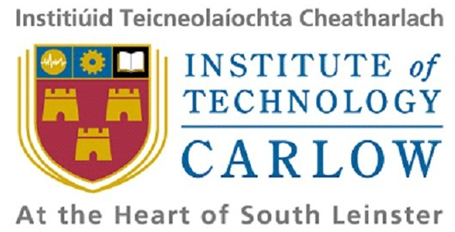
****

**Student ID: C00131026**

**Student Name: Guanting Su**

**Supervisor: Joseph Kehoe**

**Date of submission: 8.4.2011**

**Code Listing**

**<STM Lua>**

**(CW228)**

Content

[1. Create metatable 1](#_Toc290277840)

[2. Create Transaction 2](#_Toc290277841)

[3. Run function with thread 6](#_Toc290277842)

[4. Head file of C code 7](#_Toc290277843)

[5. Global variables 8](#_Toc290277844)

[6. Call function from Lua 8](#_Toc290277845)

[7. Waiting Lua call 9](#_Toc290277846)

[8. Create thread 10](#_Toc290277847)

[9. Thread run functions 11](#_Toc290277848)

[10. Lock operatons 12](#_Toc290277849)

[11. Register into Lua 13](#_Toc290277850)

[12. Test 14](#_Toc290277851)

|  |
| --- |
| Create metatable |
| function createmetatable(data)  local \_t=data  --overwrite metatable mt  local mt = {  \_\_index = function (t,k)  print("\*access to element " .. tostring(k))  if \_t[k]==nil then  return ('the element is not defined!!')  else  -- access the original table  return \_t[k].value  end  end,  \_\_newindex = function (t,k,v)  print("\*update of element " .. tostring(k) ..  " to " .. tostring(v))    if type(v)=='table' then  - update original table  \_t[k] = v  else  \_t[k].value = v  end  end,  --transform table to string  \_\_tostring = function(t)  local s = "{"  local sep = ""  for k in pairs(t) do  s = s .. sep .. k  sep = ", "  end  return s .. "}"  end,  --get current version number of element  readversion = function(k)  print print("access the element "..  tostring(k).."'s version)  if \_t[k]==nil then  return("the element is not defined!!")  else  return \_t[k].version  end  end,  --update new version into element  updateversion=function(k,v)  if \_t[k]==nil then  return ("the element is not defined!!")  else  \_t[k].version=v  end  end  }  return mt  end |

|  |
| --- |
| Create Transaction |
| function createTransaction()  local x={}  local array={}  local count=1  local data={}  meta=createmetatable(data)  --put new element into table x  function x.add(key,v)  x[key]={value=v,version=1}  table.insert(array,key)  array[key]=v  end  --get new values from global table  function x.read(key)  if g[key]~=nil and x[key] ~=nil then  x[key]=g[key].value  x.updateversion(key,g[key].version)  else  return('the element is not exist!!')  end  end  --update element to global table  function x.write(key)  g[key]={value=x[key],version=x.readversion(key)}  if g[key]==nil then  g[key]={value=x[key],  version=x.readversion(key),  lock=false}  g[key].value=x[key]  g[key].version=x.readversion(key)  elseif g[key]~=nil then  g[key].value=x[key]  g[key].version=x.readversion(key)  end  --~ return g[key]  End  function x.commit()  local done=false  local update=true  print("strat to cmmit")  setlock()  for k,v in ipairs(array) do  if g[v]~=nil and g[v].lock==false then  update=true  elseif g[v]==nil then  update=true  else  update=false  break  end  end  --~ print("check lock")  if update==false then  unlock()  x.commit()  elseif update==true then  --~ lock()  updatelockt()  local versionmatch=true  for k1,v1 in ipairs(array) do  if g[v1]~=nil then  print(x.readversion(v1),g[v1].version)  end  if g[v1]~=nil and  x.readversion(v1)==g[v1].version then  ersionmatch=true  done=true  elseif g[v1]==nil then  ersionmatch=true  done=true  elseif x.readversion(v1)~=g[v1].version  and g[v1]~=nil then  versionmatch=false  done=false  break  end  end  print("match version "..tostring(versionmatch))  for k,v in ipairs(array) do  if x[v]~=nil and g[v]==nil and  done==true then  x.updateversion(v,x.readversion(v)+1)  x.write(v)  elseif x[v]~=nil and g[v]==nil and  done==false then  x[v]=array[v]  elseif x[v] ~=nil and  g[v]~=nil and  versionmatch==true then  x.updateversion(v,g[v].version+1)  x.write(v)  elseif x[v] ~=nil and  g[v]~=nil and  versionmatch==false and  done==false then  if g[v]~=nil then  x.read(v)  end  end  end  end  --~ setlock()  --~ if(done==true) then  --~ for k, v in ipairs(array) do  --~ g[v].lock=false  --~ end  --~ end  --~ unlock()  --~ lockf()  setlock()  updatelockf()  print("end commit"..tostring(done))  return done  end  function x.printarray()  --return(x.tostring(array))  local str=''  for k,v in ipairs(array) do  --print(k, v)  str=str..k..'='..v  end  return str  end  function x.readversion(key)  local num=meta.readversion(key)  return num  end  function x.updateversion(key,version)  meta.updateversion(key,version)  end  function x.tostring(ls)  return "{" .. table.concat(ls, ", ") .. "}"  end  setmetatable(x,meta)  return x  end |

|  |
| --- |
| Run function with thread |
| --run function is defined by userd  function paraDo(...)  local funcname=nil  for i=1, select("#",...) do  local func=select(i,...)  if(type(func)~="function") then  print("Last parameter  not function: "..tostring(func) )  else  func()  end  end  end  function callfunc(func\_name)  strFun="paraDo"  local foo = loadstring(strFun .. "(" .. func\_name .. ")")  if foo~=nil then  foo()  else  print("no this function!!"..tostring(func\_name))  end  end  --call C function to create thread  function createT(func\_name)  createthread(func\_name)  end  function updatelockt()  for k,v in pairs(g) do  g[k].lock=true  end  unlock()  end  function updatelockf()  for k,v in pairs(g) do  g[k].lock=false  end  unlock()  end  --print out global table  function printGtable()  for k,v in pairs(g) do print(k,v.value) end  end  --read variable from global table  function readVariable(k)  if g[k]~=nil then  return g[k].value  elseif g[k]==nil then  print("please make sure input right variable name")  elseif type(k)~="string" then  print("please a string value")  end  end |

|  |
| --- |
| Head file of C code |
| /\*connect libraries of Lua\*/  #pragma comment(lib,"../lib/lua5.1.lib")  /\*open Lua head file use C API\*/  #include "lua.h"  #include "lauxlib.h"  #include "lualib.h"  #include <string.h>  #include <stdio.h>  #include <ctype.h>  #include <stdlib.h>  //#include "threadtest.h"  #ifndef \_THREAD\_STACK\_SIZE  # if (defined PLATFORM\_WIN32) || (defined PLATFORM\_POCKETPC) || (defined PLATFORM\_CYGWIN)  # define \_THREAD\_STACK\_SIZE 0  #endif  #endif |

|  |
| --- |
| Global variables |
| #include "stmlock.h"  #include <windows.h>  #include<process.h>  const char \*funcname;  lua\_State\* MainState;  HANDLE Mutex;  void createThread();  void use\_lua\_paraDo(lua\_State \*L,const char \*func\_name);  void static runThread (void \* pParam);  static unsigned \_\_stdcall threadfunc(void \* pParam);  DWORD WINAPI Fun1Proc(LPVOID param);  void newThread(void( \*ThreadProc )( void \* )); |

|  |
| --- |
| Call function from Lua |
| void use\_lua\_paraDo(lua\_State \*L)  {  printf("call fuction \n");  /\* function to be called \*/  lua\_getglobal(L, "callfunc");  if (!lua\_isfunction(L, -1)){  luaL\_error(L,"fail to a function");  return;  }  /\*push arguments into stack\*/  /\*lua\_pushstring(L, "paraDo");  lua\_pushstring(L, func\_name); \*/  lua\_pushstring(L,funcname);  /\* do the call (1 arguments, 0 result) \*/  if (lua\_pcall(L, 1, 0, 0) != 0){  luaL\_error(L,"fail to call function");  return;  }  printf("function end\n");  } |

|  |
| --- |
| Waiting Lua call |
| int lua\_createThread(lua\_State \*L){  /\*check user input the right function name or not\*/  if(!lua\_isstring(L,1)){  luaL\_error(L,"Please type right function name");  return -1;  }else{  //get the function name from lua  funcname=lua\_tostring(L,1);  }  /\* Get Lua State \*L \*/  MainState=L;  /\*empty the stack\*/  //lua\_pop(L,2);  lua\_pop(L,1);    /\*create mutex lock\*/  Mutex = CreateMutex(NULL,FALSE,NULL);  /\*call createThread function to create thread\*/  createThread();  /\* return 0 result\*/  return 0;  } |

|  |
| --- |
| Create thread |
| void createThread(){  /\*declare a new handle\*/  HANDLE newThread;  /\*declare a get error no.\*/  int ErrorNumber, DOSErrorNumber;  /\*create a thread to run function that is define by user\*/  printf("create thread to run function %s \n",funcname);  newThread=(HANDLE)\_beginthreadex(NULL,0,threadfunc,NULL, 0, NULL) ;  /\*if fail to create thread print out the error detail\*/  if(newThread==0){  ErrorNumber = errno;  DOSErrorNumber = \_doserrno;  printf( "errno = %d\n", ErrorNumber );  printf( "doserrno = %d\n", DOSErrorNumber );  }  printf("end thread \n");  Sleep(100);  //release memory  CloseHandle(newThread);  } |

|  |
| --- |
| Thread run functions |
| static unsigned \_\_stdcall threadfunc(void \* pParam)  {  printf("run thread \n");  /\*call function from Lua code\*/  use\_lua\_paraDo(MainState);  printf("end call \n");  //exit thread  \_endthreadex(0);  return 0;  }  void static runThread(void \* pParam){    /\*lua\_State \*L;  L= lua\_open();  InitLuaState(L);  if(luaL\_loadfile(L, "../test4.lua")||lua\_pcall(L, 0, LUA\_MULTRET, 0)){  luaL\_error(L,"fail to load script");  lua\_close(L);  \_endthread();  return ;  }else{  luaL\_error(L,"fail to load script");  }\*/  printf("run thread \n");  //use\_lua\_paraDo(L,funcname);  //use\_lua\_paraDo(MainState,funcname);  use\_lua\_paraDo(MainState);  \_endthread();  } |

|  |
| --- |
| Lock operatons |
| static int Lua\_endLock(lua\_State \*L)  {  printf("create a lock \n");  Mutex = CreateMutex(NULL,FALSE,NULL);  return 0;  }  static int Lua\_setlock(lua\_State \*L)  {  printf("set a lock \n");  WaitForSingleObject(Mutex,INFINITE);  return 0;  }  static int Lua\_unlock(lua\_State \*L)  {  printf("release a lock \n");  ReleaseMutex(Mutex);  return 0;  }  static int lua\_updateLock1(lua\_State \*L)  {  /\* function to be called \*/  lua\_getglobal(L, "updatelockt");  if (!lua\_isfunction(L, -1)){  luaL\_error(L,"fail to a function");  return -1;  }  WaitForSingleObject(Mutex,INFINITE);  if (lua\_pcall(L, 0, 0, 0) != 0){  luaL\_error(L,"fail to call function");  return -1;  }  printf("function end\n");  ReleaseMutex(Mutex);  return 0;  }  static int lua\_updateLock2(lua\_State \*L)  {  lua\_getglobal(L, "updatelockf"); /\* function to be called \*/  if (!lua\_isfunction(L, -1)){  luaL\_error(L,"fail to a function");  return -1;  }  WaitForSingleObject(Mutex,INFINITE);  if (lua\_pcall(L, 0, 0, 0) != 0){  luaL\_error(L,"fail to call function");  return -1;  }  printf("function end\n");  ReleaseMutex(Mutex);  return 0;  } |

|  |
| --- |
| Register into Lua |
| /\*This array has elements of type luaL\_Reg, which is a structure with two  fields: a string and a function pointer\*/  static const struct luaL\_reg mylibs [] = {  {"createlock", Lua\_endLock},  {"setlock", Lua\_setlock},  {"unlock", Lua\_unlock},  {"lockt",lua\_updateLock1},  {"lockf",lua\_updateLock2},  //{"deletelock", LuaLockFinal},  {"createthread", lua\_createThread},  {NULL, NULL} /\* sentinel \*/  };  /\*register the C function into Lua \_G-save global variables\*/  \_\_declspec(dllexport) int functionExport(lua\_State \*L)  {  //luaL\_openlib(L, "\_G", mylibs, 0);  luaL\_register(L,"\_G",mylibs);  return 1;  } |

# Test

|  |
| --- |
| **Test1** |
| require "stm"  --simple samples  function op1()  local done=false  y=createTransaction()  y.add("a",4)  y.add("b",5)  y.add("c",7)  while done==false do  y.a=y.a+y.b  y.b=y.c-y.b  y.c=y.c+1  done=y.commit()  end  end  function op2()  local done=false  z=createTransaction()  z.add("a",6)  z.add("b",8)  z.add("c",9)  while done==false do  z.a=z.a+z.b  z.b=z.c-z.b  z.c=z.c+1  done=z.commit()  end  end  i=1  while i<3 do  createT("op1")  i=i+1  end  --~ createT("op2")  --~ callfunc("op1")  --~ callfunc("op2")  printGtable() |

|  |
| --- |
| **Test2** |
| require "stm"  function op1()  local done=false  y=createTransaction()  y.add("a",4)  y.add("b",5)  y.add("c",7)  while done==false do  y.a=y.a+y.b  y.b=y.c-y.b  y.c=y.c+1  done=y.commit()  end  end  function op2()  local done=false  z=createTransaction()  z.add("d",6)  z.add("e",8)  z.add("f",9)  while done==false do  z.d=z.d+z.e  z.e=z.f-z.d  z.f=z.f+1  done=z.commit()  end  end  i=1  while i<50 do  createT("op1")  createT("op2")  i=i+1  end  --~ callfunc("op1")  --~ callfunc("op2")  printGtable() |

|  |
| --- |
| **Test3** |
| require "stm"  function op1()  local done=false  y=createTransaction()  y.add("b",1)  while done==false do  y.b=y.b+1  done=y.commit()  end  end  function op2()  local done=false  z=createTransaction()  z.add("b",1)  while done==false do  if z.b>1 then  z.b=z.b-1  end  done=z.commit()  end  end  i=1  while i<=3 do  createT("op1")  createT("op2")  i=i+1  end  --~ j=1  --~ while j<=3 do  --~ createT("op2")  --~ j=j+1  --~ end  printGtable() |