PCYNLITX SOFTWARE TECHNOLOGIES

PCYNLITX CYBERNETIC THREAD MANAGEMENT LIBRARY FOR C++

ERKAM MURAT BOZKURT

What is pcynlitx multithreading library

Basically, pcynlitx multithreading is a C++ template library providing thread control functions for multithreading applications. However, differently from the classical C++ multithreading applications, in pcynlitx library, the theoretical foundation of the software cybernetics is used.

You can find the theoretical foundation and the scientific outcomes of the research study related with pcynlitx multi-threading library from the citation given below. The research paper published on a first class, peer-reviewed computer science jounal and it includes the explanation of the approach that is used on the pcynlitx library thread control functions, a comprehensive literature study and the performance tests that are carried out for cybernetic thread management. In fact, on the research paper, a general software development approach is introduced and the same methodology can be applied for different languages.

Bozkurt,Erkam Murat,The usage of cybernetic in complex software systems and its application to the deterministic multi-threading. Concurrency Computation Pract. Exper. 2022; 34(28):e7375. WILEY. doi:10.1002/cpe.7375

Fixing the execution order of the critical sections

Differently from the classical thread control, the software developer can fix the execution order of the critical sections with the help of the cybernetic thread control technology. Determination of the execution sequence of the code sections significantly improves internal determinism and This mechanism provides natural protection against the ordering violations that are faced multi-thread software applications. In other words, fixing the execution order of the code sections improves internal determinism and simplifies multi-threaded software development.

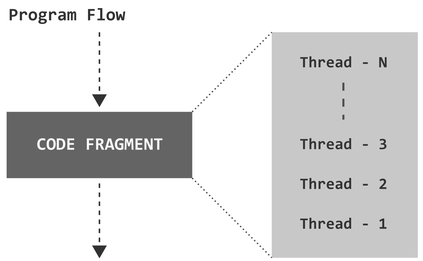

Figure 1. Fixing the execution order of the critical sections

Abstraction on multi-threading operations

The main objective of the pcynlitx software technologies is the simplification on the software development. Therefore, from the software developers perspective, it can be easily said that the pcynlitx multi- threading library tries to minimize the complexity of the multi-threaded software development and provides abstract interface for multi-threaded software management.

The thread control mechanism that is used in pcynlitx multi-threading

In the classical C++ multi-threading, the block status of the threads is changed when a particular condition occurs. In this classical approach, the block status of the threads change without recognizing the identity of the threads. However, instead, in pcynlitx multi-threading, the threads are controlled by means of their id numbers which are given by the software developers to the threads. In other words, the threads are blocked using ID numbers which are given by the programmer to the threads on thread creation. Moreover, you can control the threads in therms of the names of the functions which are executed by the threads. This situation has been illustrated on the figure which is given below.

Typical thread control operation

```
void function( ) {
     mt.lock();
     if(condition){
        cv.wait();
     }
     mt.unlock();
}
```

Pcynlitx block operation

```
void function( ) {
     syn.stop(1);
}
```

Figure 2. The comperation of the multi-threading approaches

A typical example for pcynlitx multithreading

```
#include <iostream>
#include <thread>
#include <pcynlitx>

using namespace pcynlitx;
void function(syncronizer & syn){
      syn.stop(2,1);
      std::cout  << "Thread Number:" << syn.number() << std::endl;
      syn.run(2,1);
}
int main( ){
     threads th(2);
     for(int i=0;i<2;i++){
          th.create(function,i,"function");
     }
     for(int i=0;i<2;i++){
          th.join(i);
     }
     return 0;
}
```

Explanation of the example

In the main function, at first, an instance of the pcynlitx::threads class constructed with name 'th'. Then, the threads are created with create member function of the threads object. The create member function works as variadic and you can pass any number of argument to the create member function in pcynlitx multithreading. However, the first three argument is mandatory. These are the address of the function to be executed, the id number of the thread and the name of the function in terms of std::string. In fact, the cybernetic thread management system collects its data from the software developer over the create member function. Meanwhile, an instance of the synchronizer object must be declared.The synchronizer object is responsible from the thread control operations and it is a public member of the instance of the threads class. Moreover, it is passed to the thread function automatically.

Secure messaging between the threads

In pcynlitx applications, to accomplish message-sending concurrency, you can send any information between the threads over a channel object. However, in real multithreading applications, a new questions arises when threads are synchronized by means of messages: which message arrives first!. The pcynlitx multithreading provides a solution to this problem with controlling the thread execution order. In pcynlitx applications, Channel class works based on a last in first out queue class. A typical example for channel class usage has been given in below.

```
void PrintMessage(synchronizer_channel<std::string>  &  syn){
     syn.stop(3,0); // The thread 3 always executed after thread-0
     if(syn.number() == 0){
        std::string s = "Hello";
        syn <<  s;
     }
     if(syn.number() == 3){
       std::string s;
       syn >>  s;
       std::cout <<  "The message coming:" <<  s  << std::endl;
     }
     syn.run(3,0);      /* The thread-3 starts its execution after
                           thread-1 calls syn.run(3,0) */
}

int main( ){
    channel<std::string>  ch;
    ch.set_producer(0);
    ch.set_consumer(3);
    threads<Test>  th(this,4);
    for(int i=0;i<4;i++){
       th.create(Test::PrintMessage,i);
    }
    for(int i=0;i<4;i++){
       th.join(i);
    }
}
```


er.muratbozkurt@gmail.com