CODE EXAMPLE-5 FOR PCYNLITX

Developer:Erkam Murat Bozkurt

M.Sc in Control Systems Engineering

Example for smart pointers

1 BASIC DEFINITIONS

If you read before the definitions given in below, you can skip this page and look the example directly. In this document, the execution of the program is shown step by step in the slayt form. In order to do that, some basic definitions are given in below.

1.1 Thread_Server:

For each applications, a server class that is responsible for the management of the threads is produced automatically by the Pcynlitx platform. If the programmer does not determine a name for that class, Pcynlitx set the name of this class as "Thread_Server". The instance of that class creates the threads and it is named as "Server" in the following examples. In each thread creation, the addres of the server object is automatically passed to the each thread by means of a thread-specific data structure (*thds*). This data structure has been explained in below. Therefore, the server object is a container for every object that is shared between the threads. Each member of the server object is automatically reachable on the thread scopes. For more information, please read the tutorial.

1.2 TM_Client:

TM_Client is a special class and an instance of that class must be used on each function routines executed by the threads. The instance of the TM_Client class, which is named as "Manager" in the following examples, makes indirection to the object which is responsible from thread synchronization.

1.3 Namespace declaration

In pcynlitx platform, you can determine the namespace of the library constructed by the pcynlitx platform. If you does not enter any name to the namespace section to the platform before the library construction process, the default name which is "pcynlitx" is setted.

1.4 thds data structure:

The data structure that is named as "thds" holds the addresses of the variables which are shared between the threads. It is ab abbrevation for the term thread-specific data structure using in order to indicate the information that is passed only a certain thread.

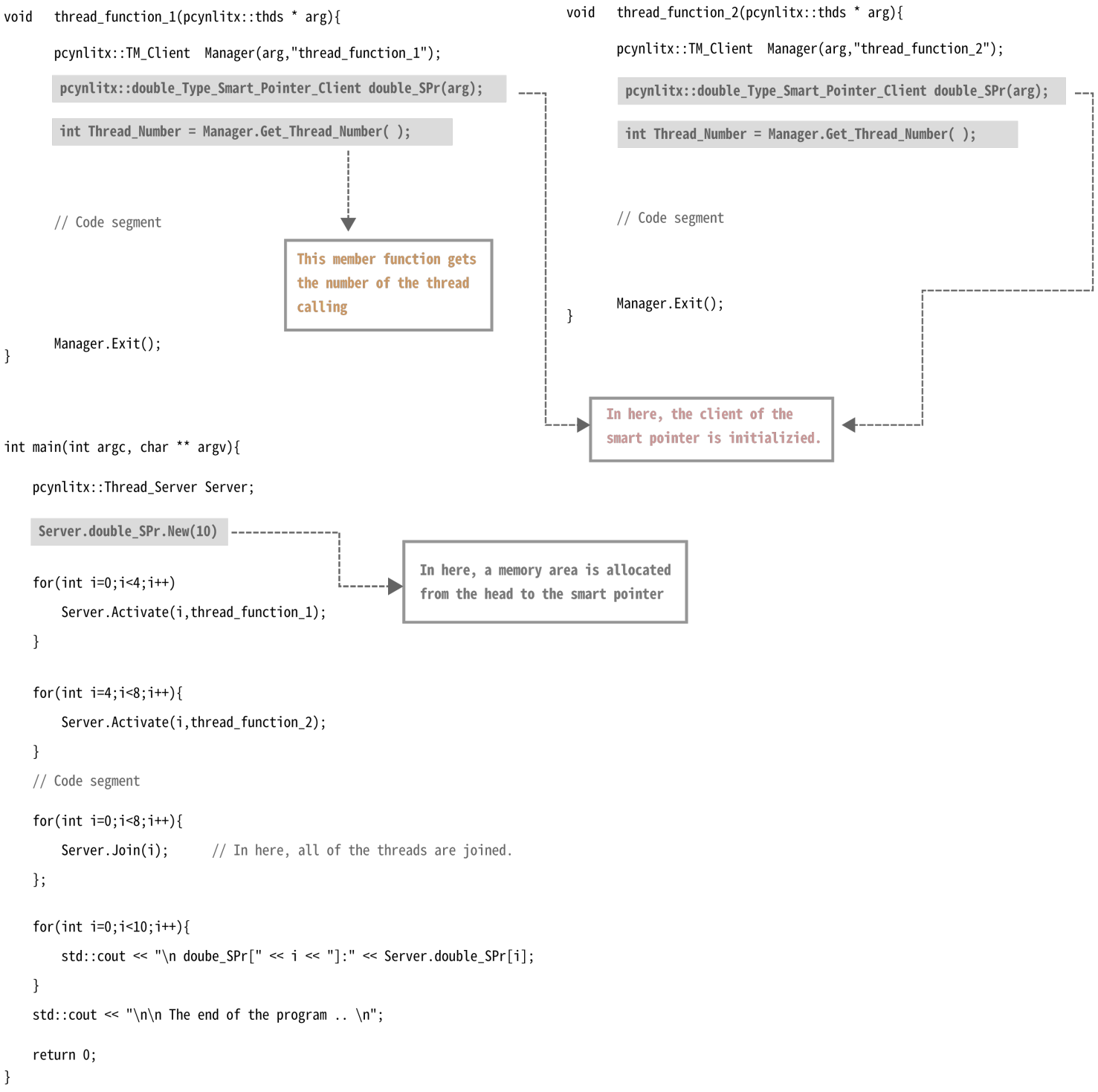

Figure 1: STEP-1

```
void thread_function_1(pcynlitx::thds * arg){

     pcynlitx::TM_Client Manager(arg,"thread_function_1");

     pcynlitx::double_Type_Smart_Pointer_Client double_SPr(arg);

     int Thread_Number = Manager.Get_Thread_Number( );

     if((Thread_Number == 0 )||(Thread_Number == 2))){

         for(int i=0;i<10;i++){

              Manager.lock( );

              double_SPr[i] = double_SPr[i] + 1;

              Manager.unlock( );
         }
     }

     if(((Thread_Number == 1 ) || (Thread_Number == 3))){

        for(int i=0;i<10;i++){

            Manager.lock( );

            double_SPr[i] = double_SPr[i] + 2;

            Manager.unlock( );
        }
     }

     // Code segment

     Manager.Exit();
}
```

These operations are typical mutex operations that locks and unlocks the mutex

the mutex operations prevent from the simultaneous axess to memory area. On the inside of the mutex, pthread_mutex function is called.

```
void thread_function_1(pcynlitx::thds * arg){

     pcynlitx::TM_Client Manager(arg,"thread_function_1");

     pcynlitx::double_Type_Smart_Pointer_Client double_SPr(arg);

     int Thread_Number = Manager.Get_Thread_Number( );

     if(((Thread_Number == 4 ) || (Thread_Number == 6))){

         for(int i=0;i<10;i++){

             Manager.lock( );

             double_SPr[i] = double_SPr[i] + 1;

             Manager.unlock( );
         }
     }

     if(((Thread_Number == 5 ) || (Thread_Number == 7))){

          for(int i=0;i<10;i++){

              Manager.lock( );

              double_SPr[i] = double_SPr[i] + 2;

              Manager.unlock( );
         }
     }

     // Code segment

     Manager.Exit();
}
```

```
int main(int argc, char ** argv){

    pcynlitx::Thread_Server Server;

    Server.double_SPr.New(10);

    for(int i=0;i<4;i++){

        Server.Activate(i,thread_function_1);
    }

    for(int i=4;i<8;i++){

        Server.Activate(i,thread_function_2);
    }
    // Code segment

    for(int i=0;i<8;i++){
        Server.Join(i);      // In here, all of the threads are joined.
    };

    for(int i=0;i<10;i++){

        std::cout << "\n doube_SPr[" << i << "]:" << Server.double_SPr[i];
    }

    std::cout << "\n\n The end of the program .. \n";

    return 0;
}
```

In this example, an array of memory is allocated to the smart pointer in the main thread's scope and the type of the array ise double.

Each element of the memory area incrases with +1 if the number of the thread is odd. Otherwise, increases with +2

In pcynlitx, you can determine your own computing scenario. You can de-allocate the memory area in any plays of the program. If you forget de-allocate the memory of the smart-ponter, it will be de-allocated by the destructor of the smart-pointer.

The client's of the smart pointer holds the adress of the smart pointer and makes indirection from the thread's scope to the main thread's scope. By this way, the programmers can use the client of the smart pointer just as the smart pointer itself.

Figure 2: STEP-2