电子科技大学

作 业 报 告

学生姓名： 冯文森 **学 号：201522010534 指导教师：李林**

学生 E-mail：1071881690@qq.com

一、作业名称

编写一个多队列线程池应用

2、 作业要求

1.该消息队列不能依赖于已有库的队列，必须是自己设计实现的

2.该消息队列是线程安全的

3.该消息队列能接收 CLMessage 继承体系的消息

4.该消息队列能融入到本课程所提供的程序库中，能支持线程之间的通信（即需要提供发送消息的通信类，以及消息循环的管理类）

5.编写一个实现了整数加法运算的线程池

6.在这个应用中，有三种线程：

1） 一个用户线程。用于向线程池（主控线程）提出加法计算请求

2） 一个主控线程。用于接收用户线程发送的加法计算请

求，并负载均衡地将请求下发到若干计算线程

3）若干计算线程。用于接收来自于主控线程转发的加法计算请求，完成加法计算，并将计算结果直接返回给用户线程

三、设计与实现

根据作业要求，需实现三种线程，工作线程、主控线程和用户线程，这三种线程互相可通信，用户线程向主控线程发送计算请求，主控线程将用户线程发送来的计算请求分发给工作线程，工作线程再将计算结果直接回馈给用户线程，其关系图见 3.1，工作线程由链表实现，从而主控线程可将工作均匀的分配给工作线程。

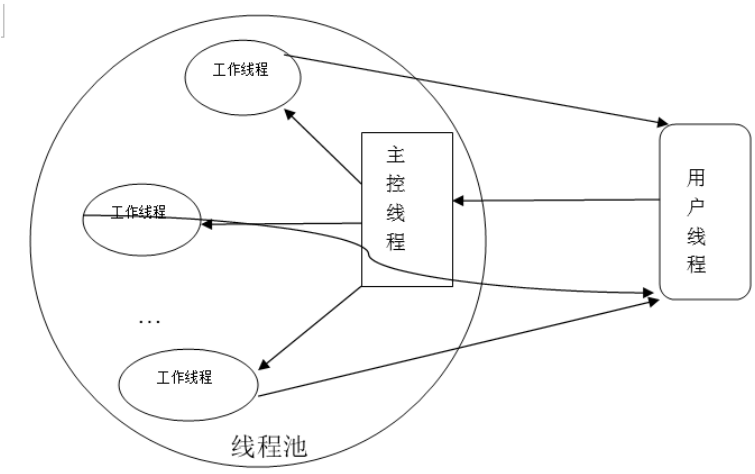

图 3-1 三种线程之间的关系

具体实现如下：

代码清单 3.1 CLMyThreadPool.cpp 线程池实现

```
#include<iostream>
#include "CLMyThreadPool.h"
```

```
using namespace std;

class CLMyCalculate;
// 构造函数，初始化线程数，默认为5，同时主线程名为mainThread
CLMyThreadPool::CLMyThreadPool () {

   ThreadSum = 5;
   MainThreadName = "mainThread";
   ThreadHead = NULL;
}
// 初始化线程池中线程数目
CLMyThreadPool::CLMyThreadPool (int number) {

   if (number <= 0)
      ThreadSum = 5;
   else
      ThreadSum = number;

   MainThreadName = "mainThread";
   ThreadHead = NULL;
}

CLMyThreadPool::~CLMyThreadPool() {

}
// 初始化线程池，
void CLMyThreadPool::InitThreadPool() {
     // 主控线程
   CLThread *mainThread = new CLThread(
                                    new CLExecutiveFunctionForMsgLoop(
                  new CLMsgLoopManagerForSTLqueue(
                  new CLMain(this), MainThreadName.c_str())),  false);
   mainThread->Run(0);

   for (int i = 0; i < ThreadSum; ++i) {
      // 创建 ThreadSum 个工作线程，同时记录线程名
      CLThreadLink *thread = new CLThreadLink;
      char t[32];
      string s;
      sprintf(t, "%d", i);
      s = t;
      thread->ThreadName = "thread_" + s;
      thread->Next = ThreadHead;
```

```
    ThreadHead = thread;

    CLThread *proxy = new CLThread(
                                new CLExecutiveFunctionForMsgLoop(
                new CLMsgLoopManagerForSTLqueue (
                new CLMyCalculate, (thread->ThreadName).c_str())),  false);
    proxy->Run(0);
  }

  CurrentThread = ThreadHead;
}
// 用户线程将消息发送给主控线程
void CLMyThreadPool::PostMessageToMainThread(CLMessage *msg) {

  CLExecutiveNameServer::PostExecutiveMessage(MainThreadName.c_str(), msg);
}
```

代码清单 3.1 中为线程池实现类，线程池中线程数目默认为 5 个，使用者也可自己定义，InitThreadPool()函数创建主控线程和若干工作线程，PostMessageToMainThread(CLMessage *msg)中，用户向主控线程发送消息。

代码清单 3.2 CLMyMainPro.cpp 主控线程实现

```
#include<iostream>
#include "./CLMyMainPro.h"
using namespace std;
//
CLMain::CLMain(CLMyThreadPool *tPool) {
  this->threadPool = tPool;
}

CLMain::~CLMain() {

}
// 主控线程初始化，注册 ADD_MSG 和 QUIT_MSG 事件
CLStatus CLMain::Initialize(CLMessageLoopManager *pMessageLoop, void *pContext) {

                    pMessageLoop->Register(ADD_MSG,      (CallBackForMessageLoop)
(&CLMain::Do_PostMsgToThreadPool));
                    pMessageLoop->Register(QUIT_MSG,     (CallBackForMessageLoop)
(&CLMain::Do_QuitMsg));
  return CLStatus(0, 0);
}
// 将工作分发给工作线程
CLStatus CLMain::Do_PostMsgToThreadPool(CLMessage *pM) {
```

```
    CLAddMessage *p = (CLAddMessage*)pM;
    CLMessage *pp = new CLAddMessage(p->m_Op1, p->m_Op2);
                    CLExecutiveNameServer::PostExecutiveMessage(threadPool->CurrentThread-
>ThreadName.c_str(), pp);

    threadPool->CurrentThread = threadPool->CurrentThread->Next;
    // 循环使用工作线程
        if (threadPool->CurrentThread == NULL)
        threadPool->CurrentThread = threadPool->ThreadHead;
    return CLStatus(0, 0);
}
// 退出消息时，将退出消息发送给每个工作线程
CLStatus CLMain::Do_QuitMsg(CLMessage *pM) {
    CLThreadLink *p = threadPool->ThreadHead;
    while(p) {
        CLQuitMessage *q = new CLQuitMessage();
        CLExecutiveNameServer::PostExecutiveMessage(p->ThreadName.c_str(), q);
        p = p->Next;
    }
    return CLStatus(QUIT_MESSAGE_LOOP, 0);
}
```

代码清单 3.2 中，主控线程的任务主要是注册 ADD_MSG 和 QUIT_MSG 事件，将接收到的非退出分发消息给工作线程，有退出消息时将退出消息发送给每个工作线程。

代码清单 3.3 CLMyCalculatePro.cpp 计算线程实现

```
#include<iostream>
#include"./CLMyCalculatePro.h"
using namespace std;

CLMyCalculate::CLMyCalculate() {

}

CLMyCalculate::~CLMyCalculate(){

}
// 注册 ADD_MSG 和 QUIT_MSG 事件
CLStatus CLMyCalculate::Initialize(CLMessageLoopManager *pMsgLoop, void *pContext) {

                    pMsgLoop->Register(ADD_MSG,     (CallBackForMessageLoop)
(&CLMyCalculate::Do_AddMsg));
                    pMsgLoop->Register(QUIT_MSG,    (CallBackForMessageLoop)
(&CLMyCalculate::Do_QuitMsg));
```

```
    return CLStatus(0, 0);
}
// 加法事件处理，将结果回馈给用户线程
CLStatus CLMyCalculate::Do_AddMsg(CLMessage *pM) {

    CLAddMessage *pAddMsg = (CLAddMessage*)pM;
    int result = pAddMsg->m_Op1 + pAddMsg->m_Op2;
        CLResultMessage *pResultMsg = new CLResultMessage(pAddMsg->m_Op1, pAddMsg->m_Op2, result);
    CLExecutiveNameServer::PostExecutiveMessage("user_thread", pResultMsg);
    return CLStatus(0, 0);
}
// 退出事件处理
CLStatus CLMyCalculate::Do_QuitMsg(CLMessage *pM) {

    return CLStatus(QUIT_MESSAGE_LOOP, 0);
}
```

代码清单 3.3 中为计算线程实现，其工作主要是处理主控线程发来的事件，将处理后的加法事件回馈给用户线程.

代码清单 3.4 CLUserThreadPro.cpp 用户线程实现

```
#include<iostream>
#include"CLUserThreadPro.h"
using namespace std;

CLUser::CLUser(CLMyThreadPool *pool) {

    this->Pool = pool;
}

CLUser::~CLUser() {}
// 再注册完 RESULT_MSG 和 QUIT_MSG 事件后，向主线程发送 1000 条加法消息
CLStatus CLUser::Initialize(CLMessageLoopManager *pMessageLoop, void *pContext) {
                    pMessageLoop->Register(RESULT_MSG,      (CallBackForMessageLoop)
(&CLUser::Do_ResultMsg));
    SendCount = 1000;
                        pMessageLoop->Register(QUIT_MSG,      (CallBackForMessageLoop)
(&CLUser::Do_QuitMsg));
    ReceivedResultCount = 0;
    for (int i = 0; i < SendCount; ++i) {
        // 向主线程发送加法消息，加数分别为(0, 0),(1, 1)…(999, 999)
        Pool->PostMessageToMainThread(new CLAddMessage( i, i));
    }
```

```
    return CLStatus(0, 0);
}
// 返回结果事件实现
CLStatus CLUser::Do_ResultMsg(CLMessage *pM) {
    CLResultMessage *pResult = (CLResultMessage*)pM;
     cout << pResult->m_Op1 << "+" << pResult->m_Op2 << "=" << pResult->m_Result <<
endl;
    ReceivedResultCount++;
    if (ReceivedResultCount == SendCount) {
        Pool->PostMessageToMainThread(new CLQuitMessage());
        return CLStatus(QUIT_MESSAGE_LOOP, 0);
    }
    return CLStatus(0, 0);
}
// 退出事件实现
CLStatus CLUser::Do_QuitMsg(CLMessage *pM) {
    return CLStatus(QUIT_MESSAGE_LOOP, 0);
}
```

代码清单 3.4 为用户线程实现，其中主要功能是向主线程发送加法消息请求，发送了 1000 条消息。同时注册了 RESULT_MSG 和 QUIT_MSG 事件，分别处理来自工作线程的结果事件和处理退出事件.

四、测试

代码清单 4.1 test.cpp 测试代码

```
int main()
{
    // 创建线程池
    CLMyThreadPool *pool = new CLMyThreadPool(10);
    pool->InitThreadPool();  // 初始化

    sleep(2);
    // 创建用户线程
    CLThread* userThread = new CLThread(
                new CLExecutiveFunctionForMsgLoop(
                new CLMsgLoopManagerForSTLqueue(
                new CLUser(pool), "user_thread")), true);
    // 启动用户线程
    userThread->Run(0);
    userThread->WaitForDeath();
    return 0;
}
```

代码清单 4.1 为测试代码，首先创建了线程池，并初始化，再创建了用户线程（具体实现

见代码清单 3.4），用户线程向主线程发送 1000 条加法消息请求。
实现结果截图如下：

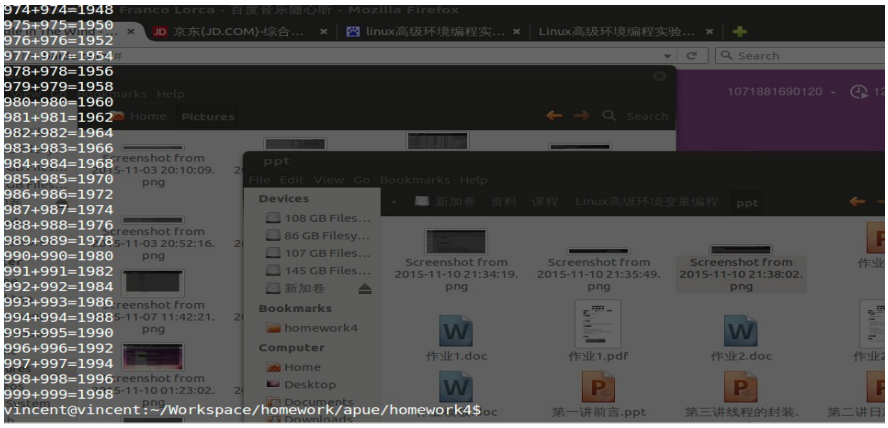

图 4.1 测试结果

如图 4.1 所示，用户线程收到的加法结果。
详细的测试结果可见 src/homework4/log.txt 文件：

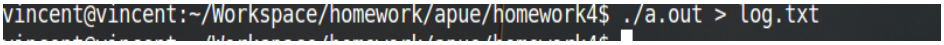

图 4.2 测试结果导入 log.txt 文件

五、对本课程或本作业的建议和意见

六、附录

详细代码实现见文件 src/homework4