CODE EXAMPLE-2 FOR PCYNLITX

Developer:Erkam Murat Bozkurt

M.Sc in Control Systems Engineering

Example for "barier_wait()" member function

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.

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

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


     // Code segment

     Manager.barier_wait();

     // Code segment

     Manager.Exit();
}
```

```
void thread_function_2(pcynlitx::thds * arg){

     pcynlitx::TM_Client Manager(arg,"thread_function_2");


     // Code segment

     Manager.barier_wait();

     // Code segment

     Manager.Exit();
}
```

Thread [0] starts its execution and when it reaches the barier_wait( ) function, it stops.

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

     pcynlitx::Thread_Server Server;


     for(int i=0;i<2;i++){
          Server.Activate(i,thread_function_1);
     }


     for(int i=2;i<4;i++){
          Server.Activate(i,thread_function_2);
     }


     for(int i=0;i<4;i++){
          Server.Join(i);      // In here, all of the threads are joined.
     };


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

     return 0;
}
```

Thread [0]

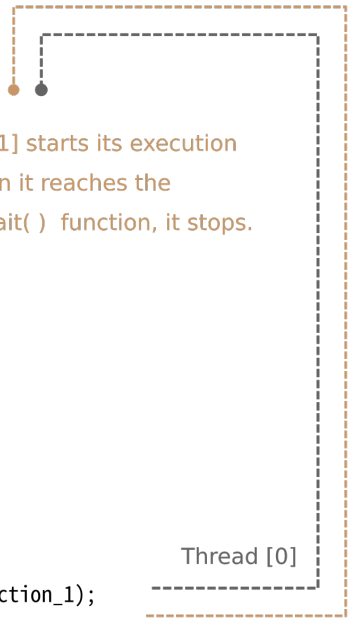
Figure 1: STEP-1

```
void thread_function_1(pcynlitx::thds * arg){
     pcynlitx::TM_Client Manager(arg,"thread_function_1");

     // Code segment
     Manager.barier_wait();
     // Code segment
     Manager.Exit();
}
```

```
void thread_function_2(pcynlitx::thds * arg){
     pcynlitx::TM_Client Manager(arg,"thread_function_2");

     // Code segment
     Manager.barier_wait();
     // Code segment
     Manager.Exit();
}
```




```
int main(int argc, char ** argv){
    pcynlitx::Thread_Server Server;

    for(int i=0;i<2;i++){
        Server.Activate(i,thread_function_1);
    }

    for(int i=2;i<4;i++){
        Server.Activate(i,thread_function_2);
    }

    for(int i=0;i<4;i++){
        Server.Join(i);      // In here, all of the threads are joined.
    };

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

    return 0;
}
```

Figure 2: STEP-2

```
void thread_function_1(pcynlitx::thds * arg){
     pcynlitx::TM_Client Manager(arg,"thread_function_1");

     // Code segment
     Manager.barier_wait();
     // Code segment

     Manager.Exit();
}
```

```
void thread_function_2(pcynlitx::thds * arg){
     pcynlitx::TM_Client Manager(arg,"thread_function_2");

     // Code segment
     Manager.barier_wait();
     // Code segment

     Manager.Exit();
}
```

Thread [2] starts its execution and when it reaches the barier wait( ) function, it stops.

```
int main(int argc, char ** argv){
     pcynlitx::Thread_Server Server;

     for(int i=0;i<2;i++){
          Server.Activate(i,thread_function_1);
     }

     for(int i=2;i<4;i++){
          Server.Activate(i,thread_function_2);
     }

     for(int i=0;i<4;i++){
          Server.Join(i);      // In here, all of the threads are joined.
     };

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

     return 0;
}
```

Thread [0]

Thread [1]

Thread [2]

Figure 3: STEP-3

```
void thread_function_1(pcynlitx::thds * arg){
     pcynlitx::TM_Client Manager(arg,"thread_function_1");

     // Code segment
     Manager.barier_wait();
     // Code segment
     Manager.Exit();
}
```

```
void thread_function_2(pcynlitx::thds * arg){
     pcynlitx::TM_Client Manager(arg,"thread_function_2");

     // Code segment
     Manager.barier_wait();
     // Code segment
     Manager.Exit();
}
```

when the last thread (thread [3]) reaches this point, it rescues the other threads on the process.

```
int main(int argc, char ** argv){
     pcynlitx::Thread_Server Server;

     for(int i=0;i<2;i++){
          Server.Activate(i,thread_function_1);
     }

     for(int i=2;i<4;i++){
          Server.Activate(i,thread_function_2);
     }

     for(int i=0;i<4;i++){
          Server.Join(i);      // In here, all of the threads are joined.
     };

     std::cout << "\n\n The end of the program .. \n";
     return 0;
}
```

Thread [0]

Thread [1]

Thread [2]

Thread [3]

Figure 4: STEP-4