000
Made 4093
2028 ((56 * 31) - 16) = 1720.000000
Made 4094
2029 ((59 / 60) / ((44 / 60) - 61)) = -0.016316
Made 4095
2030 (2 / (23 - 14)) = 0.222222
2031 41 = 41.000000
2032 49 = 49.000000
2033 ((35 * (35 - 32)) + 63) = 168.000000
2034 62 = 62.000000
2035 ((((41 / 4) - 18) / 20) * (((6 / 43) + (63 / 35)) + 46)) =
-18.576570
2036 37 = 37.000000
2037 ((2 - ((((47 * ((24 / 28) - 48)) + 7) + (3 / 55)) - 59)) -
48) = 2221.659740
2038 11 = 11.000000
2039 (25 - 31) = -6.000000
2040 ((((40 * (14 + 25)) / 46) * 7) - (20 * (15 * (50 + (22 /
63))))) = -14867.370600
2041 38 = 38.000000
2042 (20 - 46) = -26.000000
2043 (58 * ((16 * 27) * ((36 - 61) / 12))) = -52200.000000
2044 (33 / 11) = 3.000000
2045 48 = 48.000000
2046 31 = 31.000000
2047 19 = 19.000000
```

332.Safe C memory programming (50 Points)

The program below parses a given file path into a linked list of heap-allocated `DirEntryName` instances. A single heap-allocated `PathName` instance owns this list.

Please write:

- `getPathText()`
- `parseRestOfPath()`
- `destroy()`

Sample output:

```
$ ./dirPath /
Start from the root directory /
$ ./dirPath /hello
Start from the root directory /
  hello
$ ./dirPath /hello/there
Start from the root directory /
  hello
  there
$ ./dirPath hello/there
Start from current directory
  hello
  there
$ ./dirPath ~hello/there
Start from the home directory of hello
  there
$ ./dirPath ~hello/there.txt
Start from the home directory of hello
  there.txt
$ ./dirPath //
Missing directory name!
$ ./dirPath ~/hello/there
Start from the root directory /
  home          (the shell added this)
  instructor    (the shell added this too)
```

```
  hello
  there
$ ./dirPath ~/hello/the@re
Illegal character @ in path!

/*-------------------------------------------------------------
------------*
 *---
  ---*
 *---            dirPath.c
  ---*
 *---
  ---*
 *---        This file defines a program that parses a path
into its ---*
 *---    component entries.
  ---*
 *---
  ---*
 *---    ----    ----    ----    ----    ----    ----    ----
  ----    ---*
 *---
  ---*
 *---    Version 1.0            2019 May 6             Joseph
Phillips ---*
 *---
  ---*
 *-------------------------------------------------------------
------------*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define          LINE_LEN       256

struct   DirEntryName
```

```
{
  char*                 name_;
  struct DirEntryName*  nextPtr_;
};


struct   PathName
{
  int                   isRoot_;
  int                   isMyHome_;
  int                   isSomeonesHome_;
  struct DirEntryName*  dirEntryNamePtr_;
};


//  PURPOSE:  To return a non-zero value if 'c' is legal in a
restricted Unix
//       directory entry name, or '0' otherwise.
int             isLegalDirEntryChar
                       (char          c
                       )
{
  return( isalnum(c) || (c == '-') || (c == '_') || (c ==
'.') );
}


//  PURPOSE:  To return 'argv[1]' if there is at least one
argument on the
//       command line according to 'argc'.  Otherwise:
//       (1) asks the user for a path,
//       (2) enters it into 'textSpace' of length
'textSpaceLen',
//       (3) removes the ending '\n' from 'textSpace',
//       (4) and returns 'textSpace'.
const char*     getPathText    (int           argc,
                        char*         argv[],
                        char*         textSpace,
```

```
                          int           textSpaceLen
                         )
{
  //  YOUR CODE HERE
}


//  PURPOSE:  To create and return a linked list of heap-
allocated
//       struct DirEntryName instances that represent the path
'linePtr'.
//       (1) If 'linePtr' is empty (points to '\0') then
returns 'NULL'.
//       (2) If 'linePtr' is not empty then it
//           (a) allocates a 'struct DirEntryName' instance
from the heap,
//           (b) allocates a C-string to hold the directory
entry 'linePtr'
//               points to.
//           (c) allocates a new 'struct DirEntryName' for the
next directory
//               entry, etc.
//
//       Directory entries are separated by the '/' char.
//       If the directory entry is empty (e.g. "//") then it
does:
//               fprintf(stderr,"Missing directory name!\n");
//               exit(EXIT_FAILURE);
//       If the directory entry has a character not accepted
by
//       'isLegalDirEntryChar()' then it does:
//               fprintf(stderr,"Illegal character %c in
path!\n",*linePtr);
//               exit(EXIT_FAILURE);
struct DirEntryName*
         parseRestOfPath(const char*   linePtr
                         )
{
```

```
  //  I.  Application validity check:
  if  (linePtr == NULL)
  {
    fprintf(stderr,"NULL ptr to parseRestOfPath()\n");
    exit(EXIT_FAILURE);
  }


  //  II.  Return value:
  //  II.A.  Handle when at end of 'linePtr':
  //  YOUR CODE HERE

  //  II.B.  Handle when 'linePtr' points to '/',
  //          and thus a missing directory name:
  //  YOUR CODE HERE

  //  II.C.  Get entry name:
  char*  entryNamePtr   = linePtr;

  //  II.C.1.  Leave 'entryNamePtr' pointing to the beginning
of the entry name,
  //            and advance 'linePtr' until the characters are
no longer legal:
  //  YOUR CODE HERE

  //  II.C.2.  If you have stopped because of anything other
than '/' or '\0'
  //            then the user gave you an illegal char.  Give
error message here:
  //  YOUR CODE HERE

  //  II.C.3.  Allocate a new 'struct DirEntryName*' pointer
here.
  //            Allocate memory for its name and copy entry
name into that mem:
  //  YOUR CODE HERE
```

```
  //  II.C.4.  If 'linePtr' encountered '/' it should get the
value for
  //           'nextPtr_' by recursion.  If it points to '\0'
it should set
  //           'nextPtr_' to 'NULL'.
  //  YOUR CODE HERE

  //  III.  Finished:
  //  RETURN YOUR 'struct DirEntryName*' POINTER HERE
}



//  PURPOSE:  To return the address of a heap-allocated 'struct
PathName'
//       instance that encodes the path given by 'linePtr'.
struct PathName*
         getPath        (const char*   linePtr
                        )
{
  //  I.  Application validity check:
  if  (linePtr == NULL)
  {
    fprintf(stderr,"NULL ptr to getPath()!\n");
    exit(EXIT_FAILURE);
  }

  //  II.  Create 'struct PathName' object:
  //  II.A.  Obtain heap memory:
  struct PathName*      toReturn       = (struct PathName*)
                                 malloc(sizeof(struct
PathName));

  //  II.B.  Initialize flags of '*toReturn':
  toReturn->isRoot_                    = 0;
  toReturn->isMyHome_                  = 0;
  toReturn->isSomeonesHome_            = 0;

  switch  (*linePtr)
```

```
  {
  case '/' :
    toReturn->isRoot_    = 1;
    linePtr++;
    break;

  case '~' :
    linePtr++;

    if  (*linePtr == '\0')
      toReturn->isMyHome_       = 1;
    else
    if  (*linePtr == '/')
    {
      toReturn->isMyHome_       = 1;
      linePtr++;
    }
    else
      toReturn->isSomeonesHome_ = 1;

    break;

  case '\0' :
    fprintf(stderr,"Empty path!\n");
    exit(EXIT_FAILURE);
  }

  //  II.B.  Initialize 'dirEntryNamePtr_' of '*toReturn':
  toReturn->dirEntryNamePtr_    = parseRestOfPath(linePtr);

  //  III.  Finished:
  return(toReturn);
}


//  PURPOSE:  To print out the constructed path '*pathNamePtr'.
No return
//       value.
```

```
void            print          (struct PathName*
                                        pathNamePtr
                       )
{
  //  I.  Application validity check:
  if  (pathNamePtr == NULL)
  {
    fprintf(stderr,"NULL ptr to print()!\n");
    exit(EXIT_FAILURE);
  }

  //  II.  Print path:
  //  II.A.  Print beginning of path:
  int    sum    = pathNamePtr->isRoot_        +
           pathNamePtr->isMyHome_      +
           pathNamePtr->isSomeonesHome_;

  if  ( (sum < 0) || (sum > 1) )
  {
    fprintf(stderr,"Inconsistent pathname!\n");
    exit(EXIT_FAILURE);
  }

  struct DirEntryName*  run    = pathNamePtr-
>dirEntryNamePtr_;

  if  (pathNamePtr->isRoot_)
    printf("Start from the root directory /\n");
  else
  if  (pathNamePtr->isMyHome_)
    printf("Start from your home directory ~\n");
  else
  if  (pathNamePtr->isSomeonesHome_)
  {
    printf("Start from the home directory of %s\n",run->name_);
    run           = run->nextPtr_;
  }
  else
```

```
    printf("Start from current directory\n");

  //  II.B.  Print rest of path:
  for  ( ;  run != NULL;  run = run->nextPtr_)
    printf("  %s\n",run->name_);

  //  III.  Finished:
}


//  PURPOSE:  To 'free()' the memory of 'pathNamePtr': all
DirEntryName
//       'name_' and 'nextPtr_' member vars, and the memory of
'pathNamePtr'
//       itself.  No return value.
void            destroy        (struct PathName*
                                        pathNamePtr
                       )
{
  //  YOUR CODE HERE
}


//  PURPOSE:  To do the high level work of this program.
'argc' tells the
//       number of command line arguments.  'argv[]' points to
the arguments.
//       Returns 'EXIT_SUCCESS' to OS on success or
'EXIT_FAILURE' otherwise.
int             main           (int           argc,
                        char*         argv[]
                       )
{
  char   textSpace[LINE_LEN];
  const char*           linePtr =
getPathText(argc,argv,textSpace,LINE_LEN);
  struct PathName*      pathPtr = getPath(linePtr);
```

```
    print(pathPtr);
    destroy(pathPtr);
    return(EXIT_SUCCESS);
}
```