Lab 06

Started: Feb 26 at 12:35pm

Quiz Instructions

Question 1 **100 pts**

Task Description

Implement a solution to the Dining Philosophers problem using the pthread library's mutex and conditional variable functions.

Recall the pseudocode from the lecture notes:

```
monitor DiningPhilisophers {

   enum { THINKING; HUNGRY; EATING} state[5];
   condition self[5];

   void pickup (int i)
   {
      state[i] = HUNGRY;
      test(i);
      if (state[i] != EATING)
         self[i].wait();
   }

   void putdown (int i)
   {
      state[i] = THINKING;

      // test left and right neighbors
      test((i + 4) % 5);
      test((i + 1) % 5);

   }

   void test (int i)
   {
      if ( (state[(i + 4) % 5] != EATING) &&
         (state[i] == HUNGRY) && (state[(i + 1) % 5] != EATING) )
      {
         state[i] = EATING ;
         self[i].signal();
      }
   }

   initialization_code()
```

```
   {
      for (int i = 0; i < 5; i++)
         state[i] = THINKING;
   }
}
```

Implementation

The content of the archive file **philoBase.zip** is a naive implementation of the dining philosophers paradigm. Analyze the code. Do you see any problems with the code? Compile and run the code a number of times. At some point the program will hang.

If it does not, then experiment with some larger values for the number of seats and the number of rounds. Add the following line between the locking requests for chopsticks:

```
usleep(DELAY*2);
```

Add a similar line between the calls to return the chopsticks:

```
usleep(DELAY*4);
```

Experiment with the length of the delay.

Can you explain why the program gets stuck?

One way to fix the problem is to utilize `pthread_mutex_trylock()` with spinning. That implies using the approach in which picking both chopsticks is attempted, but if only one is available, then the other must be returned; no deadlocks will occur occurs. Try it out.

A better approach is to use monitors. C does not provide monitors, so you will implement them. A monitor synchronizes execution of its member functions, so only one function can be executing at any given time. The same effect can be achieved with pthread mutexes. The following is a pseudocode that adds a C function to a virtual monitor defined by a mutex:

```
function()
{
   wait(monitor_mutex);

      ...

   // body of the function

      ...

   signal(monitor_mutex);
```

```
}
```

You will need just one monitor for this assignment, so just one mutex is sufficient for all functions.

Note that a conditional variable must be associated with a mutex. It will be an error to call wait or signal on a conditional variable from outside of a critical section protected by the associated mutex.

Also, you can allocate a mutex statically in the following way:

```
pthread_mutex_t monitor_mutex = PTHREAD_MUTEX_INITIALIZER;
```

Note furthermore, that trying to lock the same lock twice blocks the thread.

With the `pickup()` and `putdown()` functions synchronized with a monitor, each philosopher is a thread using the following template:

```
...

pickup(i);

// EAT

putdown(i);

...
```

Generate a random number and use it in a delay in lieu of the eating time.

NOTE: Your implementation MUST follow this guideline.

Submission

You must submit the following:

- the signed source code,
- the `MakeLists.txt` file with which you built your application, and
- multiple scripts of tests that confirm that your program does not lock under any condition.

Upload Choose a File

Previous

Not saved Submit Quiz