Purpose:

To practice working with POSIX threads, mutexes and conditions; and to practice good pointer programming in C

Assignment:

1. **The "telephone" game (50 Points)**

   Overview

   We are going to simulate the children's game of "telephone". In this game N (in our case, 10) children are in a circle. The first one says something to the second. The second one may or may not hear what the first one said accurately but he/she passes what he/she *thinks* was said to the third. This continues until the message gets to the last child.

   The program for our game has two classes. The first one, `Sentence` holds the current state of a sentence. It has been written for you, and all you have to do is use it. The second one, `MessageSystem` holds mutexes, conditions and sentence pointer buffers between each child. You'll have to finish this one.

Assignment Project Exam Help

   For this assignment you have to:

https://powcoder.com

   - You have to fill in the mutex and condition portions of `MessageSystem`.
   - Write the function `void* child (void*)` that the child threads will run.
   - Finish `main()` to invoke and wait for the child threads.

Add WeChat powcoder

   Each child `i` will get the pointer to its sentence from `i-1` and will transmit it (imperfectly) to `i`. Thus it needs to access two buffers: one at `i-1` and the other at `i`. All buffers are protected by mutexes, so child `i-1` and child `i` don't step on each other's toes when trying to access the buffer at `i-1`.

   Before child `i` can get the sentence pointer from the buffer at `i-1` the sentence has to be there. If it is not it should temporarily surrender the lock (if it obtained it) and wait until the sentence becomes available.

   Assignment

   Cut and paste the following

```
/*-------------------------------------------------------------------*
 *---                                                          ----*
 *----           telephoneGame                                 --
 *----                                                         ----*
 *---- This program simulates the children's game of "telephone",----*
 *----where an original message mutates as it is imperfectly ----*
 *----transmited among children in a pairwise manner.          ----*
 *----                                                         ----*
 *----     It demonstrates Linux/Unix thread programming using  ----*
 *----    pthread_mutex_t and pthread_cond_t.                   ----*
 *----                                                          ----*
 *----    Compile with:                                         ----*
 *----linux> g++ -lpthread telephoneGame.cpp -o telephoneGame   ----*
 *----                                                          ----*
 *----    ----    ----    ----    ----    ----    ----    ----  ----*
 *----                                                          ----*
 *-----------------------------------------------------------------*/



/*---------------------------------------------------------------------*
 *----                                                          ----*
 *----            Includes and namespace designations:          ----*
 *----                                                          ----*
 *-------------------------------------------------------------------*/

#include <cstdlib>
#include <iostream>
#include <string>
#include <pthread.h>

using     namespace     std;



/*----------------------------------------------------------------------*
 *----                                                          ----*
 *----                        Definitions of constants:         ----*
 *----                                                          ----*
 *--------------------------------------------------------------------*/

/*   PURPOSE:  To tell the number of children among whom to pass the
message.
 */
const      int    NUM_CHILDREN                     = 10;



/*   PURPOSE:  To tell the number of terms in the sentence.
 */
const      int    NUM_WORDS_IN_SENTENCE            = 9;



/*   PURPOSE:  To tell the number of possible mutations for each term.
 */
const      int    NUM_CHOICES_PER_POSITION         = 3;



/*   PURPOSE:  To tell the possible terms for each position in the
sentence.
 */
const string      words[NUM_WORDS_IN_SENTENCE][NUM_CHOICES_PER_POSITION]
         = { {"The","He","Wee"},
```

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

```
             {"quick","slick","thick"},
             {"brown","round","found"},
             {"fox","box","locks"},
             {"jumped","thumped","pumped"},
             {"over","clover","white cliffs of Dover"},
             {"the","be","three"},
             {"lazy","hazy","A to Z"},
             {"dog","frog","hog"}
           };



/*------------------------------------------------------------------*
 *----                                                         ----*
 *---- Definitions of classes and their methods and functions: ----*
 *----                                                         ----*
 *------------------------------------------------------------------*/

/*  PURPOSE:  To represent the current state of a Sentence.
 */
class      Sentence
{

 int     wordChoices[NUM_WORDS_IN_SENTENCE];

 //  Disallow copy-assignment
 Sentence&        operator=(const Sentence&);

public :

 //  PURPOSE:  To initialize '*this' sentence to its default state.  No
 //       parameters.  No return value.
 Sentence ()
 {
  for  (int index = 0;  index < NUM_WORDS_IN_SENTENCE;  index++)
   wordChoices[index] = 0;
 }

 //  PURPOSE:  To make '*this' a copy of 'source'.  No return value.
 Sentence (const Sentence& source)
 {
  for  (int  index = 0;  index < NUM_WORDS_IN_SENTENCE;  index++)
   wordChoices[index] = source.wordChoices[index];
 }

 //  PURPOSE:  To release the resources of '*this'.  No parameters.  No
 //       return value.
 ~Sentence  ()
 {
 }

 //  PURPOSE:  To return the current word at position 'index'.
 const string&   getWord (int index)  const
 {
  return(words[index][wordChoices[index]]);
 }
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
 //  PURPOSE:  To (potentially) mutate one term of '*this' sentence.
No
 //      parameters.  No return value.
 void    imperfectlyTransmit    ()
 {
  wordChoices[rand()%NUM_WORDS_IN_SENTENCE] = rand()
%NUM_CHOICES_PER_POSITION;
 }

};



/*  PURPOSE:  To print the text of Sentence 'sentence' to output stream
'os'
 *and to return 'os'.
 */
ostream& operator<<     (ostream& os, const Sentence& sentence)
{
 for  (int index = 0;  index < NUM_WORDS_IN_SENTENCE;  index++)
 {
  os << sentence.getWord(index);
  os << ((index == (NUM_WORDS_IN_SENTENCE-1)) ? '.' : ' ');
 }

 return(os);
}
```

Assignment Project Exam Help

https://powcoder.com

```
/*  PURPOSE:  To hold the state of the messaging system, including the
 *mutexes, conditions and sentence buffers.
 */
```

Add WeChat powcoder

```
class    MessageSystem
{

 //  YOUR CODE HERE FOR AN ARRAY OF NUM_CHILDREN+1 MUTEXES

 //  YOUR CODE HERE FOR AN ARRAY OF NUM_CHILDREN+1 CONDITIONS

 Sentence*                sentenceArray[NUM_CHILDREN+1];

 //  Disallow copy-construction
 MessageSystem            (const MessageSystem&);

 //  Disallow copy-assignment
 MessageSystem&  operator=(const MessageSystem&);

public :

 //  PURPOSE:  To initialize the array of mutexes, the array of
conditions
 //      and the array of sentence pointers to be NULL.
 MessageSystem   ()
 {
  for  (int index = 0;  index <= NUM_CHILDREN;  index++)
  {
   // YOUR CODE HERE TO INITIALIZE MUTEX NUMBER index
   // YOUR CODE HERE TO INITIALIZE CONDITION NUMBER index
   sentenceArray[index] = NULL;
  }
 }


 //  PURPOSE:  To destroy the mutexes in their array, to destroy the
conditions
 //       in their array, to delete() the sentence pointers in their
array.  No
 //       parameters.  No return value.
 ~MessageSystem  ()
 {
  for  (int index = 0;  index <= NUM_CHILDREN;  index++)
  {
   // YOUR CODE HERE TO DESTROY MUTEX NUMBER index
   // YOUR CODE HERE TO DESTROY CONDITION NUMBER index
   delete(sentenceArray[index]);
  }
 }


 //  PURPOSE:  To return a *pointer* to the lock at position 'index'.
 pthread_mutex_t*getLockPtr     (int    index)
 {
  return(/* YOUR CODE HERE TO RETURN A POINTER TO index-th MUTEX */
NULL);
 }


 //  PURPOSE:  To return a *pointer* to the condition at position
'index'.
 pthread_cond_t* getCondPtr     (int    index)
 {
  return(/* YOUR CODE HERE TO RETURN A POINTER TO index-th CONDITION */
NULL);
 }


 //  PURPOSE:  To "give away" (set equal to NULL) the pointer to the
sentence
 //       at position 'index'.
 Sentence*       giveSentencePtr (int index)
 {
  Sentence*       ptr = sentenceArray[index];

  sentenceArray[index]   = NULL;
  return(ptr);
 }


 //  PURPOSE:  To set the sentence pointer at position 'index' equal to
 //       'sentencePtr'.  No return value.
 void            setSentencePtr  (int index, Sentence* sentencePtr)
 {
  sentenceArray[index]   = sentencePtr;
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
 }



 //  PURPOSE:  To return 'true' if the sentence at position 'index' is
ready
 //       to be transmitted.
 bool            isReady        (int index) const
 {
  return(sentenceArray[index] != NULL);
 }

};



/*--------------------------------------------------------------------*
 *----                                                          ----*
 *----           Definitions of global variables:               ----*
 *----                                                          ----*
 *------------------------------------------------------------------*/

/*   PURPOSE:  To hold the global messaging system.
 */
MessageSystem    messageSystem;



/*------------------------------------------------------------------*
 *----                                                          ----*
 *----           Definitions of global functions:               ----*
 *----                                                          ----*
 *-------------------------------------------------------------------*/

/*   PURPOSE:  To get the necessary locks, get the sentence, print it,
 *transmit it (imperfectly), and unlock and signal the next child.
 */
void*    child   (void*  argPtr)
{

 //  I.  Applicability validity check:


 //  II. Run for current child:

 //  II.A.  Get 'index':

 //  YOUR CODE HERE


 //  II.B.  Announce that this child is ready:

 //  YOUR CODE HERE


 //  II.C.  Get both locks and wait until signaled (if need to):
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
 if   ( (rand() % 2) == 1)
 {
  //   YOUR CODE HERE
 }
 else
 {
  //   YOUR CODE HERE
 }

 //   YOUR CODE HERE


 //   II.D.  Get pointer to sentence, print it and transmit it:

 //   YOUR CODE HERE


 //   II.E.  Signal next child that message is ready and unlock their

 //   YOUR CODE HERE


 //   III.  Finished:

}



/*  PURPOSE:  To play the telephone game.  'argc' tells how many command
line
 *arguments there are.  'argv[]' points to each.  Returns 'EXIT_SUCCESS'
 *to OS.
 */
int       main (int argc, const char* argv[])
{

 //   I.  Applicability validity check:


 //   II.  Play game:

 //   II.A.  Seed random number generator:

 int              randNumSeed;

 if   (argc >= 2)
  randNumSeed = strtol(argv[1],NULL,10);
 else
 {
  string  entry;

  cout << "Random number seed? ";
  getline(cin,entry);
  randNumSeed = strtol(entry.c_str(),NULL,10);
 }
```

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

```
 srand(randNumSeed);


 //  II.B.  Play game:

 Sentence sentence;
 int              childIndex;
 int              indices[NUM_CHILDREN+1];
 //  YOUR CODE HERE FOR AN ARRAY OF NUM_CHILDREN+1 THREADS

 messageSystem.setSentencePtr(0,&sentence);

 for  (childIndex = NUM_CHILDREN;  childIndex > 0;  childIndex--)
 {
  indices[childIndex] = childIndex;
  //  YOUR CODE HERE TO INITIALIZE THREAD NUMBER childIndex
 }

 for  (childIndex = 1; childIndex <= NUM_CHILDREN;  childIndex++)
 {
  //  YOUR CODE HERE TO WAIT FOR THREAD NUMBER childIndex
 }

 cout << "Finally we have: \"" << *messageSystem.giveSentencePtr(10) <<
"\""
   << endl;


 //  III.  Finished:

 return(EXIT_SUCCESS);
}
```

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Finish `class MessageSystem`

The class needs two more arrays: one of NUM_CHILDREN+1 pthread-mutexes and another of NUM_CHILDREN+1 pthread-conditions. All the objects in both arrays should be initialized in the constructor, destroyed in the destructor, and should have pointers returned in `getLockPtr()` and `getCondPtr()`.

Write `void* child (void* argPtr)`

It might be useful to define an integer variable `index` corresponding to the integer pointed to by `argPtr`. Please note that `argPtr` has type `void*` and therefore does not know it points to an integer.

A simple `cout` statement should suffice here.

Time for mutexes! See that if statement? In the "then" part of it lock `messageSystem.getLockPtr(index)` first and `messageSystem.getLockPtr(index-1)` second. In the "else" part of it lock `messageSystem.getLockPtr(index-1)` first and `messageSystem.getLockPtr(index)` second. (We do this to try to convince ourselves that our program is robust over the vagaries of timing.) Then we `cout`. Finally we *loop* while our incoming message is not yet ready (method `isReady(index-1)`). In this loop we should wait to be signaled.

Here we've got to get a `Sentence*` (method `giveSentencePtr(index-1)`). We print out what we got and call method `imperfectlyTransmit()` on it (how do you run a method on an object given a pointer to it?). Finally we set the updated sentence with `setSentencePtr(index,`*yourSentencePtrVar*`)`

We got both locks `index-1` and `index`, and here we unlock them. Further, we should signal that there's a `Sentence` pointer in the buffer at index for child index+1.

Finish `int main ()`

We need an array of NUM_CHILDREN+1 pthreads. This array will be initialized in the first loop, where all child threads are to run your function `child()` and all are passed the *address* of `indices[childIndex]`. The child threads are initialized in *reverse* order to show how robust our solution is; our children are *well-behaved* and will not **throw a tantrum** if made to wait.

Assignment Project Exam Help

https://powcoder.com

In the second loop we wait for each thread to finish.

Sample output:

Add WeChat powcoder

```
$ telephoneGame 1
Child 9 ready to start
Child 9 got all his/her locks
Child 9 surrendering lock waiting for signal
Child 8 ready to start
Child 8 got all his/her locks
Child 8 surrendering lock waiting for signal
Child 7 ready to start
Child 7 got all his/her locks
Child 7 surrendering lock waiting for signal
Child Child 106 ready to start
 ready to start
Child 6 got all his/her locks
Child 6 surrendering lock waiting for signal
Child 5 ready to start
Child 5 got all his/her locks
Child 5 surrendering lock waiting for signal
Child 4 ready to start
Child 4 got all his/her locks
Child 4 surrendering lock waiting for signal
Child 3 ready to start
Child 3 got all his/her locks
Child 3 surrendering lock waiting for signal
Child 2 ready to start
Child 2 got all his/her locks
Child 2 surrendering lock waiting for signal
Child 1 ready to start
Child 1 got all his/her locks
Child 1 says "The quick brown fox jumped over the lazy dog."
Child 2 says "The quick brown fox jumped clover the lazy dog."
Child 3 says "The quick brown fox jumped clover the lazy dog."
Child 4 says "The quick brown fox jumped clover the lazy dog."
Child 5 says "The quick brown fox jumped clover the lazy dog."
Child 6 says "The quick brown fox jumped clover the hazy dog."
Child 7 says "The quick brown fox jumped clover the hazy hog."
Child 8 says "The quick brown fox jumped clover the hazy hog."
Child 9 says "The quick brown fox jumped clover the hazy hog."
Child 10 got all his/her locks
Child 10 says "The quick brown fox jumped clover the hazy frog."
Finally we have: "The slick brown fox jumped clover the hazy frog."
$ telephoneGame 2
Child Child 10 ready to start9 ready to start
Child 10 got all his/her locks
Child 10 surrendering lock waiting for signal

Child 9 got all his/her locks
Child 9 surrendering lock waiting for signal
Child 8 ready to start
Child 8 got all his/her locks
Child 8 surrendering lock waiting for signal
Child 6 ready to start
Child 6 got all his/her locks
Child 6 surrendering lock waiting for signal
Child 7 ready to start
Child 5 ready to start
Child 5 got all his/her locks
Child 5 surrendering lock waiting for signal
Child 4 ready to start
Child 4 got all his/her locks
Child 4 surrendering lock waiting for signal
Child 3 ready to start
Child 3 got all his/her locks
Child 3 surrendering lock waiting for signal
Child 2 ready to start
Child 2 got all his/her locks
Child 2 surrendering lock waiting for signal
Child 1 ready to start
Child 1 got all his/her locks
Child 1 says "The quick brown fox jumped over the lazy dog."
Child 2 says "The quick brown fox jumped over the lazy dog."
Child 3 says "Wee quick brown fox jumped over the lazy dog."
Child 4 says "Wee quick brown fox jumped over the lazy dog."
Child 5 says "Wee quick brown fox jumped over the lazy dog."
Child 6 says "Wee quick brown locks jumped over the lazy dog."
Child 7 got all his/her locks
Child 7 says "Wee quick brown locks jumped over the lazy hog."
Child 8 says "Wee quick brown locks jumped over the lazy hog."
Child 9 says "Wee slick brown locks jumped over the lazy hog."
Child 10 says "Wee slick brown locks jumped over the lazy hog."
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
Finally we have: "The slick brown locks jumped over the lazy hog."
$ telephoneGame 3
Child 10 ready to start
Child 10 got all his/her locks
Child 10 surrendering lock waiting for signal
Child 9 ready to start
Child 9 got all his/her locks
Child 9 surrendering lock waiting for signal
Child 8 ready to start
Child 8 got all his/her locks
Child 8 surrendering lock waiting for signal
Child 7 ready to start
Child 7 got all his/her locks
Child 7 surrendering lock waiting for signal
Child 6 ready to start
Child 6 got all his/her locks
Child 6 surrendering lock waiting for signal
Child 5 ready to start
Child 5 got all his/her locks
Child 5 surrendering lock waiting for signal
Child 4 ready to start
Child 4 got all his/her locks
Child 4 surrendering lock waiting for signal
Child 3 ready to start
Child 3 got all his/her locks
Child 3 surrendering lock waiting for signal
Child 2 ready to start
Child 2 got all his/her locks
Child 2 surrendering lock waiting for signal
Child 1 ready to start
Child 1 got all his/her locks
Child 1 says "The quick brown fox jumped over the lazy dog."
Child 2 says "The quick brown fox thumped over the lazy dog."
Child 3 says "He quick brown fox thumped over the lazy dog."
Child 4 says "He quick brown fox jumped over the lazy dog."
Child 5 says "He quick brown fox jumped over be lazy dog."
Child 6 says "He quick brown fox jumped over be lazy dog."
Child 7 says "He quick brown fox jumped over be lazy dog."
Child 8 says "He quick brown fox jumped over be lazy dog."
Child 9 says "He quick brown fox pumped over be lazy dog."
Child 10 says "He quick found fox pumped over be lazy dog."
Finally we have: "He slick found fox pumped over be lazy dog."
$ telephoneGame 4
Child Child Child 109 ready to start ready to start
Child 9 got all his/her locks
Child 9 surrendering lock waiting for signal
8 ready to start
Child 8 got all his/her locks
Child 8 surrendering lock waiting for signal

Child 7 ready to start
Child 7 got all his/her locks
Child 7 surrendering lock waiting for signal
Child 6 ready to start
Child 6 got all his/her locks
Child 6 surrendering lock waiting for signal
Child 5 ready to start
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
Child 5 got all his/her locks
Child 5 surrendering lock waiting for signal
Child 4 ready to start
Child 4 got all his/her locks
Child 4 surrendering lock waiting for signal
Child 1 ready to start
Child 1 got all his/her locks
Child 1 says "The quick brown fox jumped over the lazy dog."
Child 3 ready to start
Child 3 got all his/her locksChild
Child 3 surrendering lock waiting for signal
2 ready to start
Child 2 got all his/her locks
Child 2 says "The quick brown fox jumped white cliffs of Dover the lazy
dog."
Child 3 says "The quick brown fox pumped white cliffs of Dover the lazy
dog."
Child 4 says "The quick brown fox jumped white cliffs of Dover the lazy
dog."
Child 5 says "The slick brown fox jumped white cliffs of Dover the lazy
dog."
Child 6 says "The slick brown fox jumped white cliffs of Dover three
lazy dog."
Child 7 says "The slick brown fox jumped white cliffs of Dover three
lazy dog."
Child 8 says "The slick brown fox jumped white cliffs of Dover the lazy
dog."
Child 9 says "He slick brown fox jumped white cliffs of Dover the lazy
dog."
Child 10 got all his/her locks
Child 10 says "He slick brown fox jumped white cliffs of Dover the lazy
dog."
Finally we have: "He slick brown fox jumped white cliffs of Dover the
lazy dog."
$ telephoneGame 5
Child Child 9 ready to start10 ready to start
Child 9 got all his/her locks
Child 9 surrendering lock waiting for signal

Child 8 ready to start
Child 8 got all his/her locks
Child 8 surrendering lock waiting for signal
Child 6 ready to start
Child 6 got all his/her locks
Child 6 surrendering lock waiting for signal
Child 7 ready to start
Child 5 ready to start
Child 5 got all his/her locks
Child 5 surrendering lock waiting for signal
Child 4 ready to start
Child 4 got all his/her locks
Child 4 surrendering lock waiting for signal
Child 3 ready to start
Child 3 got all his/her locks
Child 3 surrendering lock waiting for signal
Child 2 ready to start
Child 2 got all his/her locks
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
Child 2 surrendering lock waiting for signal
Child 1 ready to start
Child 1 got all his/her locks
Child 1 says "The quick brown fox jumped over the lazy dog."
Child 2 says "Wee quick brown fox jumped over the lazy dog."
Child 3 says "The quick brown fox jumped over the lazy dog."
Child 4 says "The thick brown fox jumped over the lazy dog."
Child 5 says "The thick brown fox jumped over the lazy dog."
Child 6 says "The thick brown fox pumped over the lazy dog."
Child 7 got all his/her locks
Child 7 says "The thick brown fox pumped over the lazy frog."
Child 8 says "The thick brown fox pumped over the lazy frog."
Child 9 says "The thick brown locks pumped over the lazy frog."
Child 10 got all his/her locks
Child 10 says "The thick brown locks thumped over the lazy frog."
Finally we have: "The thick brown locks thumped clover the lazy frog."
$ telephoneGame 6
Child 9 ready to start
Child 9 got all his/her locks
Child 9 surrendering lock waiting for signal
Child 8 ready to start
Child 8 got all his/her locks
Child 8 surrendering lock waiting for signal
Child 7 ready to start
Child 7 got all his/her locks
Child 7 surrendering lock waiting for signal
Child 6 ready to start
Child 6 got all his/her locks
Child 6 surrendering lock waiting for signal
Child 5 ready to start
Child 5 got all his/her locks
Child 5 surrendering lock waiting for signal
Child 4 ready to start
Child 4 got all his/her locks
Child 4 surrendering lock waiting for signal
Child 3 ready to start
Child 3 got all his/her locks
Child 3 surrendering lock waiting for signal
Child 2 ready to start
Child 2 got all his/her locks
Child 2 surrendering lock waiting for signal
Child 1 ready to start
Child 1 got all his/her locks
Child 1 says "The quick brown fox jumped over the lazy dog."
Child 2 says "The quick brown fox pumped over the lazy dog."
Child 3 says "The quick brown fox pumped over the lazy dog."
Child 4 says "The quick brown fox pumped over the lazy frog."
Child 5 says "The quick brown fox pumped over the lazy frog."
Child 6 says "The quick brown fox pumped over the lazy frog."
Child 7 says "The quick brown fox pumped over the lazy frog."
Child 8 says "The quick brown fox pumped clover the lazy frog."
Child 9 says "The quick brown locks pumped clover the lazy frog."
Child 10 ready to start
Child 10 got all his/her locks
Child 10 says "The quick brown locks pumped clover the lazy dog."
Finally we have: "The quick brown locks pumped over the lazy dog."
$
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Useful knowledge:

Function	What it does
int pthread_create (/* Pointer to a pthread_t object */ pthread_t* restrict threadPtr, /* Pointer to optional object for properties of child */ const pthread_attr_t* restrict attr, /* Name of function to run: void* fncName(void* ptr) */ void *(*fncName)(void*), /* Ptr to object that is parameter to fncName() */ void *restrict arg)	Makes a thread in the space pointed to by threadPtr The thread run the function void* fncName(void*) and passes arg to it. Just leave attr as NULL for a generic thread.
int pthread_join (/* Which thread to wait for */ pthread_t thread, /* Pointer to pointer to receive pointer returned by exiting thread's function. */ void** valuePtrsPtr)	Waits for thread thread to finish. When it does valuePtr (the thing that valuePtrsPtr points to) is set to the thread's function's returned pointer value **or** it is ignored if valuePtr==NULL
int pthread_mutex_init (/* Ptr to space for mutex */ pthread_mutex_t *restrict mutexPtr, /* Type of mutex (just pass NULL) */ const pthread_mutexattr_t *restrict attr);	Initializes lock object pointed to by mutexPtr. Just use NULL for 2nd parameter.
int pthread_mutex_destroy	Releases resources taken by mutex pointed to

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

(/* Ptr to mutex to destroy *. pthread_mutex_t *mutex);	by mutexPtr.
int pthread_mutex_lock (/* Pointer to mutex to lock */ pthread_mutex_t *mutexPtr);	Either d. Gains lock and proceeds, or e. Waits for lock to become available
int pthread_mutex_unlock (/* Pointer to mutex to unlock */ pthread_mutex_t *mutexPtr);	Releases lock.
int pthread_cond_init (/* Pointer to space in which to make condition */ pthread_cond_t *restrict condPtr, /* Type of condition (just pass NULL) */ const pthread_condattr_t *restrict attr);	Creates a condition.
int pthread_cond_destroy (/* Pointer to condition to destroy */ pthread_cond_t *condPtr);	Destroys pointed to condition.
int pthread_cond_wait (/* Pointer to condition on which to wait */ pthread_cond_t *restrict condPtr, /* Pointer to mutex to surrender until receive signal */ pthread_mutex_t *restrict mutexPtr);	Suspends thread until receives signal on condPtr. While thread is suspended it surrenders lock on mutexPtr
int pthread_cond_signal (/* Ptr to condition which is signaled */ pthread_cond_t *condPtr);	Wakes up at least one thread waiting for signal on condPtr.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder