< 2018 학년도 2 학기 – 운영체제(Operating System) 과제 #2 >

이름 : 유기성

학번 : 20120139

제출일 : 2018 년 12 월 11 일 (화)

1. Definition of the Problem

posix_timer 를 활용해서
자신만의 타이머, user_defined_timer 를 만들어보는 것이 이번 과제이다.

posix_timer 의 주기는 10ms 로 하며
10ms 마다 user_defined_timer 의 time_left_to_invoke 를 일정하게 감소시켜서
주기적으로 thread 에 conditional signal 을 보내서 wake-up 하는 것이 핵심이다.

2. Architecture of the Solution

구현 과정을 요약하면 다음과 같다.

1. 필요한 변수를 모두 선언한다. pthread_mutex_t, pthread_cond_t 는 몇 개가 필요하고 스레드를 관리하는 TCB (Thread Control Block) 에는 어떤 멤버변수가 필요한지, posix thread 를 만들때 넘겨줄 argument 는 어떤 멤버함수를 지닌 구조체로 설정해야 하는지, posix_timer 의 주기는 얼마로 설정할지, user_defined_timer 는 어떤식으로 동작하도록 만들지 등을 결정한다.
2. 위에서 선언한 변수를 모두 초기화한다.
3. 스레드를 여러개 생성한다. 이 스레드는 모두 user-defined-timer 에 의해서 관리된다.
   3-1. thread_to_be_managed 가 바로 스레드의 원형이다.
   각각의 스레드는 자신의 thread ID 와 현재시간을 콘솔로 출력하며
   스케쥴링이 제대로 이루어지고 있음을 보여준다.
   3-2. tt_thread_register 에서는 TCB 에 스레드 자신의 정보를 담는다.
   3-3. tt_thread_wait_invocation 에서는 user_defined_timer 가 보내는 (periodic_handler 함수 내부의) pthread_cond_signal 이 실행될 때까지 대기하는 pthread_cond_wait 를 호출한다.
4. posix_timer 를 생성한다. posix_timer 는 하나이고 이를 활용하는 user_defined_timer 는 여러개일 수도 있다.
   4-1. posix_timer 는 SIGEV_THREAD 타입이며 sigev_notify_function 으로 periodic_handler( ) 함수를 선택한다. sigev_value.sival_ptr 를 통해서 periodic_handler 함수에 argument 로 넘겨줄 데이터를 void* 타입으로 지정할 수 있다.
   4-2. periodic_handler 의 내부 구현은 이렇다.
   TCB 는 세 가지 변수로 구성된다.
   1) period : 초기에 설정한 주기 (실제 시간이 아니라 가상의 시간)
   2) time_left_to_invoke : 한 주기가 끝날 때까지 남은 시간 (이것은 가상의 시간)
   3) tid : cond_signal 을 보낼 스레드를 지정한다.
   일정한 시간마다 이 함수가 실행되며 TCB_array 에 있는 모든 TCB 의 time_left_to_invoke 를 조정한다. 그렇게 매 주기가 끝날 때마다 cond_signal 을 보내서 생성된 스레드들을 실행한다.

3. Encountered Obstacles and Realizations

처음에는 API 라면 당연히 OOP 방식으로 구현하는 것이 옳다고 믿으며 클래스로 구조화해서 구현하고자 했다. 그러나 클래스 멤버 변수로 구현할 경우에는 크게 두 가지 부분에서 문제점이 있었다.
첫번째로 pthread_create 에 넘겨줄 때에 타입이 달라서 [type “void* (&CLASS_TYPE::thread_to_be_managed) (void*)” cannot be converted to type “void* (*) (void*)”] 와 같은 에러메세지에 직면했다.
모든 클래스 멤버 변수는 implicitly 첫번째 argument 로 자기 자신을 갖는데, 이 때문에 namespace 나 class 개념없이 global static anonymous 로 동작하는 C 언어와 완벽하게 호환이 되지 않는다. 클래스 멤버 변수의 첫번째 인자가 자기 자신이라는 것은 python 에서 객체 지향 프로그래밍 방식으로 코드를 구현하면 항상 클래스 메소드의 parameter list 가 (self,)로 시작하는 것으로 확인할 수 있다.
두번째로 sigev_notify_function 의 함수를 지정할 때에도 [type “void (&CLASS_TYPE::periodic_handler) (sigval_t)” cannot be converted to type “void (*) (sigval_t)”] 라는 에러메세지를 만나며 난관에 봉착한다.
이런 문제점을 우회하기 위해서 static member function 을 만들었지만 static function 은 내부에서 다른 static function 과 static member variables 만을 다룰 수 있기에 제약이 많았다.

결국엔 모든 함수를 global namespace 에 구현하기로 결정했다. class 로 바꿔서 구현하는 것은 나중에 다시 도전해볼 것이다.
대신에 코드를 최대한 깔끔하게 정리하기 위해서
‘하나의 파일은 연관성이 큰 함수만 포함한다’ 와
‘하나의 함수는 하나의 기능을 한다’ 라는 문장을 최대한 의식하며 설계하고자 노력했다. 물론 프로그램 전체의 모든 함수들이 side effect 를 최소화했다. 함수 자체의 길이도 최대한 짧게 줄여서 가독성을 높였다.

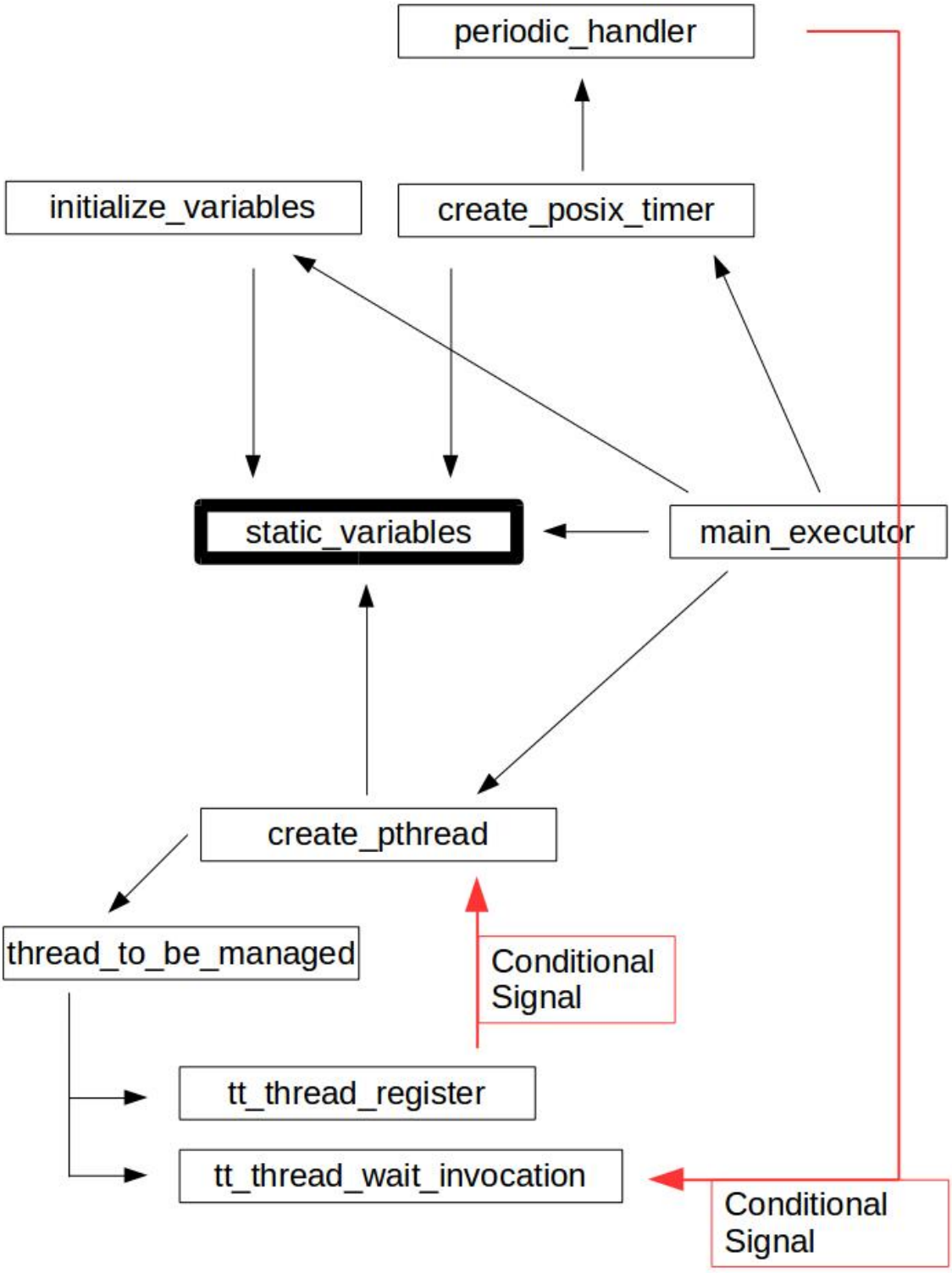
4. Source Codes (with comments)

프로젝트 구조

```
.
├── CMakeLists.txt
├── include
│   ├── create_posix_timer.hpp
│   ├── create_pthread.hpp
│   ├── initialize_variables.hpp
│   ├── main_executor.hpp
│   ├── periodic_handler.hpp
│   ├── print_functions.hpp
│   ├── static_variables.hpp
│   ├── thread_to_be_managed.hpp
│   ├── tt_thread_register.hpp
│   └── tt_thread_wait_invocation.hpp
├── Screenshot from 2018-12-07 18-58-24.png
└── src
    ├── create_posix_timer.cpp
    ├── create_pthread.cpp
    ├── initialize_variables.cpp
    ├── main_executor.cpp
    ├── periodic_handler.cpp
    ├── print_functions.cpp
    ├── static_variables.cpp
    ├── thread_to_be_managed.cpp
    ├── tt_thread_register.cpp
    └── tt_thread_wait_invocation.cpp

2 directories, 22 files
```

Header Files Dependency Graph



why I choose smart pointers

```
void initialize_variables(void)
```

모든 변수를 std::shared_ptr 타입에 저장한다 .

C++ 스마트 포인터의 장점
1) heap allocated 객체의 생성과 소멸을 안전하게 자동화 해 준다 . 예상치 못한 에러가 발생하는 상황에서도 메모리 누수가 발생하는 것을 방지할 수 있다 .
2) 메모리 최적화가 잘 되어 있어서 raw pointer 보다 효율적으로 작동한다 .
3) 스택 메모리 영역에 생성했더라도 나중에 reset( ) 을 호출하므로써 실행 중에 소멸할 수도 있다 .

IDX_Table

```
std::unordered_map<pthread_t, size_t> IDX_Table
```

tt_thread_wait_invocation( ) 함수는 pthread_t 타입 변수를 argument 로 받아서 스레드에 해당하는 conditional signal 을 기다리는 작업을 한다 . 해당 스레드 변수가 어떤 conditional variable 과 연관이 있는지 빠르게 확인하기 위해서 look-up 테이블을 생성했다 .

API_Mutex & API_Cond

pthread_mutex_t API_Mutex

std::vector<pthread_cond_t> API_Cond_Array

프로그램을 실행시키면서 posix_timer 에 notify_function 으로 “ void periodic_handler(sigval_t)” 함수를 지정했다 . itimerspec 에 지정한 주기마다 커널은 이 함수를 실행할 것이므로 프로그램 작성자는 이 함수에 자신이 원하는 코드를 삽입한다 .

periodic_handler 함수에는 posix_timer 를 활용하는 ‘ user defined timer’ 가 구현되어 있다 . 타이머가 함수를 실행할 때마다 time_left_to_invoke 를 감소시켜서 0 이하의 값이 되는 순간 API_Cond_Array 에 conditional signal 을 보낸다 .

이 conditional signal 은 pthread_create( ) 함수를 통해서 생성된 스레드 안의
“void tt_thread_wait_invocation(pthread_t)” 함수에서 호출된 cond_wait 지점에 도달한다 . 이를 통해서 블록 (blocked) 되어 있던 스레드가 다시 running 상태가 된다 .

TCB_position_Mutex & TCB_position_Cond

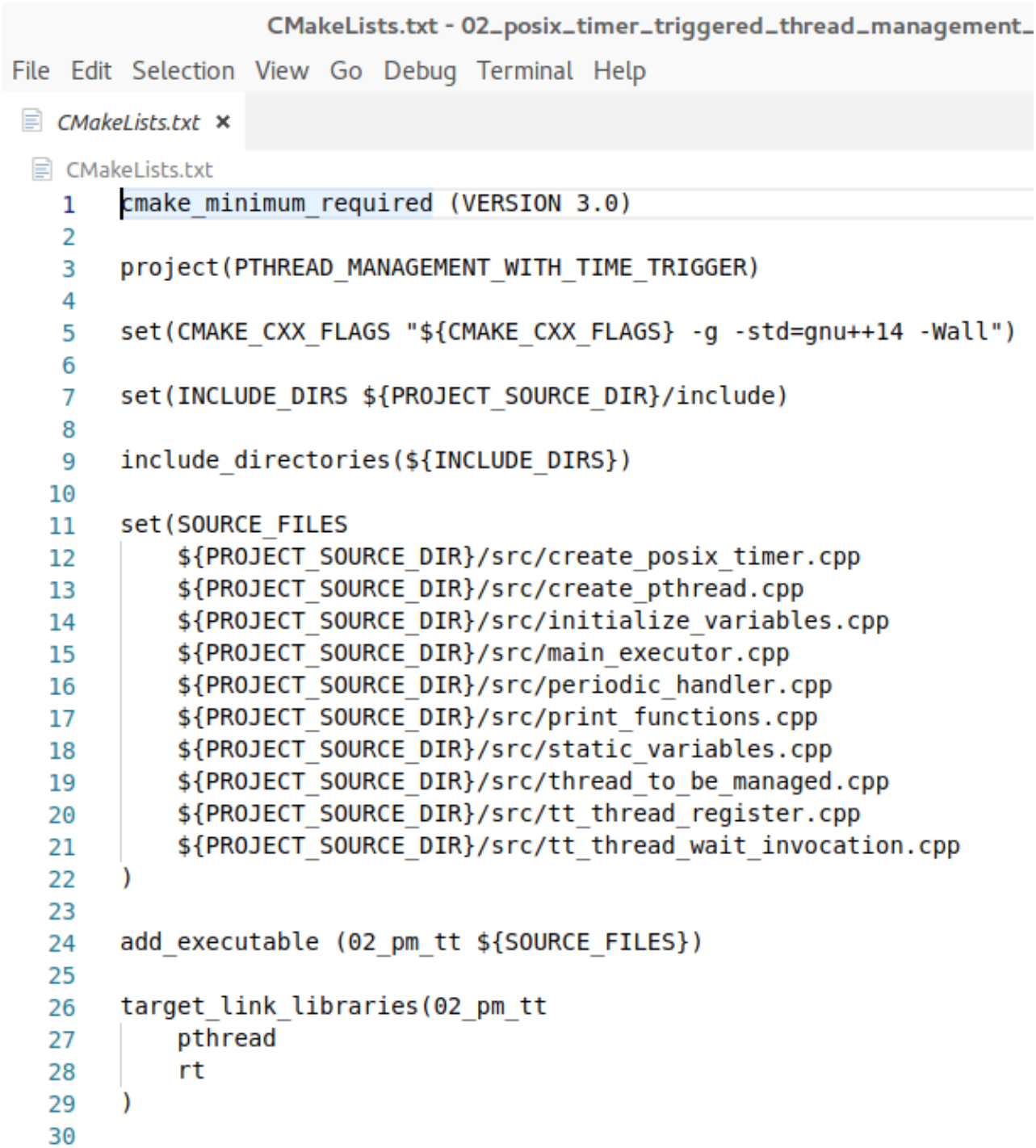
pthread_mutex_t TCB_position_Mutex

pthread_cond_t TCB_position_Cond

TCB_position 변수는
create_new_thread( ) 를 실행하는 메인 스레드와
pthread_create 에서 만들어진 스레드 사이에서 공유되는 변수이다 . 추가로 생성된 스레드는 tt_thread_register 에서 TCB 변수에 적절한 값을 저장한 뒤에 TCB_position 을 +1 시켜서 TCB 배열 내 적절한 위치를 가리키도록 만든다 .

이 critical section 에서 충돌이 일어나는 것을 방지하기 위해서 TCB_position_Cond 를 생성했다 .
create_new_thread 함수가 새로운 스레드를 만든 뒤에는 이 함수가 std::vector<TCB> 에 정상적으로 등록할 때까지 cond_wait 지점에서 block 상태로 대기한다 .

CMakeLists.txt



<build>
$ mkdir build && cd build && cmake ..
$ make
$ ./02_pm_tt

<clean>
$ cd .. && rm -rf build/

static_variables.hpp

File Edit Selection View Go Debug Terminal Help

static_variables.hpp

include ▸ static_variables.hpp ▸ ...

```hpp
#ifndef STATIC_VARIABLES__83bfa9ef_1e45_4e25_81e8_ecac1c6a3ad1__HPP___
#define STATIC_VARIABLES__83bfa9ef_1e45_4e25_81e8_ecac1c6a3ad1__HPP___

#include <memory>
#include <vector>
#include <unordered_map>

#include <pthread.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>

#define MAX_THREAD_NUM      5

#define THOUSAND            1000UL
#define MILLION             1000000UL

using event = struct sigevent;
using t_spec = struct itimerspec;

struct TCB
{
    pthread_t       tid;
    long            period;
    long            time_left_to_invoke;
    // size_t          self_TCB_idx;
};
using TCB = struct TCB;

struct timer_args
{
    size_t ptimer_period;
};
using timer_args = struct timer_args;

struct thread_args
{
    size_t period;
};
using thread_args = struct thread_args;



#endif /* STATIC_VARIABLES__83bfa9ef_1e45_4e25_81e8_ecac1c6a3ad1__HPP___ */
```

(Global Scope) Ln 17, Col 1

static_variables.cpp

static_variables.cpp

src > static_variables.cpp > ...

```cpp
#include "../include/static_variables.hpp"

std::shared_ptr<pthread_mutex_t> API_Mutex;
std::shared_ptr<std::vector<pthread_cond_t>> API_Cond_Array;

std::shared_ptr<timer_args> TM_Arg;
std::shared_ptr<timer_t> Timer;

std::shared_ptr<std::vector<thread_args>> THR_Args;
std::shared_ptr<std::vector<pthread_t>> Threads;

std::shared_ptr<std::vector<TCB>> TCB_array;
std::shared_ptr<pthread_mutex_t> TCB_position_Mutex;
std::shared_ptr<pthread_cond_t> TCB_position_Cond;
std::shared_ptr<size_t> TCB_position;

std::shared_ptr<std::unordered_map<pthread_t, size_t>> IDX_Table;

```

initialize_variables.hpp

initialize_variables.hpp

include ▸ initialize_variables.hpp ▸ ...

```
1   #ifndef INITIALIZE_VARIABLES__d447ad55_e6d3_428a_b3b4_c490272dcad9__HPP___
2   #define INITIALIZE_VARIABLES__d447ad55_e6d3_428a_b3b4_c490272dcad9__HPP___
3
4   #include "static_variables.hpp"
5
6
7
8   void initialize_variables(void);
9
10
11
12  #endif /* INITIALIZE_VARIABLES__d447ad55_e6d3_428a_b3b4_c490272dcad9__HPP___ */
```

initialize_variables.cpp

initialize_variables.cpp

src ▸ initialize_variables.cpp ▸ ...

```
5   extern std::shared_ptr<pthread_mutex_t> API_Mutex;
6   extern std::shared_ptr<std::vector<pthread_cond_t>> API_Cond_Array;
7
8   extern std::shared_ptr<timer_args> TM_Arg;
9   extern std::shared_ptr<timer_t> Timer;
10
11  extern std::shared_ptr<std::vector<thread_args>> THR_Args;
12  extern std::shared_ptr<std::vector<pthread_t>> Threads;
13
14  extern std::shared_ptr<std::vector<TCB>> TCB_array;
15  extern std::shared_ptr<pthread_mutex_t> TCB_position_Mutex;
16  extern std::shared_ptr<pthread_cond_t> TCB_position_Cond;
17  extern std::shared_ptr<size_t> TCB_position;
18
19  extern std::shared_ptr<std::unordered_map<pthread_t, size_t>> IDX_Table;
20
21
22
23  void initialize_variables(void)
24  {
25      API_Mutex = std::make_shared<pthread_mutex_t>();
26      pthread_mutex_init(&(*API_Mutex), nullptr);
27      API_Cond_Array = std::make_shared<std::vector<pthread_cond_t>>();
28      API_Cond_Array->resize(MAX_THREAD_NUM);
29      for (auto& cond_var : *API_Cond_Array) {
30          pthread_cond_init(&cond_var, nullptr);
31      }
32
33      TM_Arg = std::make_shared<timer_args>();
34      Timer = std::make_shared<timer_t>();
35
36      THR_Args = std::make_shared<std::vector<thread_args>>();
37      THR_Args->resize(MAX_THREAD_NUM);
38      Threads = std::make_shared<std::vector<pthread_t>>();
39      Threads->resize(MAX_THREAD_NUM);
40
41      TCB_array = std::make_shared<std::vector<TCB>>();
42      TCB_array->resize(MAX_THREAD_NUM);
43      TCB_position_Mutex = std::make_shared<pthread_mutex_t>();
44      pthread_mutex_init(&(*TCB_position_Mutex), nullptr);
45      TCB_position_Cond = std::make_shared<pthread_cond_t>();
46      pthread_cond_init(&(*TCB_position_Cond), nullptr);
47      TCB_position = std::make_shared<size_t>();
48      *TCB_position = 0;
49
50      IDX_Table = std::make_shared<std::unordered_map<pthread_t, size_t>>();
51      IDX_Table->reserve(MAX_THREAD_NUM);
52  }
```

create_pthread.hpp

h++ *create_pthread.hpp* ×

include ▸ h++ create_pthread.hpp ▸ ...

```
1   #ifndef CREATE_PTHREAD__341b9891_7438_47b0_b275_6ddd9887c68a__HPP___
2   #define CREATE_PTHREAD__341b9891_7438_47b0_b275_6ddd9887c68a__HPP___
3
4   #include <wait.h>
5
6   #include "static_variables.hpp"
7   #include "thread_to_be_managed.hpp"
8   #include "print_functions.hpp"
9
10
11
12  void add_pthread_arguments(size_t);
13  void create_new_thread(void);
14
15
16
17  #endif /* CREATE_PTHREAD__341b9891_7438_47b0_b275_6ddd9887c68a__HPP___ */
```

create_pthread.cpp

create_pthread.cpp

src > create_pthread.cpp > ...

```cpp
#include "../include/create_pthread.hpp"



extern std::shared_ptr<pthread_mutex_t> API_Mutex;
extern std::shared_ptr<std::vector<pthread_cond_t>> API_Cond_Array;

extern std::shared_ptr<timer_args> TM_Arg;
extern std::shared_ptr<timer_t> Timer;

extern std::shared_ptr<std::vector<thread_args>> THR_Args;
extern std::shared_ptr<std::vector<pthread_t>> Threads;

extern std::shared_ptr<std::vector<TCB>> TCB_array;
extern std::shared_ptr<pthread_mutex_t> TCB_position_Mutex;
extern std::shared_ptr<pthread_cond_t> TCB_position_Cond;
extern std::shared_ptr<size_t> TCB_position;

extern std::shared_ptr<std::unordered_map<pthread_t, size_t>> IDX_Table;



void add_pthread_arguments(size_t i)
{
    THR_Args->at(i).period = static_cast<size_t>(i+1) * THOUSAND;
}

void create_new_thread(void)
{
    while (*TCB_position < MAX_THREAD_NUM)
    {
        print(std::to_string(*TCB_position + 1) + "-th for LOOP");
        add_pthread_arguments(*TCB_position);

        if (pthread_create(
            &(Threads->at(*TCB_position)),
            nullptr,
            thread_to_be_managed,
            &(THR_Args->at(*TCB_position)))
                != 0) {
            print("pthread_create() failed TT TT");
        }

        pthread_mutex_lock(&(*TCB_position_Mutex));
        pthread_cond_wait(&(*TCB_position_Cond), &(*TCB_position_Mutex));
        pthread_mutex_unlock(&(*TCB_position_Mutex));
    }
}
```

thread_to_be_managed.hpp

thread_to_be_managed.hpp

include ▸ thread_to_be_managed.hpp ▸ ...

```
1   #ifndef THREAD_TO_BE_MANAGED__e1bccf26_16c4_4d16_9c50_0772f73122bb__HPP___
2   #define THREAD_TO_BE_MANAGED__e1bccf26_16c4_4d16_9c50_0772f73122bb__HPP___
3
4   #include <time.h>
5
6   #include "static_variables.hpp"
7   #include "tt_thread_register.hpp"
8   #include "tt_thread_wait_invocation.hpp"
9   #include "print_functions.hpp"
10
11
12
13  void* thread_to_be_managed(void*);
14
15
16
17  #endif /* THREAD_TO_BE_MANAGED__e1bccf26_16c4_4d16_9c50_0772f73122bb__HPP___ */
```

thread_to_be_managed.cpp

thread_to_be_managed.cpp ×

src ▸ thread_to_be_managed.cpp ▸ ...

```cpp
#include "../include/thread_to_be_managed.hpp"



extern std::shared_ptr<pthread_mutex_t> API_Mutex;
extern std::shared_ptr<std::vector<pthread_cond_t>> API_Cond_Array;

extern std::shared_ptr<timer_args> TM_Arg;
extern std::shared_ptr<timer_t> Timer;

extern std::shared_ptr<std::vector<thread_args>> THR_Args;
extern std::shared_ptr<std::vector<pthread_t>> Threads;

extern std::shared_ptr<std::vector<TCB>> TCB_array;
extern std::shared_ptr<pthread_mutex_t> TCB_position_Mutex;
extern std::shared_ptr<pthread_cond_t> TCB_position_Cond;
extern std::shared_ptr<size_t> TCB_position;

extern std::shared_ptr<std::unordered_map<pthread_t, size_t>> IDX_Table;



void* thread_to_be_managed(void* arg)
{
    auto period_v = reinterpret_cast<thread_args*>(arg)->period;
    auto tid = pthread_self();

    tt_thread_register(period_v, tid);

    while (1)
    {
        tt_thread_wait_invocation(tid);

        time_t curtime;
        time(&curtime);
        std::string S = " ";
        S += std::to_string(IDX_Table->at(tid) + 1) + "-th thread runniing... \t";
        S += "|| \t" + std::string(asctime(localtime(&curtime)));
        print(S, "");
    }
}

```

tt_thread_register.hpp

tt_thread_register.hpp ×

include › tt_thread_register.hpp › ...

```
1   #ifndef TT_THREAD_REGISTER__2622de81_8607_437b_be05_2c69eb066ae0__HPP___
2   #define TT_THREAD_REGISTER__2622de81_8607_437b_be05_2c69eb066ae0__HPP___
3
4   #include "static_variables.hpp"
5   #include "print_functions.hpp"
6
7
8
9   void tt_thread_register(size_t, pthread_t);
10
11
12
13  #endif /* TT_THREAD_REGISTER__2622de81_8607_437b_be05_2c69eb066ae0__HPP___ */
```

tt_thread_register.cpp

tt_thread_register.cpp

src > tt_thread_register.cpp > ...

```cpp
#include "../include/tt_thread_register.hpp"



extern std::shared_ptr<pthread_mutex_t> API_Mutex;
extern std::shared_ptr<std::vector<pthread_cond_t>> API_Cond_Array;

extern std::shared_ptr<timer_args> TM_Arg;
extern std::shared_ptr<timer_t> Timer;

extern std::shared_ptr<std:: class std::vector<pthread_t>;
extern std::shared_ptr<std::vector<pthread_t>> Threads;

extern std::shared_ptr<std::vector<TCB>> TCB_array;
extern std::shared_ptr<pthread_mutex_t> TCB_position_Mutex;
extern std::shared_ptr<pthread_cond_t> TCB_position_Cond;
extern std::shared_ptr<size_t> TCB_position;

extern std::shared_ptr<std::unordered_map<pthread_t, size_t>> IDX_Table;



void tt_thread_register(size_t new_period, pthread_t tid)
{
    pthread_mutex_lock(&(*API_Mutex));

    pthread_mutex_lock(&(*TCB_position_Mutex));
    TCB_array->at(*TCB_position).period = new_period;
    TCB_array->at(*TCB_position).time_left_to_invoke = new_period;
    TCB_array->at(*TCB_position).tid = tid;

    (*IDX_Table)[tid] = *TCB_position;
    ++(*TCB_position);
    pthread_cond_signal(&(*TCB_position_Cond));
    pthread_mutex_unlock(&(*TCB_position_Mutex));

    pthread_mutex_unlock(&(*API_Mutex));
}
```

tt_thread_wait_invocation.hpp

tt_thread_wait_invocation.hpp

include ▸ tt_thread_wait_invocation.hpp ▸ ...

```
1   #ifndef TT_THREAD_WAIT_INVOCATION__4643ac13_fc01_40b8_981c_e2bd22cb03c3__HPP___
2   #define TT_THREAD_WAIT_INVOCATION__4643ac13_fc01_40b8_981c_e2bd22cb03c3__HPP___
3
4
5
6   #include "static_variables.hpp"
7   #include "print_functions.hpp"
8
9   void tt_thread_wait_invocation(pthread_t);
10
11
12
13  #endif /* TT_THREAD_WAIT_INVOCATION__4643ac13_fc01_40b8_981c_e2bd22cb03c3__HPP___ */
```

tt_thread_wait_invocation.cpp

tt_thread_wait_invocation.cpp

src > tt_thread_wait_invocation.cpp > ...

```cpp
#include "../include/tt_thread_wait_invocation.hpp"



extern std::shared_ptr<pthread_mutex_t> API_Mutex;
extern std::shared_ptr<std::vector<pthread_cond_t>> API_Cond_Array;

extern std::shared_ptr<timer_args> TM_Arg;
extern std::shared_ptr<timer_t> Timer;

extern std::shared_ptr<std::vector<thread_args>> THR_Args;
extern std::shared_ptr<std::vector<pthread_t>> Threads;

extern std::shared_ptr<std::vector<TCB>> TCB_array;
extern std::shared_ptr<pthread_mutex_t> TCB_position_Mutex;
extern std::shared_ptr<pthread_cond_t> TCB_position_Cond;
extern std::shared_ptr<size_t> TCB_position;

extern std::shared_ptr<std::unordered_map<pthread_t, size_t>> IDX_Table;



void tt_thread_wait_invocation(pthread_t tid)
{
    auto idx = IDX_Table->at(tid);

    pthread_mutex_lock(&(*API_Mutex));
    pthread_cond_wait(&(API_Cond_Array->at(idx)), &(*API_Mutex));
    pthread_mutex_unlock(&(*API_Mutex));
}
```

print_function.hpp

print_functions.hpp

include › print_functions.hpp › ...

```hpp
#ifndef PRINT_FUNCTIONS__237216e1_b6da_45ec_bced_fe6d10f199a6__HPP___
#define PRINT_FUNCTIONS__237216e1_b6da_45ec_bced_fe6d10f199a6__HPP___

#include <iostream>

void print(void);
void perr(void);

template <typename T>
void print(const T& msg, const std::string& end="\n")
{
    std::cout << msg << end;
}

template <typename T>
void perr(const T& msg, const std::string& end="\n")
{
    std::cerr << msg << end;
}

#endif /* PRINT_FUNCTIONS__237216e1_b6da_45ec_bced_fe6d10f199a6__HPP___ */
```

print_function.cpp

print_functions.cpp

src › print_functions.cpp › ...

```cpp
#include "../include/print_functions.hpp"

void print(void)
{
    std::cout << "\n";
}

void perr(void)
{
    std::cerr << "\n";
}
```

main_executor.hpp

main_executor.hpp

include ▸ main_executor.hpp ▸ ...

```hpp
#ifndef MAIN_EXECUTOR__1021f53f_4b06_4297_ab63_d0a009894220__HPP___
#define MAIN_EXECUTOR__1021f53f_4b06_4297_ab63_d0a009894220__HPP___

#include <cstdlib>
#include <clocale>

#include <unistd.h>

#include "static_variables.hpp"
#include "initialize_variables.hpp"
#include "create_posix_timer.hpp"
#include "create_pthread.hpp"
#include "print_functions.hpp"



void main_executor(void);



#endif /* MAIN_EXECUTOR__1021f53f_4b06_4297_ab63_d0a009894220__HPP___ */
```

main_executor.cpp

main_executor.cpp

src ▸ main_executor.cpp ▸ ...

```cpp
#include "../include/main_executor.hpp"

int main()
{
    main_executor();

    return 0;
}

void main_executor(void)
{
    std::setlocale(LC_TIME, "ko_KR.utf8");

    print("inside main_executor( )");

    initialize_variables();

    create_new_thread();
    print("posix thread create OK OK OK");

    create_posix_timer();
    print("posix_timer is created OK");

    print("pause HERE");
    while(1) {
        pause();
    }
}
```

5. Results Capture