< 2018 학년도 2 학기 – 운영체제(Operating System) 과제 #3 >

이름 : 유기성

학번 : 20120139

제출일 : 2018 년 12 월 11 일 (화)

1. Definition of the Problem

이번 과제는 앞에서 제작한 User Defined Timer 를 활용해서
Pseudo Blood Pressure Detector 를 만드는 것이다.

병원에서 환자실에 누워있으면 흔히 볼 수 있는 혈압측정계의 텍스트 버전이라고 생각하면 된다.

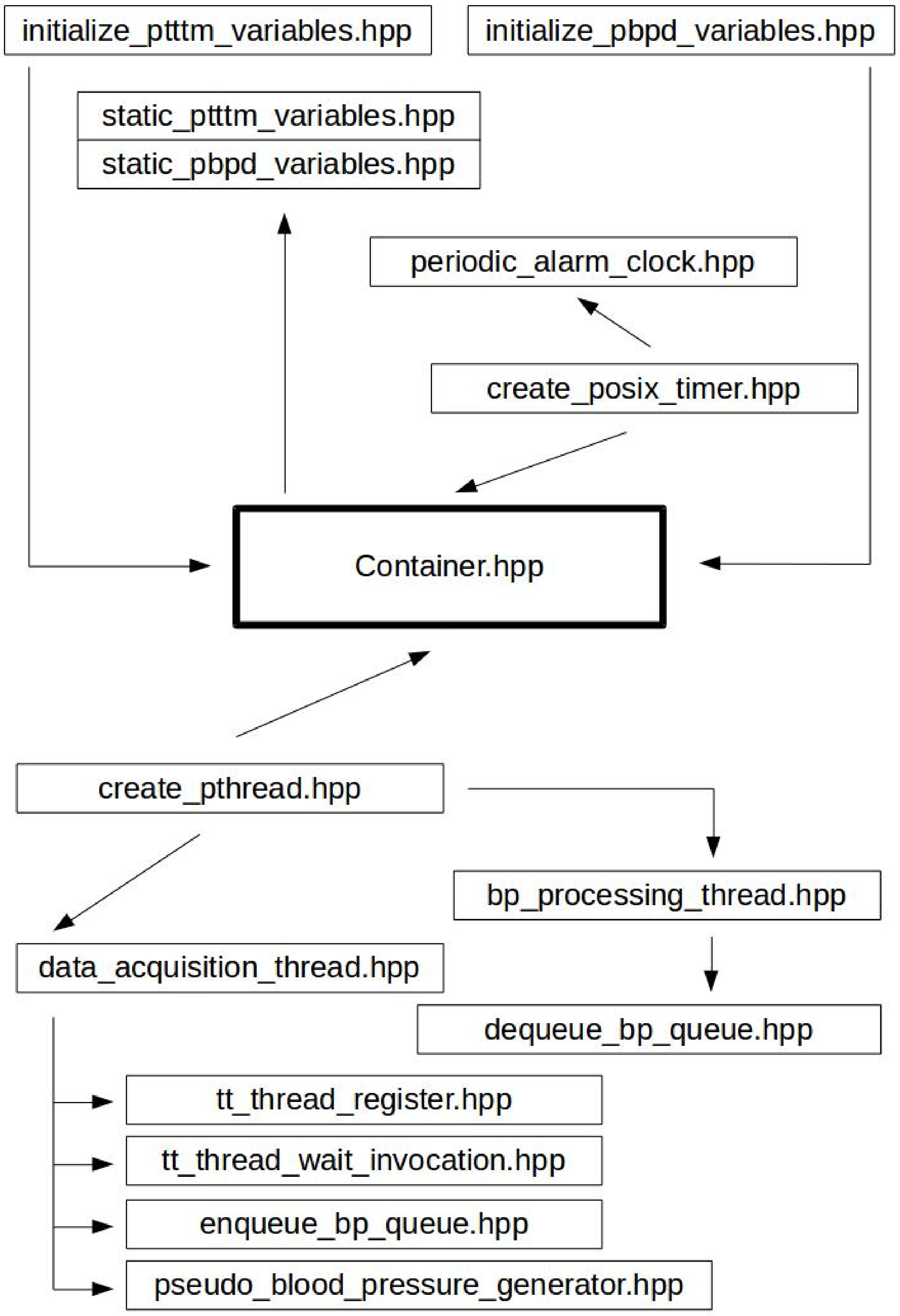
2. Architecture of the Solution

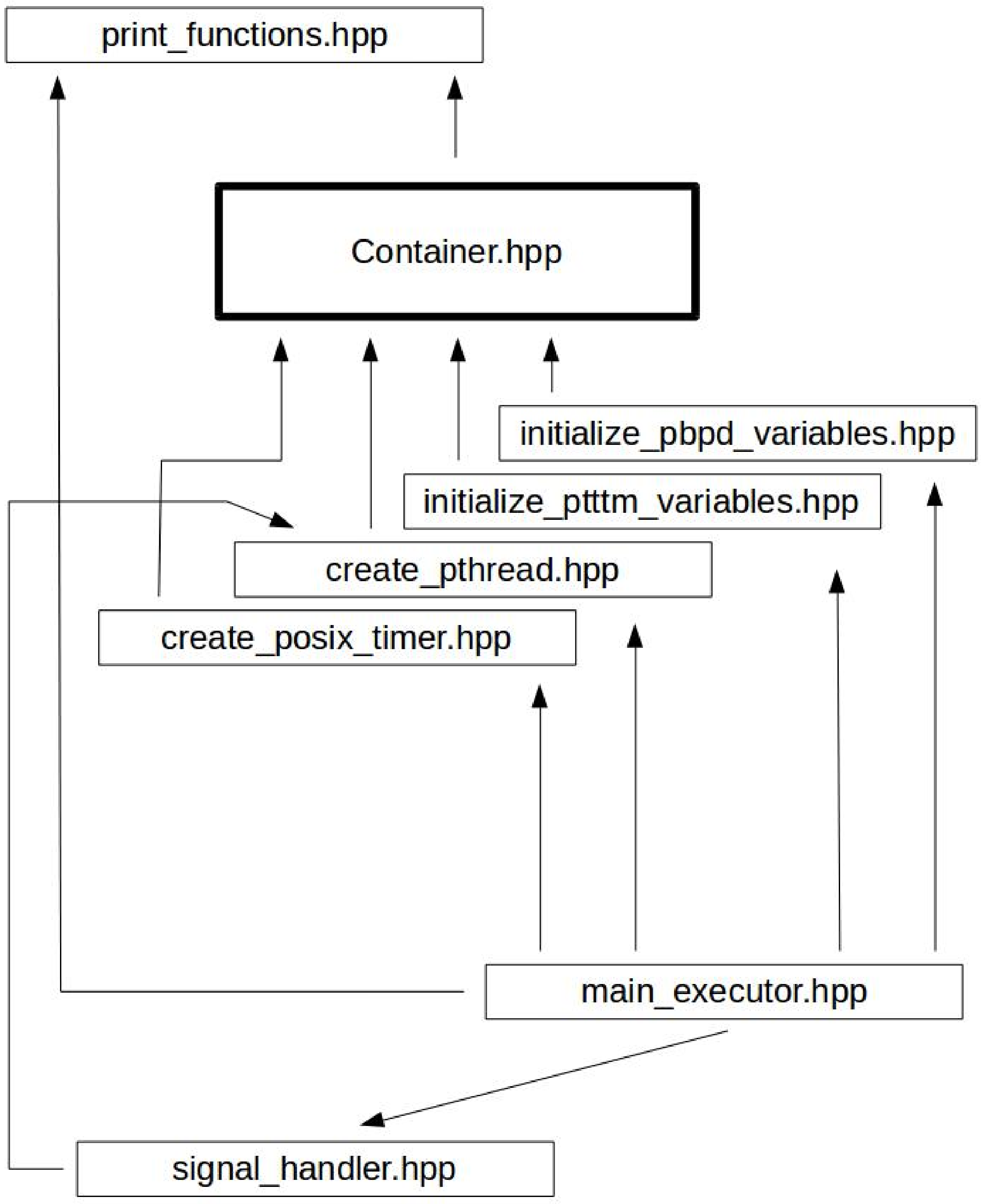
구현 과정을 요약하면 다음과 같다.

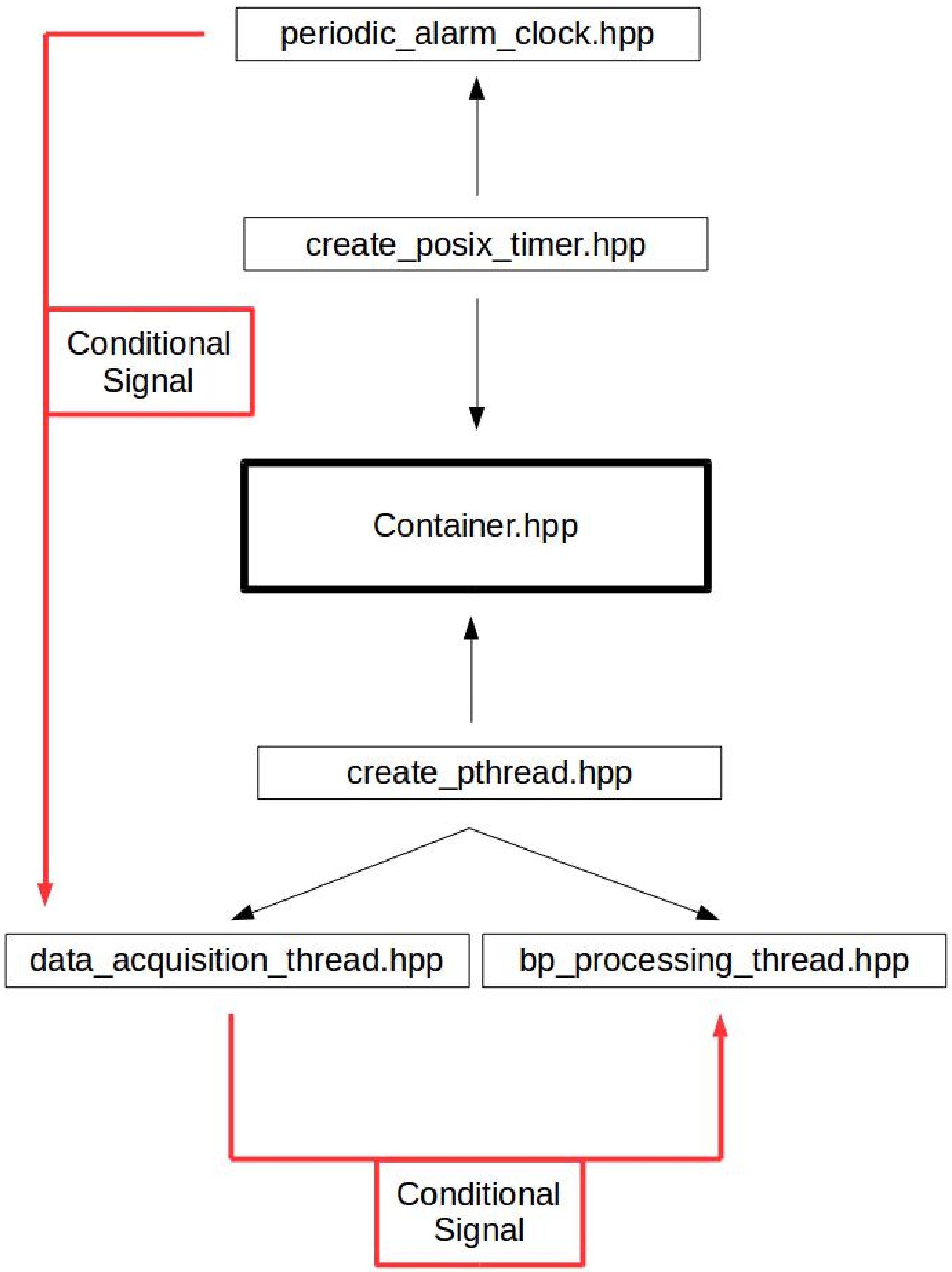
1) 모든 변수를 초기화한다.
2) Data Acquisiton Thread 를 생성한다.
3) BP Processing Thread 를 생성한다.
4) Posix Timer 를 생성해서 일정한 시간마다 User Defined Timer 를 작동하도록 만든다.

(선택사항)
5) Robust Programming with exception check
6) Signal 발생 시 (SIGRTMIN ~ SIGRTMAX) 프로그램이 gracefully exit 하도록 만든다.

Header File Dependency Graphs


initialize_ptttm_variables.hpp
initialize_pbpd_variables.hpp
static_ptttm_variables.hpp
static_pbpd_variables.hpp
periodic_alarm_clock.hpp
create_posix_timer.hpp
Container.hpp
create_pthread.hpp
bp_processing_thread.hpp
data_acquisition_thread.hpp
dequeue_bp_queue.hpp
tt_thread_register.hpp
tt_thread_wait_invocation.hpp
enqueue_bp_queue.hpp
pseudo_blood_pressure_generator.hpp


print_functions.hpp
Container.hpp
initialize_pbpd_variables.hpp
initialize_ptttm_variables.hpp
create_pthread.hpp
create_posix_timer.hpp
main_executor.hpp
signal_handler.hpp


periodic_alarm_clock.hpp
create_posix_timer.hpp
Conditional Signal
Container.hpp
create_pthread.hpp
data_acquisition_thread.hpp
bp_processing_thread.hpp
Conditional Signal

how pthread_cancel( ) works

```
pthread_cleanup_push(routine, arg)

pthread_cleanup_pop();
```

스레드를 생성하기 전에 pthread_attr_t 변수를 활용해서 옵션을 주는 기능이 있다 . 그러나 스레드를 생성한 이후에 스레드의 속성을 설정하는 함수도 다양하게 존재한다 .

여기서 해결하고자 하는 문제점은 다음과 같다 . 특정한 스레드에 대해서 pthread_cancel( ) 함수를 호출했을 때 제대로 동작하지 않았고 메인 스레드가 pthread_join( ) 에서 끝없이 블록 상태인 경우가 발생했다 .

이에 대해서는 크게 두 가지 옵션을 추가해야만 했다 .

[1] 첫째로는 pthread_setcancelstate( ) 함수를 사용해서 PTHREAD_CANCEL_ENABLE 옵션을 주는 것이다 .

[2] 둘째로는 pthread_setcanceltype( ) 함수를 사용해서 스레드를 PTHREAD_CANCEL_ASYNCHRONOUS 타입으로 만들어주는 것이다 .

PTHREAD_CANCEL_DEFERRED 타입은 pthread_testcancel( ) 부분을 스레드가 실행하고 있을때에만 pthread_cancel( ) 이 valid 하게 동작하고 나머지는 invalid 이다 . 반면에 PTHREAD_CANCEL_ASYNCHRONOUS 타입은 스레드가 어느 부분을 실행하고 있는지 상관없이 pthread_cancel( ) 이 valid 하게 정상적으로 동작한다 .

```
pthread_cleanup_push(routine, arg)

pthread_cleanup_pop();
```

여기에 더해서 cleanup code 도 지정해야 한다. 스레드가 cancel( ) 되기 위해서는 해당 스레드가 cancellation point 에 도달해 있어야만 한다. 간단하게 말하면 종료하려는 스레드가 모든 리소스를 반환한 상태여야만 한다고 보면 된다. 그러나 예를 들어 mutex_lock( ), pthread_cond_wait( ), mutex_unlock( ) 함수를 차례대로 호출했기에 스레드가 blocked 상태로 잠들어 있는 경우 mutex 변수를 점유하고 있는 것으로 봐야한다.

pthread_cancel( ) 를 호출하는 경우 스레드에 의해서 적재된 스택메모리를 모두 반환해야 하는데 이 과정에서 stack of thread-cancellation clean-up handler 가 활용된다. 자원반환을 제대로 수행하지 않으면 데드락이 발생하거나 다른 스레드의 작업 실행에 장애를 일으킨다. 이를 방지하는 매커니즘이 clean-up stack 이다.

pthread_cleanup_push( ) 를 통해서 CleanUp Handler 를 지정해서 고정할 수 있고, pthread_cleanup_pop( ) 은 반대의 작용을 한다. 클린업 과정에서는 pthread_cleanup_push( ) 을 통해 적재된 것의 역순으로 스레드가 사용한 자원을 반환한다.

Signal Handling in Multithreaded Programming

```
[ __sighandler_t ] = static void signal_handler(int);

signal( signum, __sighandler_t)
```

signal handling 방식을 간단하게 요약하면 다음과 같다 .
signal( ) 함수를 통해서 signal handler 를 지정할 수 있고 , 시그널 핸들러는 반드시 static void (*) (int) 타입으로 forward declare 해야 한다 .
이 외에 sigset_t 변수와 sigprocmask( ) 함수를 활용하므로써 시그널 별로 BLOCK, IGNORE, UNBLOCK 처리를 지정할 수 있다 .

3. Encountered Obstacles and Realizations

이번 과제를 진행하면서 부딪쳤던 몇 가지 어려움은 다음과 같다.

1) 제일 큰 어려움은 함수를 기능별로 잘게 쪼개서 구현하는 부분이었다. 프로그램에서 사용되는 기능은 무엇이 있고 중복해서 사용되는 코드는 어디이며 변수도 전역변수, 지역변수로 나누는 과정에서 많은 어려움이 있었다. 이 프로그램은 멀티스레드 프로그래밍 방식으로 구현되어 있기에 여러 개의 스레드가 서로 공유하는 변수가 많았고 코드에서 어느 부분이 Critical Section 이고 어떤 방식으로 Mutex Lock - Condition Variable – Mutex Unlock 을 지정해야 할지 고민이 많았다. 함수를 파일마다 나눠놓다보니 각각의 파일마다 " extern data-type identifier ; " 를 추가해야 하는 번거로움이 있었다.

2) 기존에 두번째 과제로 제작했던 user_defined_timer 를 이 세번째 과제의 코드 안에 정교하게 합쳐놓는 과정에서 약간의 복잡함이 있었다. 새로운 애플리케이션을 만들 때에는 기존에 아무리 잘 만들어진 API 를 선택한다고 할지라도 그대로 가져다 쓰는 것은 거의 불가능하다는 것을 깨달았다. 새롭게 구현하려는 애플리케이션에 맞춰서 API 의 내부구현을 수정해야 하는 일이 빈번하게 발생한다는 것을 명확하게 알 수 있었다.

3) 이번 과제를 하면서 한 가지 특이한 점은 pseudo blood pressure 를 관리하는 자료구조로 FIFO Queue 를 사용했다. C++에 기본으로 내장된 STL Container 인 std::queue<T>를 사용했는데, cppreference 의 도큐멘테이션에 의하면 큐타입 컨테이너가 Multithread-Unsafe 하다고 나와있음에도 불구하고 enqueu( )와 dequeue( ) 과정에서 mutual exclusion 이나 conditional variable 없이 프로그램을 동작하게 만들 수 있었다. 이것이 가능했던 이유는 첫번째로 queue.front( ), queue.back( )에 접근하는 스레드가 각각 한 개씩으로 고정되어 있었기 때문에 Race Condition 이 발생하지 않은 것이다. 두번째로는 timer 에 의해서 data_acquisition_thread 와 bp_processing_thread 가 관리되었기 때문이다. 데이터를 생산해서 큐에 삽입하는 스레드가 10ms 주기(user defined timer 가상의 시간)로 작동했다면 데이터를 큐로부터 추출해서 분석하는 스레드가 100ms 주기로 정확하게 동작하고 있었기 때문에 타이머에만 문제가 없다면 코드 전에는 segmentation fault 없이 계속해서 동작할 수 있다.

4) C++의 inline 키워드는 컴파일러에게 프로그래머의 의도를 힌트로 주는 것이다. Inline 으로 선언된 함수를 컴파일러가 실제로 inline 으로 구현할 수도 있지만 그 힌트를 따르지 않을 수도 있다. 반대로 inline 으로 선언되지 않았다고 해서 컴파일러가 항상 스택에 추가적인 메모리를 요구하면서 컴파일하지는 않으며 해당 함수를 inline 함수로 자동 변경하기도 한다. 그러나 코드를 실제로 구현하는 프로그래머 입장에서 inline function 을 다룰 경우 기억해야할 점은 컴파일 순간에 inline 으로 지정되는 함수는 컴파일러 입장에서 볼 때 'body of inline function' 을 인식할 수 있어야만 한다. 이를 통해서 Header File 안에 Declaration 뿐만 아니라 Definition 까지 존재해야만 하는 함수는크게 보아서 a) template function, b) inline function only using local variables 두 가지임을 알 수 있다. 반대로 하나의 소스파일 내에서만 사용하는 함수를 inline 으로 구현하고 싶다면 헤더파일에 함수를 선언하지 않고 소스파일에 바로 함수 정의를 구현하면 된다.

5) CmakeLists.txt 파일에서 새로운 변수를 도입할 때는 set(var value-string) 을 사용한다. 그리고 앞에서 도입한 변수를 사용하고 싶다면 ${ }와 같이 중괄호를 사용해야 한다. (This must be curly bracket, not paranthesis or square bracket) 여기서 오타를 내는 바람에 꽤 오랜 시간을 낭비하기도 했다. 앞으로는 비슷한 실수를 하지 않을 것이다.

6) Header File 에서 Declaration 할 때와 Source File 에서 Definition 을 기록할 때 function argument 를 일치시키는 것이 중요하다. 이를 철저하게 지키지 않을 경우 'undefined reference to' 에러가 발생한다.

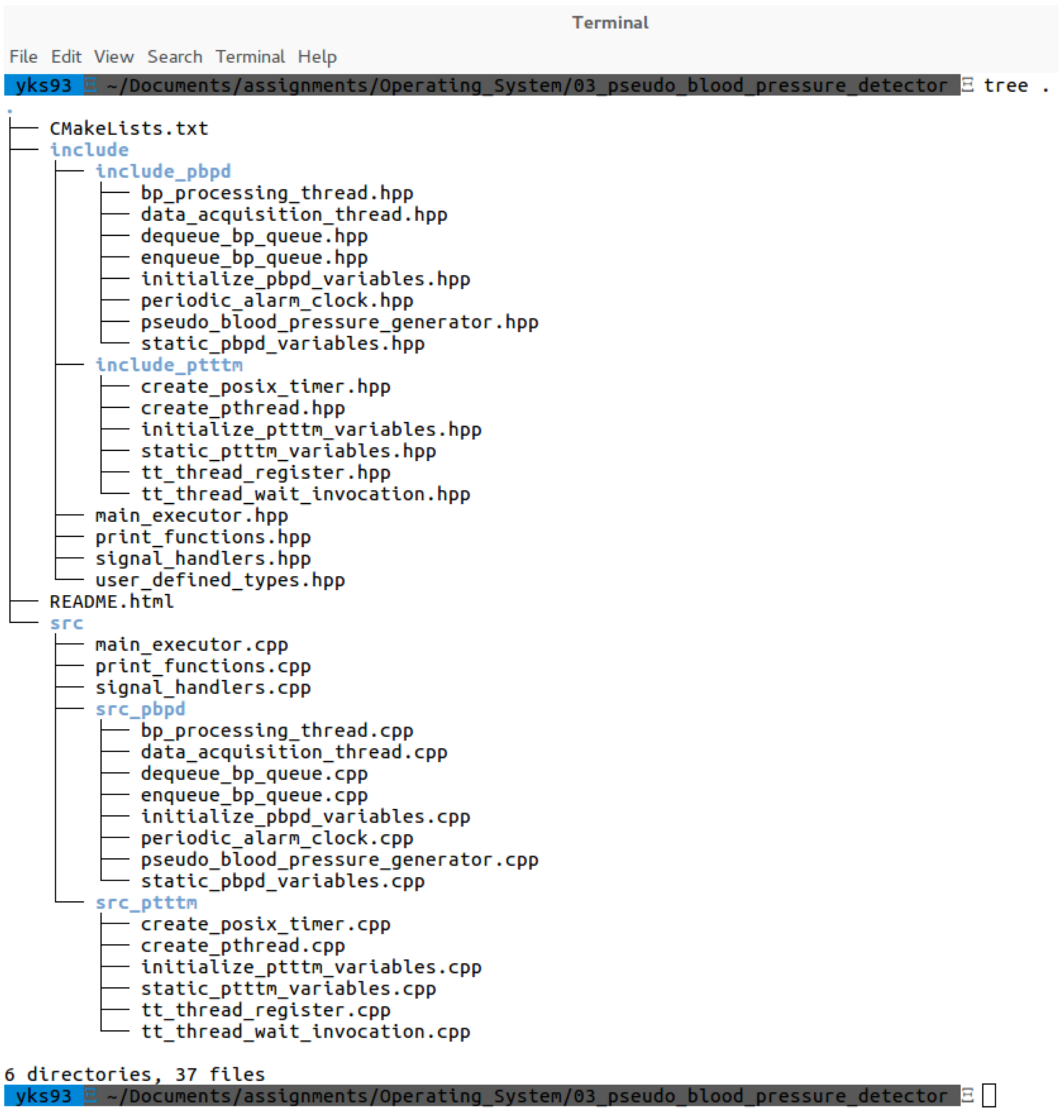
7) 포인터를 사용할 경우 ptr 의 값과 &(*ptr)의 값은 같지 않다. 그리고 포인터 변수 ptr 이 STL 컨테이너(STL Container)를 가리키고 있을 경우 ptr→some_var 이 가리키는 것이 무엇인지 확인해야 오류를 방지할 수 있다.

8) 나중에 프로그램에 수신된 임의의 시그널에 대해서 생성한 스레드 두 개를 명시적으로 소멸하기 위해서 pthread_cancel( ) 함수를 호출하려고 했다. 테스트 과정에서 pthread_cancel 함수가 제대로 실행되지 않아서 메인 스레드가 pthread_join( )에서 블록 상태로 유지되는 것을 확인했다. 이 문제점을 해결하는 것이 까다로운 부분 중 하나였다.

9) 어디선가 ‘자신이 충분한 실력이 쌓였다는 것을 어떻게 확인할 수 있을까’라는 질문에 대해서 누군가가 ‘한 번 코딩을 한 후에 한달이든 일년이든 나중에 다시 같은 코드를 봤을 때 손댈 곳이 거의 없으면 실력이 충분히 쌓인 것이다’라는 대답을 한 것을 봤다. 현재의 나는 일주일 단위로도 코드에서 손 볼 곳을 발견할 수 있고, 보다 나은 설계방식이 생각이 난다. 겨우 학부졸업생일 뿐이니 당연한 것이겠지만 아직 많이 부족하다. 진정으로 실력을 갖춘 사람이 될 때까지 더욱 정진해야겠다.

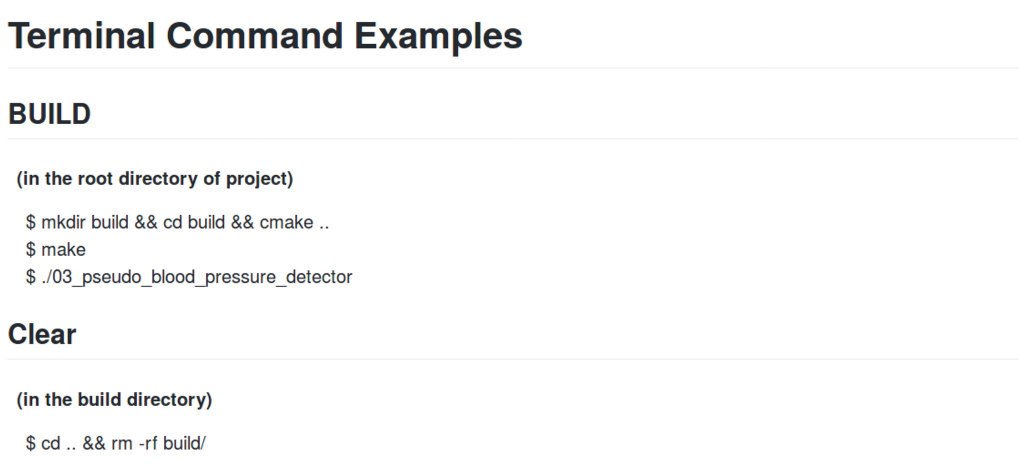
4. Source Codes (with comments)

Project Structure



```
yks93  ~/Documents/assignments/Operating_System/03_pseudo_blood_pressure_detector  tree .
.
├── CMakeLists.txt
├── include
│   ├── include_pbpd
│   │   ├── bp_processing_thread.hpp
│   │   ├── data_acquisition_thread.hpp
│   │   ├── dequeue_bp_queue.hpp
│   │   ├── enqueue_bp_queue.hpp
│   │   ├── initialize_pbpd_variables.hpp
│   │   ├── periodic_alarm_clock.hpp
│   │   ├── pseudo_blood_pressure_generator.hpp
│   │   └── static_pbpd_variables.hpp
│   ├── include_ptttm
│   │   ├── create_posix_timer.hpp
│   │   ├── create_pthread.hpp
│   │   ├── initialize_ptttm_variables.hpp
│   │   ├── static_ptttm_variables.hpp
│   │   ├── tt_thread_register.hpp
│   │   └── tt_thread_wait_invocation.hpp
│   ├── main_executor.hpp
│   ├── print_functions.hpp
│   ├── signal_handlers.hpp
│   └── user_defined_types.hpp
├── README.html
└── src
    ├── main_executor.cpp
    ├── print_functions.cpp
    ├── signal_handlers.cpp
    ├── src_pbpd
    │   ├── bp_processing_thread.cpp
    │   ├── data_acquisition_thread.cpp
    │   ├── dequeue_bp_queue.cpp
    │   ├── enqueue_bp_queue.cpp
    │   ├── initialize_pbpd_variables.cpp
    │   ├── periodic_alarm_clock.cpp
    │   ├── pseudo_blood_pressure_generator.cpp
    │   └── static_pbpd_variables.cpp
    └── src_ptttm
        ├── create_posix_timer.cpp
        ├── create_pthread.cpp
        ├── initialize_ptttm_variables.cpp
        ├── static_ptttm_variables.cpp
        ├── tt_thread_register.cpp
        └── tt_thread_wait_invocation.cpp

6 directories, 37 files
yks93  ~/Documents/assignments/Operating_System/03_pseudo_blood_pressure_detector 
```

README.md



Terminal Command Examples

BUILD

(in the root directory of project)

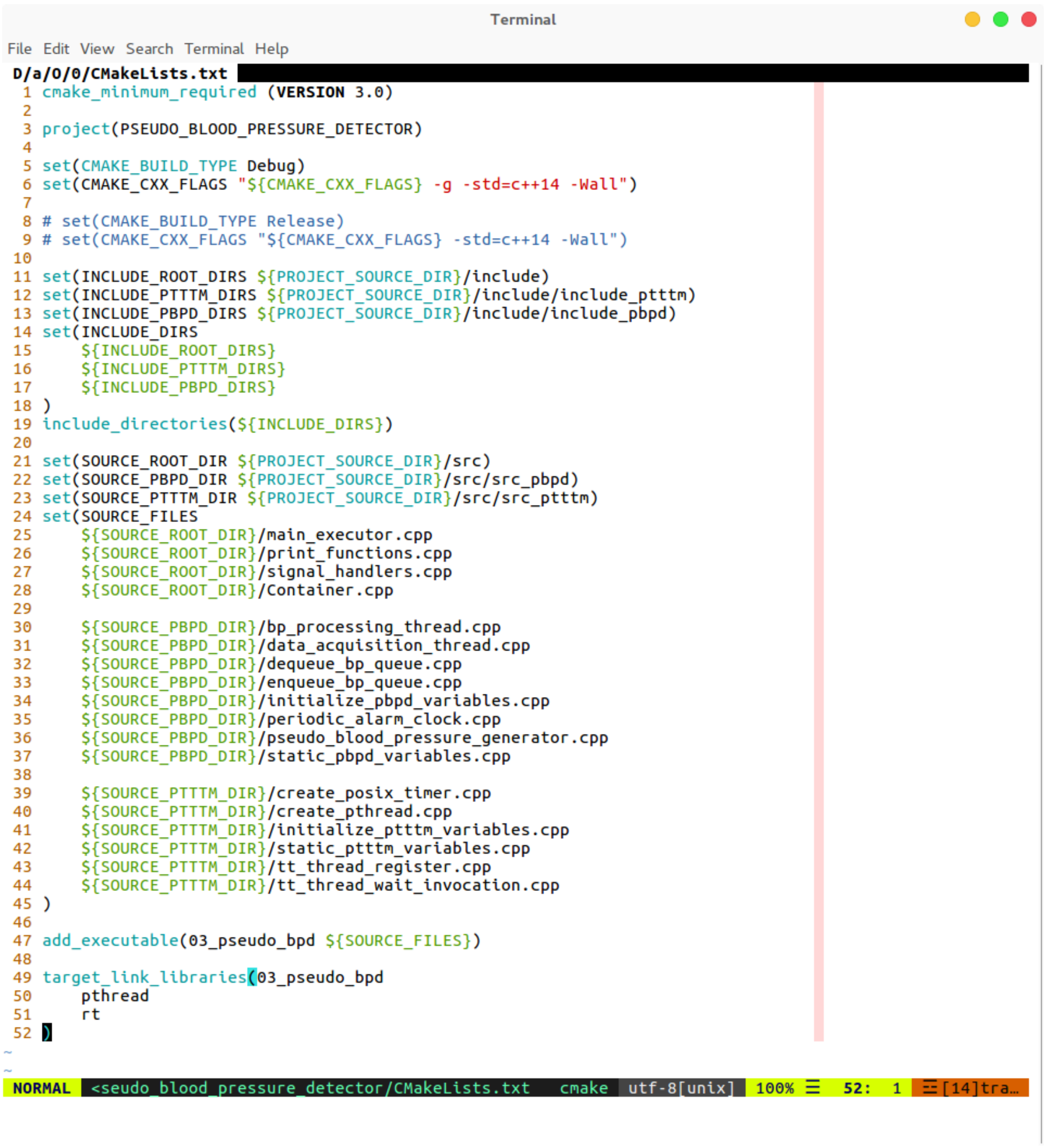
```
$ mkdir build && cd build && cmake ..
$ make
$ ./03_pseudo_blood_pressure_detector
```

Clear

(in the build directory)

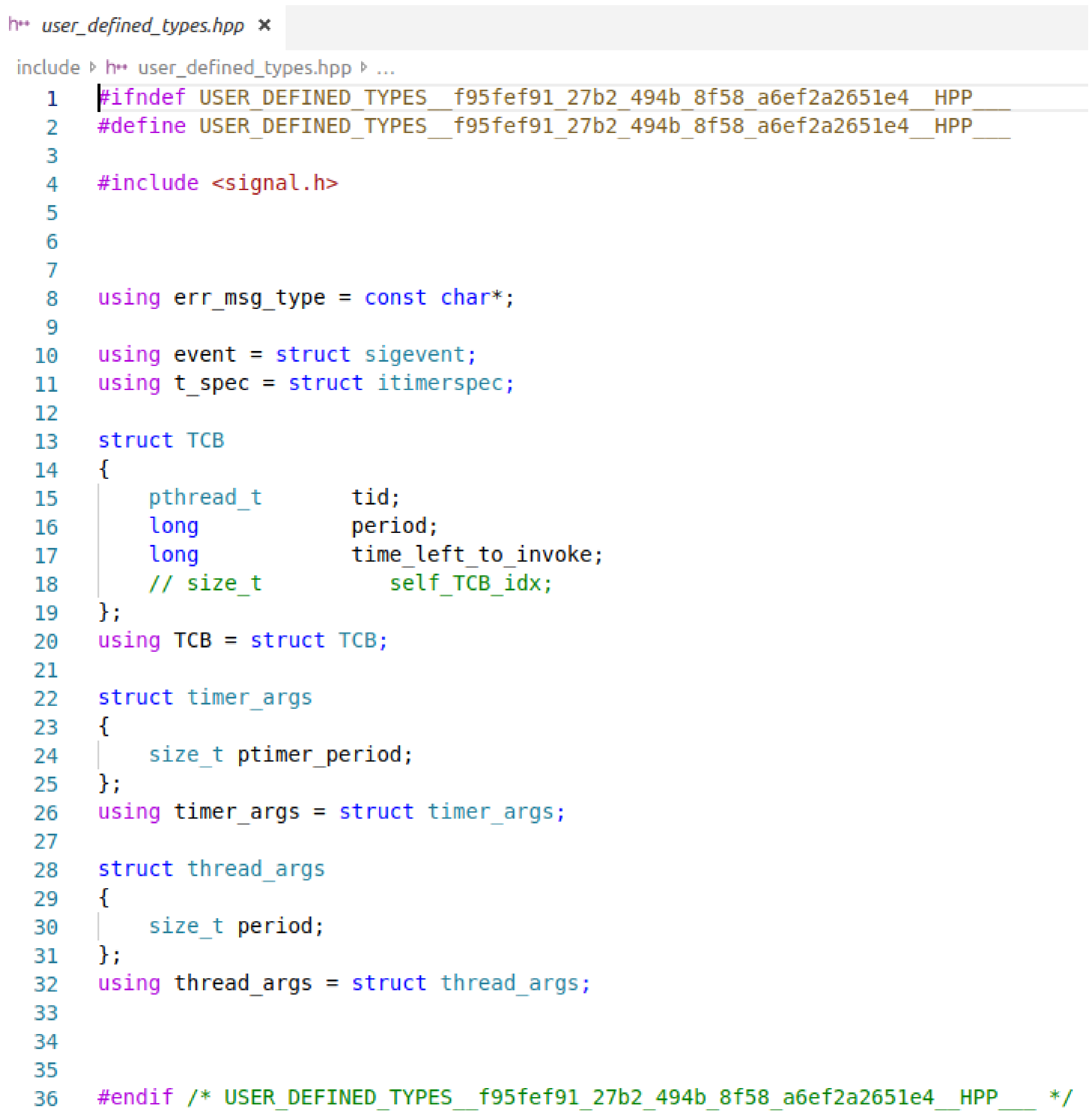
```
$ cd .. && rm -rf build/
```

CMakeLists.txt



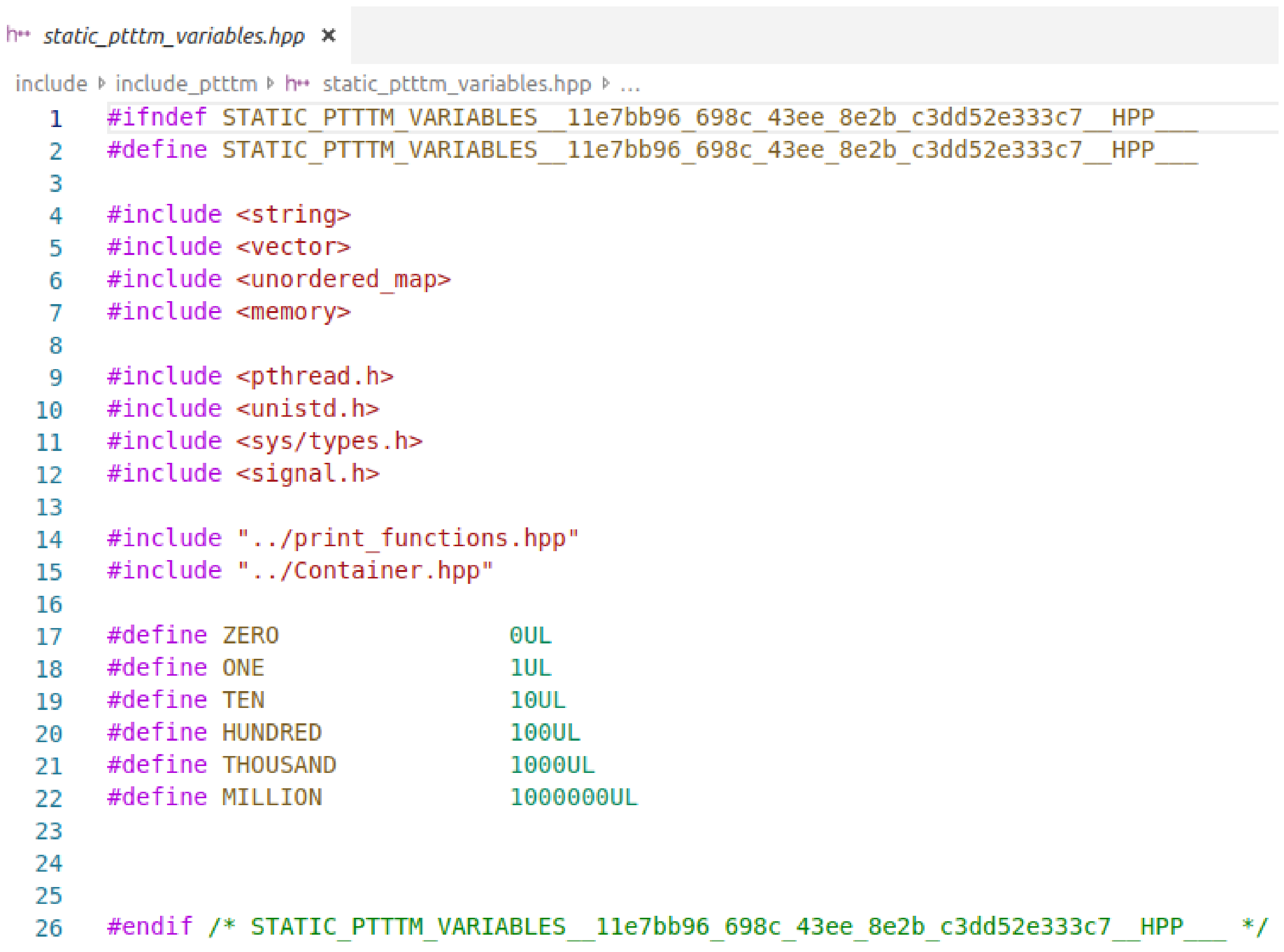
```
D/a/0/0/CMakeLists.txt
cmake_minimum_required (VERSION 3.0)

project(PSEUDO_BLOOD_PRESSURE_DETECTOR)

set(CMAKE_BUILD_TYPE Debug)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -g -std=c++14 -Wall")

# set(CMAKE_BUILD_TYPE Release)
# set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++14 -Wall")

set(INCLUDE_ROOT_DIRS ${PROJECT_SOURCE_DIR}/include)
set(INCLUDE_PTTTM_DIRS ${PROJECT_SOURCE_DIR}/include/include_ptttm)
set(INCLUDE_PBPD_DIRS ${PROJECT_SOURCE_DIR}/include/include_pbpd)
set(INCLUDE_DIRS
    ${INCLUDE_ROOT_DIRS}
    ${INCLUDE_PTTTM_DIRS}
    ${INCLUDE_PBPD_DIRS}
)
include_directories(${INCLUDE_DIRS})

set(SOURCE_ROOT_DIR ${PROJECT_SOURCE_DIR}/src)
set(SOURCE_PBPD_DIR ${PROJECT_SOURCE_DIR}/src/src_pbpd)
set(SOURCE_PTTTM_DIR ${PROJECT_SOURCE_DIR}/src/src_ptttm)
set(SOURCE_FILES
    ${SOURCE_ROOT_DIR}/main_executor.cpp
    ${SOURCE_ROOT_DIR}/print_functions.cpp
    ${SOURCE_ROOT_DIR}/signal_handlers.cpp
    ${SOURCE_ROOT_DIR}/Container.cpp

    ${SOURCE_PBPD_DIR}/bp_processing_thread.cpp
    ${SOURCE_PBPD_DIR}/data_acquisition_thread.cpp
    ${SOURCE_PBPD_DIR}/dequeue_bp_queue.cpp
    ${SOURCE_PBPD_DIR}/enqueue_bp_queue.cpp
    ${SOURCE_PBPD_DIR}/initialize_pbpd_variables.cpp
    ${SOURCE_PBPD_DIR}/periodic_alarm_clock.cpp
    ${SOURCE_PBPD_DIR}/pseudo_blood_pressure_generator.cpp
    ${SOURCE_PBPD_DIR}/static_pbpd_variables.cpp

    ${SOURCE_PTTTM_DIR}/create_posix_timer.cpp
    ${SOURCE_PTTTM_DIR}/create_pthread.cpp
    ${SOURCE_PTTTM_DIR}/initialize_ptttm_variables.cpp
    ${SOURCE_PTTTM_DIR}/static_ptttm_variables.cpp
    ${SOURCE_PTTTM_DIR}/tt_thread_register.cpp
    ${SOURCE_PTTTM_DIR}/tt_thread_wait_invocation.cpp
)

add_executable(03_pseudo_bpd ${SOURCE_FILES})

target_link_libraries(03_pseudo_bpd
    pthread
    rt
)
~
~
NORMAL  <seudo_blood_pressure_detector/CMakeLists.txt   cmake  utf-8[unix]  100% ☰  52:  1  ☲[14]tra…
```

user_defined_types.hpp



```
#ifndef USER_DEFINED_TYPES__f95fef91_27b2_494b_8f58_a6ef2a2651e4__HPP___
#define USER_DEFINED_TYPES__f95fef91_27b2_494b_8f58_a6ef2a2651e4__HPP___

#include <signal.h>



using err_msg_type = const char*;

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 /* USER_DEFINED_TYPES__f95fef91_27b2_494b_8f58_a6ef2a2651e4__HPP___ */
```

static_ptttm_variables.hpp

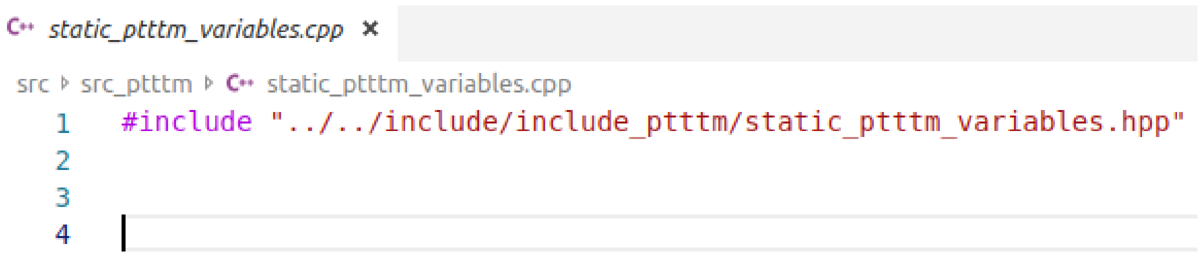


static_ptttm_variables.hpp ×

include ▹ include_ptttm ▹ static_ptttm_variables.hpp ▹ ...

```cpp
#ifndef STATIC_PTTTM_VARIABLES__11e7bb96_698c_43ee_8e2b_c3dd52e333c7__HPP___
#define STATIC_PTTTM_VARIABLES__11e7bb96_698c_43ee_8e2b_c3dd52e333c7__HPP___

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

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

#include "../print_functions.hpp"
#include "../Container.hpp"

#define ZERO                0UL
#define ONE                 1UL
#define TEN                 10UL
#define HUNDRED             100UL
#define THOUSAND            1000UL
#define MILLION             1000000UL



#endif /* STATIC_PTTTM_VARIABLES__11e7bb96_698c_43ee_8e2b_c3dd52e333c7__HPP___ */
```

static_ptttm_variables.cpp

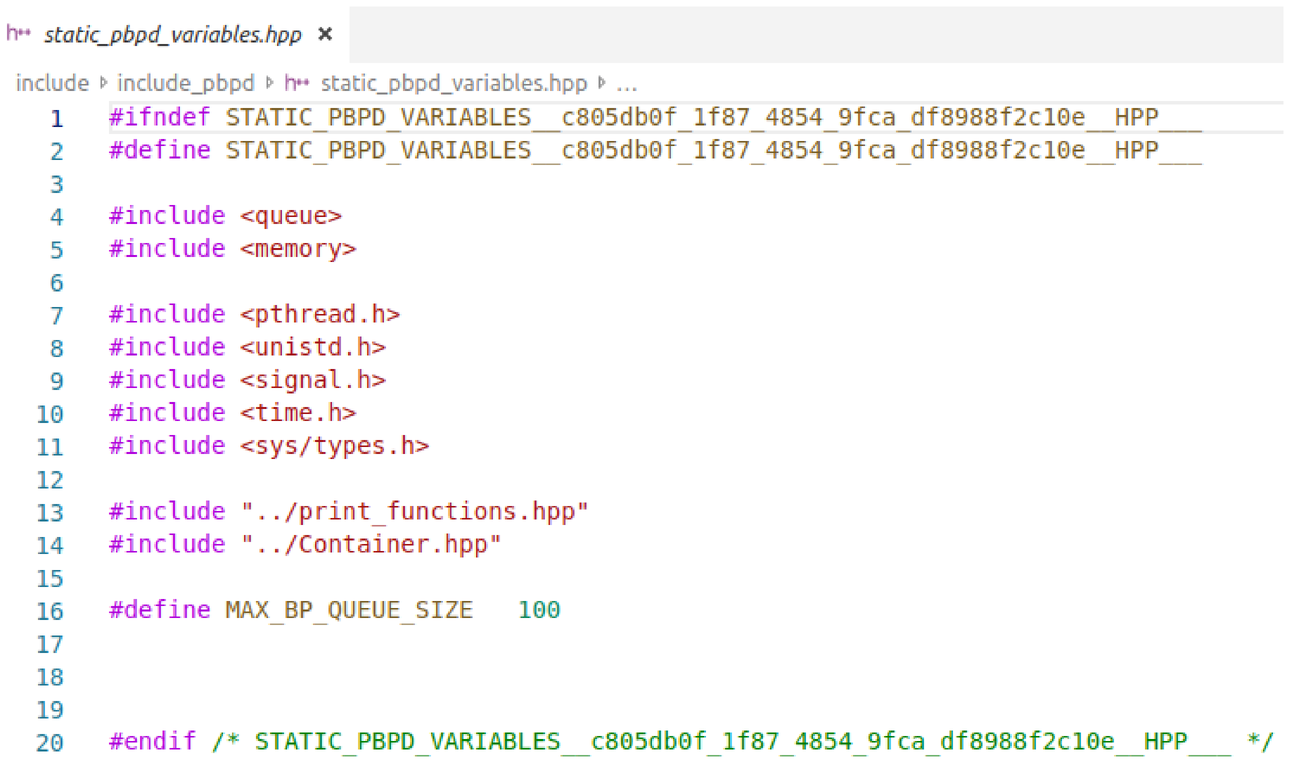


static_ptttm_variables.cpp ×

src ▹ src_ptttm ▹ static_ptttm_variables.cpp

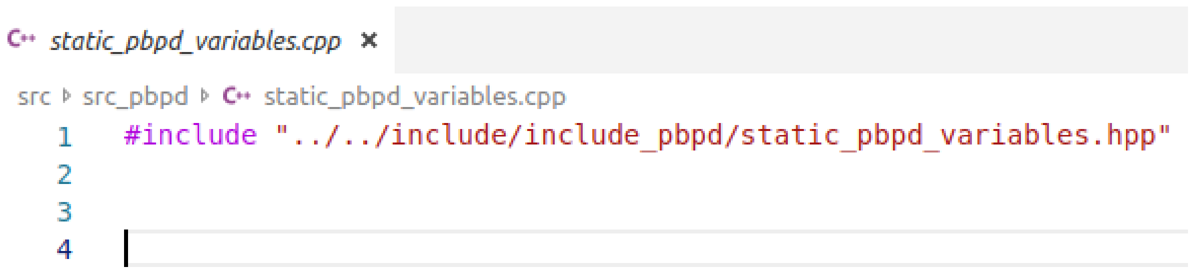
```cpp
#include "../../include/include_ptttm/static_ptttm_variables.hpp"


```

static_pbpd_variables.hpp


static_pbpd_variables.hpp
include ▸ include_pbpd ▸ static_pbpd_variables.hpp ▸ ...

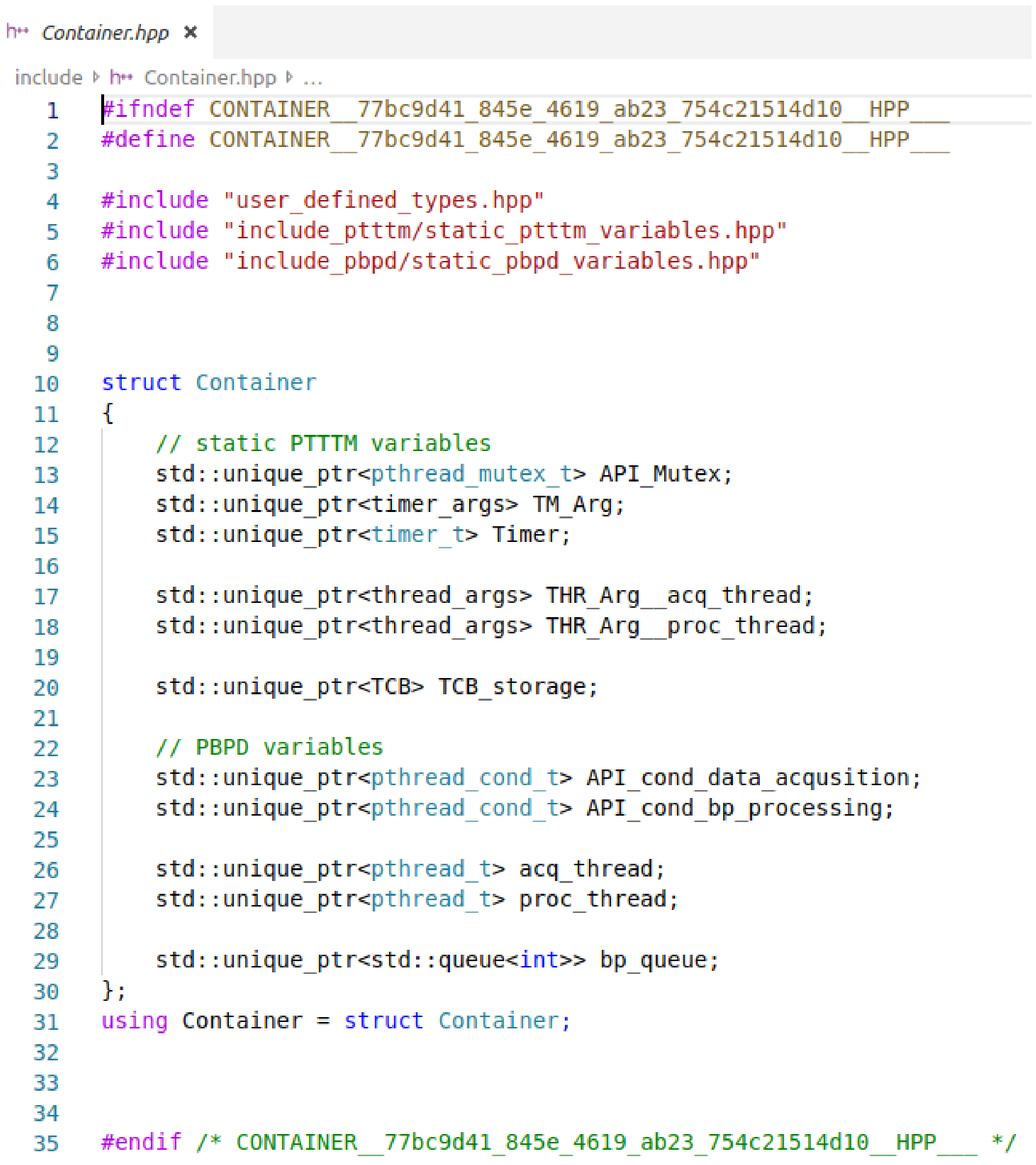
```
#ifndef STATIC_PBPD_VARIABLES__c805db0f_1f87_4854_9fca_df8988f2c10e__HPP___
#define STATIC_PBPD_VARIABLES__c805db0f_1f87_4854_9fca_df8988f2c10e__HPP___

#include <queue>
#include <memory>

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

#include "../print_functions.hpp"
#include "../Container.hpp"

#define MAX_BP_QUEUE_SIZE   100



#endif /* STATIC_PBPD_VARIABLES__c805db0f_1f87_4854_9fca_df8988f2c10e__HPP___ */
```

static_pbpd_variables.cpp


static_pbpd_variables.cpp
src ▸ src_pbpd ▸ static_pbpd_variables.cpp

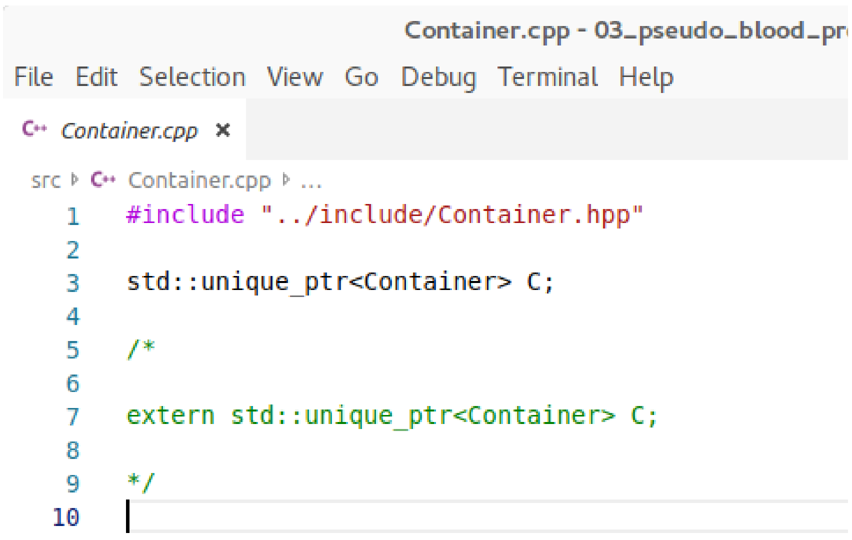
```
#include "../../include/include_pbpd/static_pbpd_variables.hpp"



```

Container.hpp



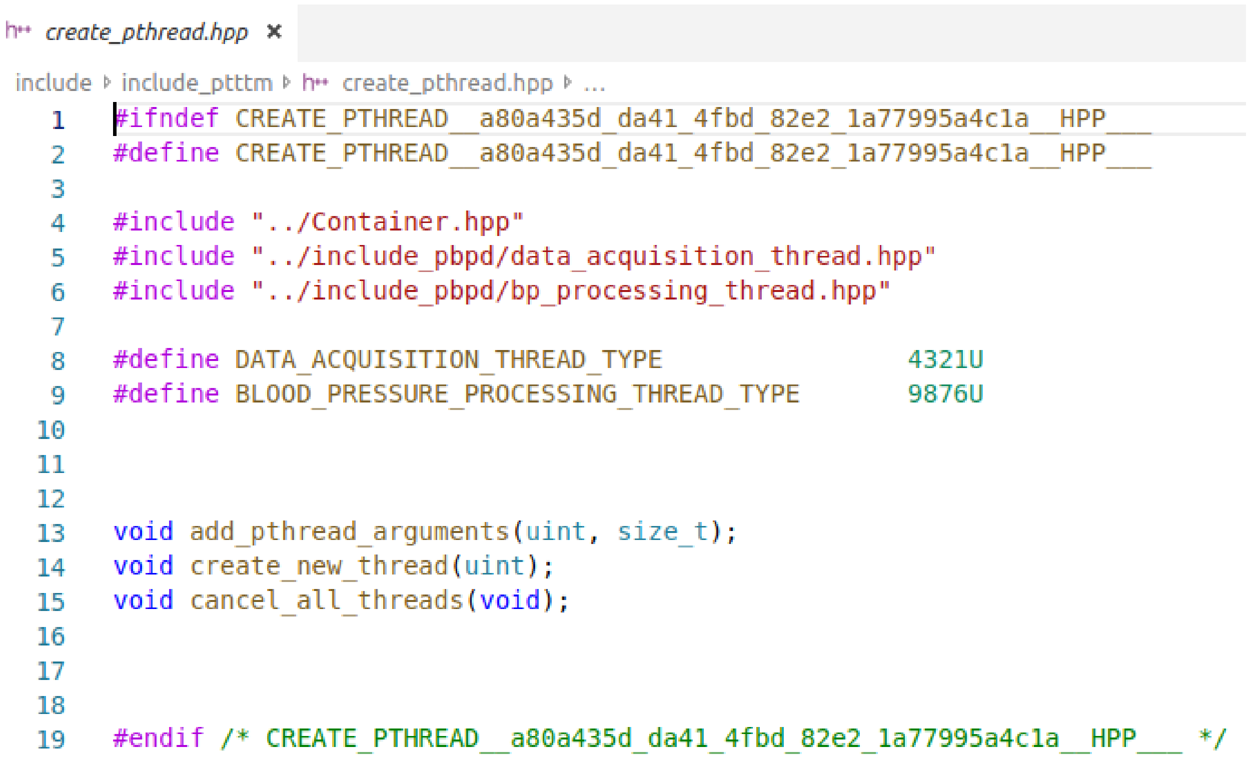
```cpp
#ifndef CONTAINER__77bc9d41_845e_4619_ab23_754c21514d10__HPP___
#define CONTAINER__77bc9d41_845e_4619_ab23_754c21514d10__HPP___

#include "user_defined_types.hpp"
#include "include_ptttm/static_ptttm_variables.hpp"
#include "include_pbpd/static_pbpd_variables.hpp"



struct Container
{
    // static PTTTM variables
    std::unique_ptr<pthread_mutex_t> API_Mutex;
    std::unique_ptr<timer_args> TM_Arg;
    std::unique_ptr<timer_t> Timer;

    std::unique_ptr<thread_args> THR_Arg__acq_thread;
    std::unique_ptr<thread_args> THR_Arg__proc_thread;

    std::unique_ptr<TCB> TCB_storage;

    // PBPD variables
    std::unique_ptr<pthread_cond_t> API_cond_data_acqusition;
    std::unique_ptr<pthread_cond_t> API_cond_bp_processing;

    std::unique_ptr<pthread_t> acq_thread;
    std::unique_ptr<pthread_t> proc_thread;

    std::unique_ptr<std::queue<int>> bp_queue;
};
using Container = struct Container;



#endif /* CONTAINER__77bc9d41_845e_4619_ab23_754c21514d10__HPP___ */
```

Container.cpp


Container.cpp - 03_pseudo_blood_pr
File Edit Selection View Go Debug Terminal Help
Container.cpp
src ▸ Container.cpp ▸ …

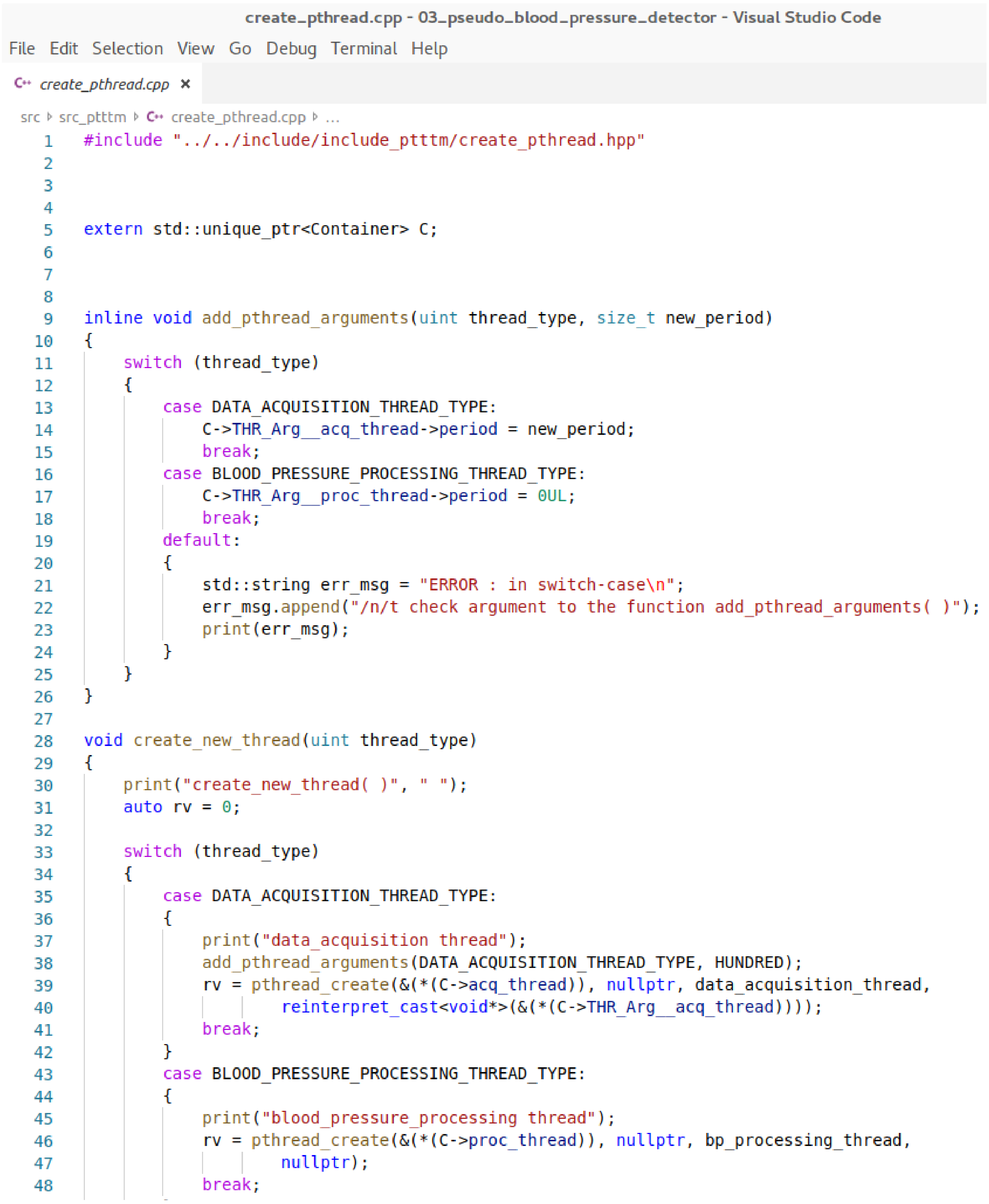
```cpp
#include "../include/Container.hpp"

std::unique_ptr<Container> C;

/*

extern std::unique_ptr<Container> C;

*/

```

create_pthread.hpp

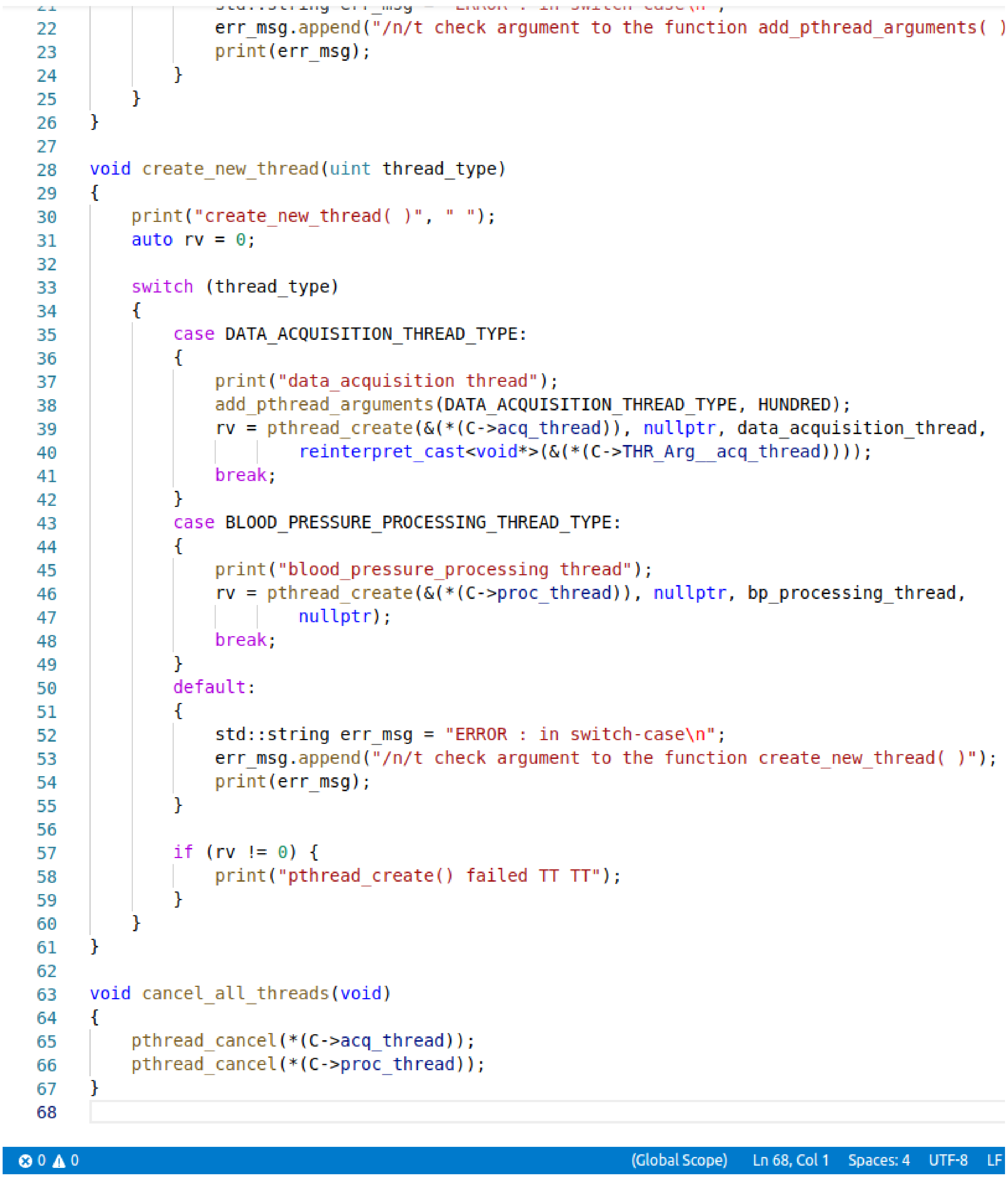


```
#ifndef CREATE_PTHREAD__a80a435d_da41_4fbd_82e2_1a77995a4c1a__HPP___
#define CREATE_PTHREAD__a80a435d_da41_4fbd_82e2_1a77995a4c1a__HPP___

#include "../Container.hpp"
#include "../include_pbpd/data_acquisition_thread.hpp"
#include "../include_pbpd/bp_processing_thread.hpp"

#define DATA_ACQUISITION_THREAD_TYPE              4321U
#define BLOOD_PRESSURE_PROCESSING_THREAD_TYPE     9876U



void add_pthread_arguments(uint, size_t);
void create_new_thread(uint);
void cancel_all_threads(void);



#endif /* CREATE_PTHREAD__a80a435d_da41_4fbd_82e2_1a77995a4c1a__HPP___ */
```

create_pthread.cpp

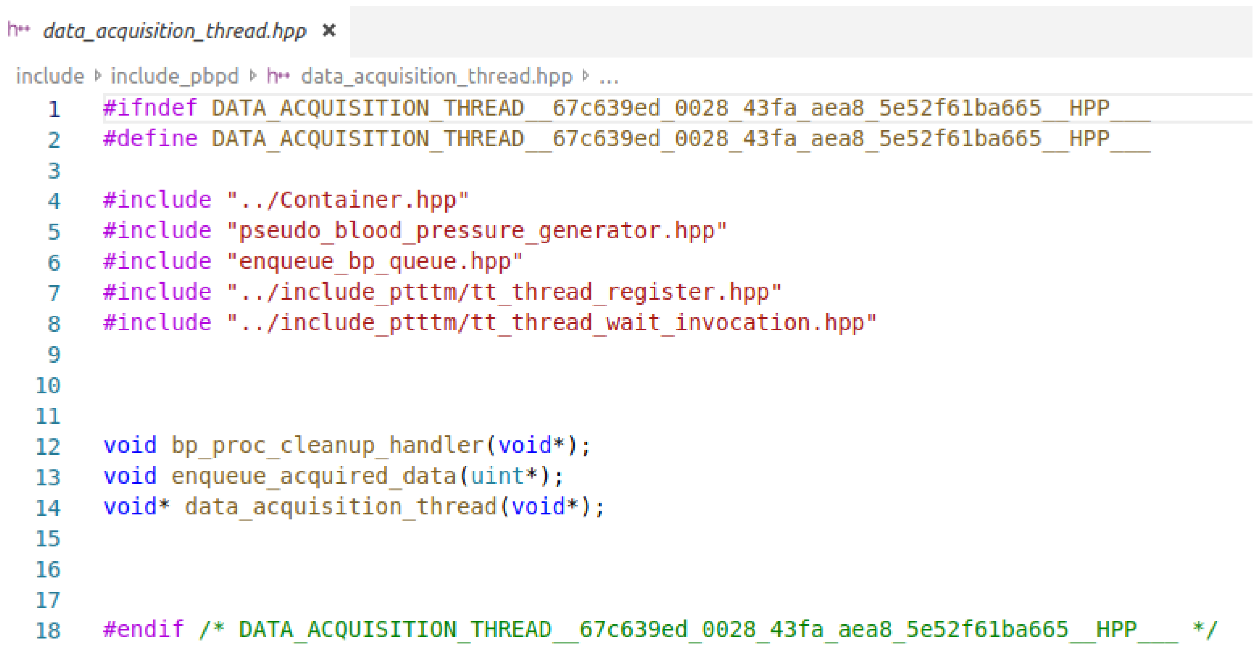

create_pthread.cpp - 03_pseudo_blood_pressure_detector - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
create_pthread.cpp
src › src_ptttm › create_pthread.cpp › ...

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



extern std::unique_ptr<Container> C;



inline void add_pthread_arguments(uint thread_type, size_t new_period)
{
    switch (thread_type)
    {
        case DATA_ACQUISITION_THREAD_TYPE:
            C->THR_Arg__acq_thread->period = new_period;
            break;
        case BLOOD_PRESSURE_PROCESSING_THREAD_TYPE:
            C->THR_Arg__proc_thread->period = 0UL;
            break;
        default:
        {
            std::string err_msg = "ERROR : in switch-case\n";
            err_msg.append("/n/t check argument to the function add_pthread_arguments( )");
            print(err_msg);
        }
    }
}

void create_new_thread(uint thread_type)
{
    print("create_new_thread( )", " ");
    auto rv = 0;

    switch (thread_type)
    {
        case DATA_ACQUISITION_THREAD_TYPE:
        {
            print("data_acquisition thread");
            add_pthread_arguments(DATA_ACQUISITION_THREAD_TYPE, HUNDRED);
            rv = pthread_create(&(*(C->acq_thread)), nullptr, data_acquisition_thread,
                    reinterpret_cast<void*>(&(*(C->THR_Arg__acq_thread))));
            break;
        }
        case BLOOD_PRESSURE_PROCESSING_THREAD_TYPE:
        {
            print("blood_pressure_processing thread");
            rv = pthread_create(&(*(C->proc_thread)), nullptr, bp_processing_thread,
                    nullptr);
            break;
```



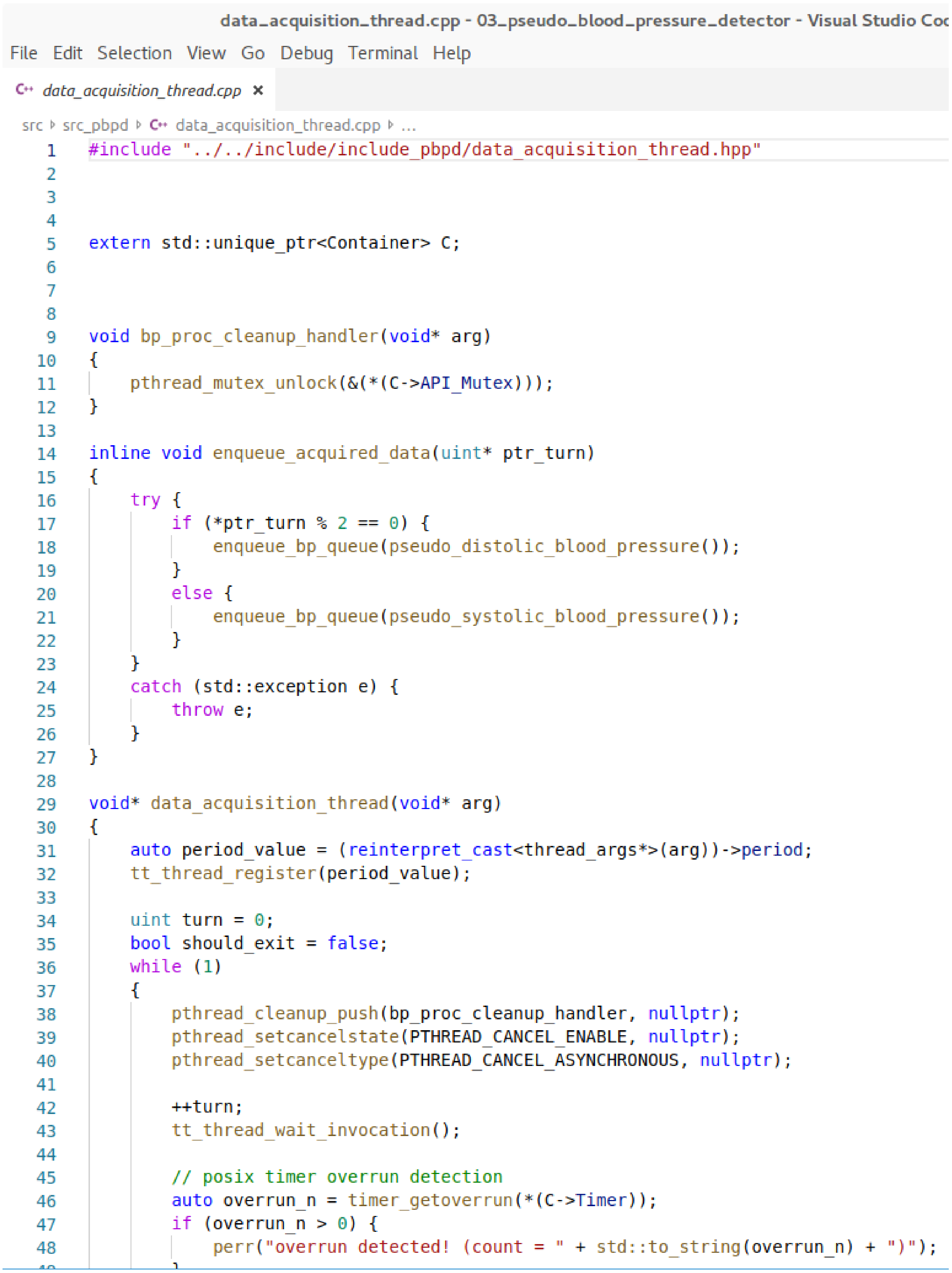
```
            std::string err_msg = "ERROR : in switch-case\n";
            err_msg.append("/n/t check argument to the function add_pthread_arguments( )
            print(err_msg);
        }
    }
}

void create_new_thread(uint thread_type)
{
    print("create_new_thread( )", " ");
    auto rv = 0;

    switch (thread_type)
    {
        case DATA_ACQUISITION_THREAD_TYPE:
        {
            print("data_acquisition thread");
            add_pthread_arguments(DATA_ACQUISITION_THREAD_TYPE, HUNDRED);
            rv = pthread_create(&(*(C->acq_thread)), nullptr, data_acquisition_thread,
                    reinterpret_cast<void*>(&(*(C->THR_Arg__acq_thread))));
            break;
        }
        case BLOOD_PRESSURE_PROCESSING_THREAD_TYPE:
        {
            print("blood_pressure_processing thread");
            rv = pthread_create(&(*(C->proc_thread)), nullptr, bp_processing_thread,
                    nullptr);
            break;
        }
        default:
        {
            std::string err_msg = "ERROR : in switch-case\n";
            err_msg.append("/n/t check argument to the function create_new_thread( )");
            print(err_msg);
        }

        if (rv != 0) {
            print("pthread_create() failed TT TT");
        }
    }
}

void cancel_all_threads(void)
{
    pthread_cancel(*(C->acq_thread));
    pthread_cancel(*(C->proc_thread));
}

```

data_acquisition_thread.hpp

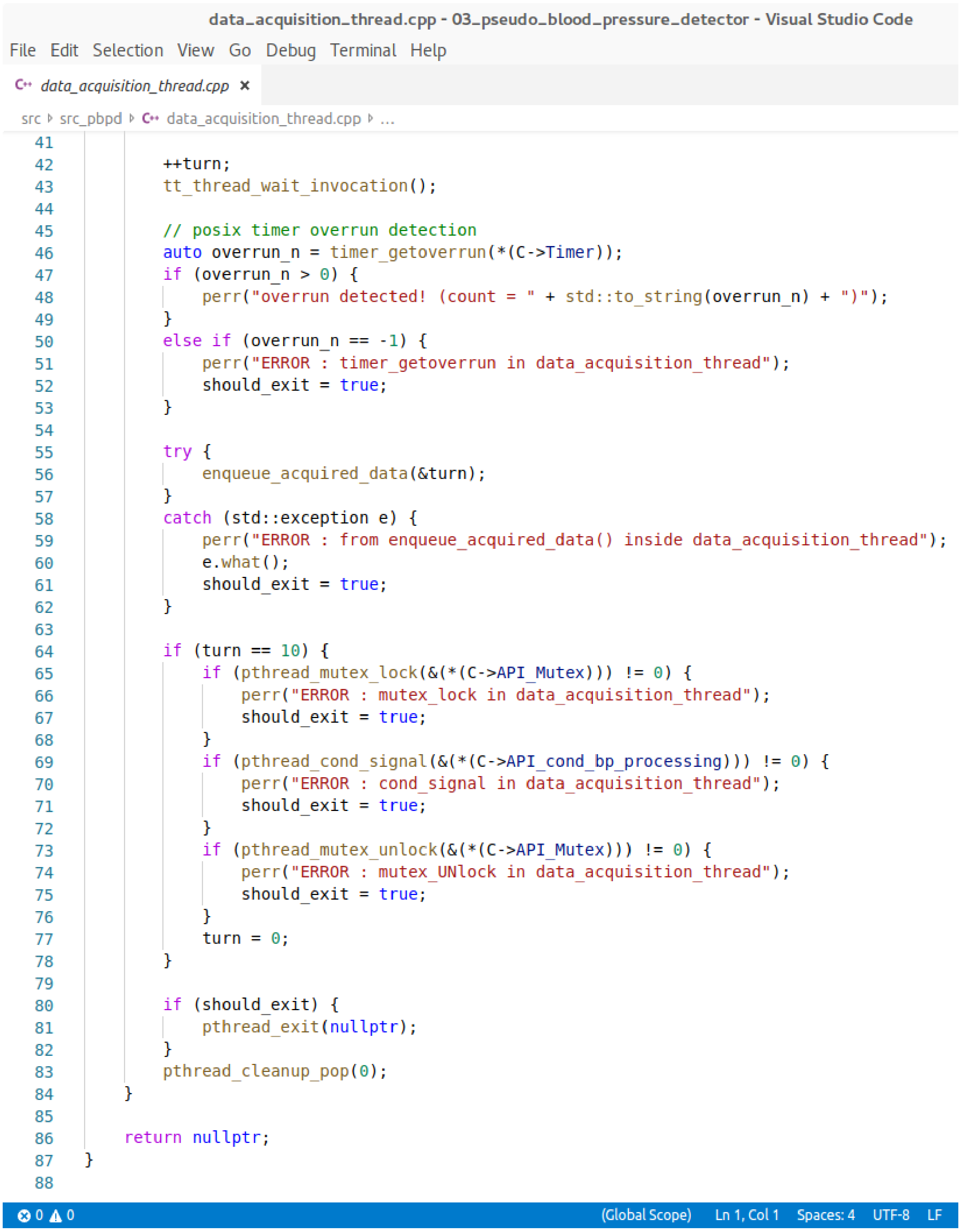

data_acquisition_thread.hpp
include › include_pbpd › data_acquisition_thread.hpp › ...

```
#ifndef DATA_ACQUISITION_THREAD__67c639ed_0028_43fa_aea8_5e52f61ba665__HPP___
#define DATA_ACQUISITION_THREAD__67c639ed_0028_43fa_aea8_5e52f61ba665__HPP___

#include "../Container.hpp"
#include "pseudo_blood_pressure_generator.hpp"
#include "enqueue_bp_queue.hpp"
#include "../include_ptttm/tt_thread_register.hpp"
#include "../include_ptttm/tt_thread_wait_invocation.hpp"



void bp_proc_cleanup_handler(void*);
void enqueue_acquired_data(uint*);
void* data_acquisition_thread(void*);



#endif /* DATA_ACQUISITION_THREAD__67c639ed_0028_43fa_aea8_5e52f61ba665__HPP___ */
```

data_acquisition_thread.cpp

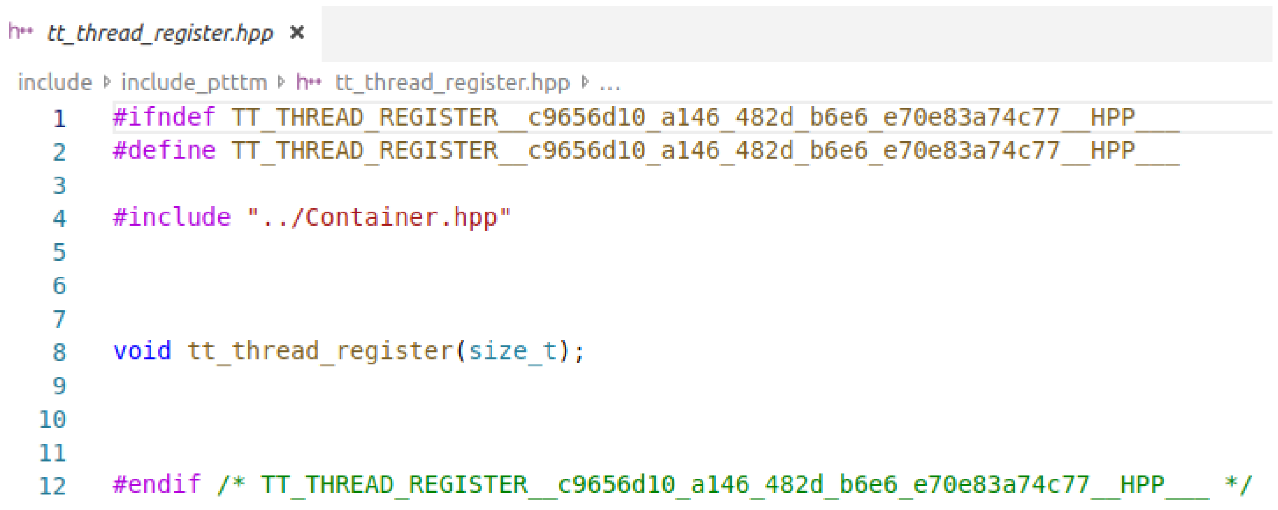

data_acquisition_thread.cpp - 03_pseudo_blood_pressure_detector - Visual Studio Coc
File Edit Selection View Go Debug Terminal Help
data_acquisition_thread.cpp
src › src_pbpd › data_acquisition_thread.cpp › ...

```cpp
#include "../../include/include_pbpd/data_acquisition_thread.hpp"



extern std::unique_ptr<Container> C;



void bp_proc_cleanup_handler(void* arg)
{
    pthread_mutex_unlock(&(*(C->API_Mutex)));
}

inline void enqueue_acquired_data(uint* ptr_turn)
{
    try {
        if (*ptr_turn % 2 == 0) {
            enqueue_bp_queue(pseudo_distolic_blood_pressure());
        }
        else {
            enqueue_bp_queue(pseudo_systolic_blood_pressure());
        }
    }
    catch (std::exception e) {
        throw e;
    }
}

void* data_acquisition_thread(void* arg)
{
    auto period_value = (reinterpret_cast<thread_args*>(arg))->period;
    tt_thread_register(period_value);

    uint turn = 0;
    bool should_exit = false;
    while (1)
    {
        pthread_cleanup_push(bp_proc_cleanup_handler, nullptr);
        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, nullptr);
        pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, nullptr);

        ++turn;
        tt_thread_wait_invocation();

        // posix timer overrun detection
        auto overrun_n = timer_getoverrun(*(C->Timer));
        if (overrun_n > 0) {
            perr("overrun detected! (count = " + std::to_string(overrun_n) + ")");
        }
```


data_acquisition_thread.cpp - 03_pseudo_blood_pressure_detector - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
data_acquisition_thread.cpp
src › src_pbpd › data_acquisition_thread.cpp › ...
0 0
(Global Scope) Ln 1, Col 1 Spaces: 4 UTF-8 LF

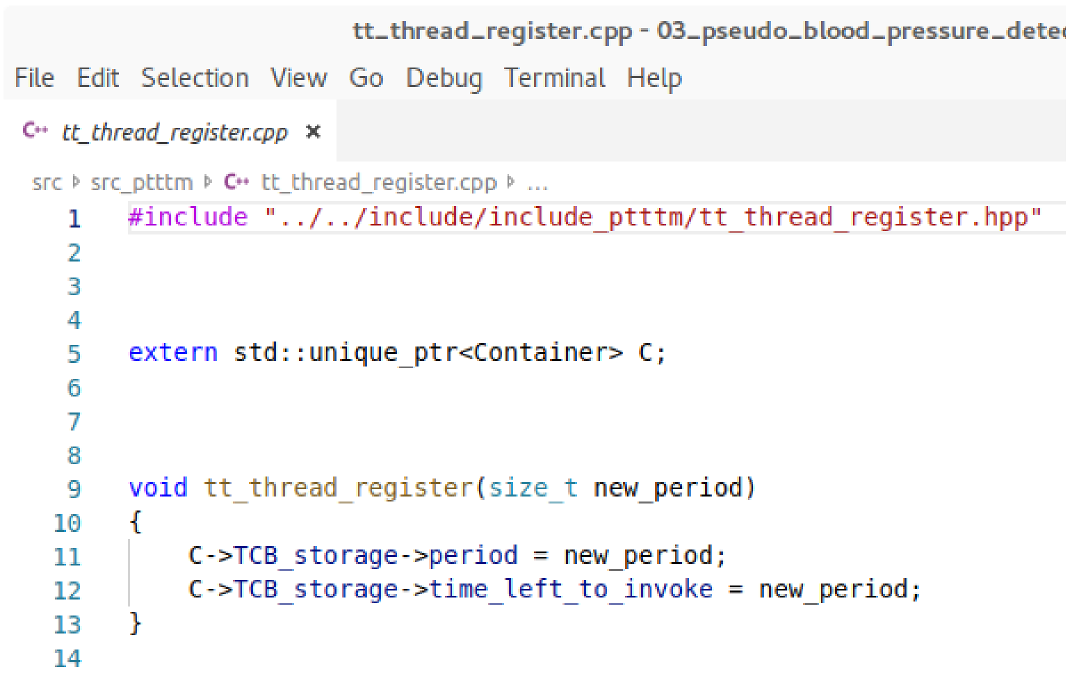
```cpp

        ++turn;
        tt_thread_wait_invocation();

        // posix timer overrun detection
        auto overrun_n = timer_getoverrun(*(C->Timer));
        if (overrun_n > 0) {
            perr("overrun detected! (count = " + std::to_string(overrun_n) + ")");
        }
        else if (overrun_n == -1) {
            perr("ERROR : timer_getoverrun in data_acquisition_thread");
            should_exit = true;
        }

        try {
            enqueue_acquired_data(&turn);
        }
        catch (std::exception e) {
            perr("ERROR : from enqueue_acquired_data() inside data_acquisition_thread");
            e.what();
            should_exit = true;
        }

        if (turn == 10) {
            if (pthread_mutex_lock(&(*(C->API_Mutex))) != 0) {
                perr("ERROR : mutex_lock in data_acquisition_thread");
                should_exit = true;
            }
            if (pthread_cond_signal(&(*(C->API_cond_bp_processing))) != 0) {
                perr("ERROR : cond_signal in data_acquisition_thread");
                should_exit = true;
            }
            if (pthread_mutex_unlock(&(*(C->API_Mutex))) != 0) {
                perr("ERROR : mutex_UNlock in data_acquisition_thread");
                should_exit = true;
            }
            turn = 0;
        }

        if (should_exit) {
            pthread_exit(nullptr);
        }
        pthread_cleanup_pop(0);
    }

    return nullptr;
}

```

tt_thread_register.hpp



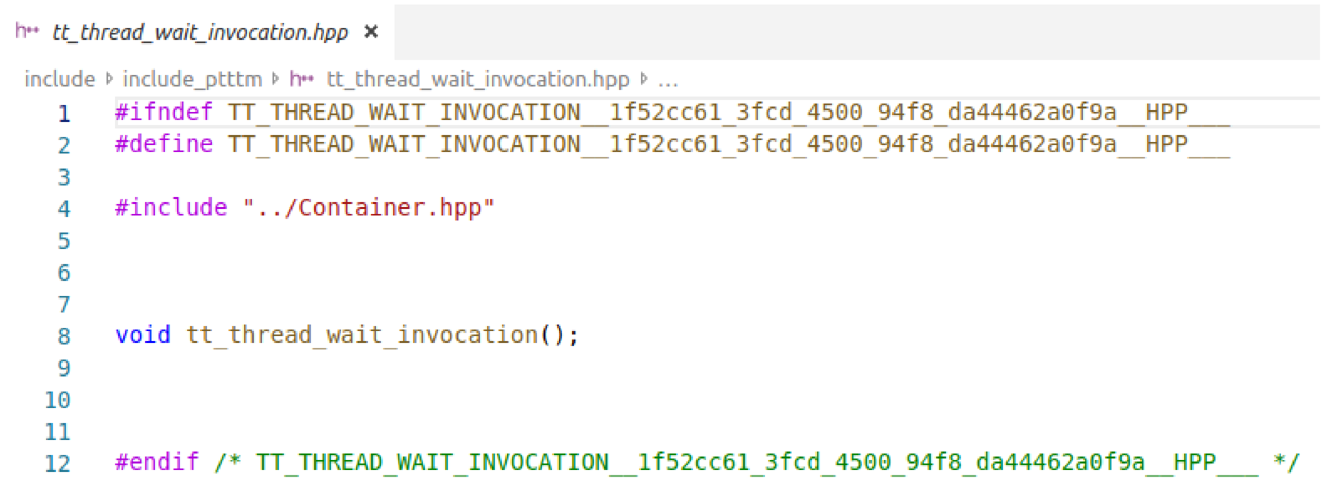
```cpp
#ifndef TT_THREAD_REGISTER__c9656d10_a146_482d_b6e6_e70e83a74c77__HPP___
#define TT_THREAD_REGISTER__c9656d10_a146_482d_b6e6_e70e83a74c77__HPP___

#include "../Container.hpp"



void tt_thread_register(size_t);



#endif /* TT_THREAD_REGISTER__c9656d10_a146_482d_b6e6_e70e83a74c77__HPP___ */
```

tt_thread_register.cpp


tt_thread_register.cpp - 03_pseudo_blood_pressure_detec
File Edit Selection View Go Debug Terminal Help
tt_thread_register.cpp ×
src › src_ptttm › tt_thread_register.cpp › ...

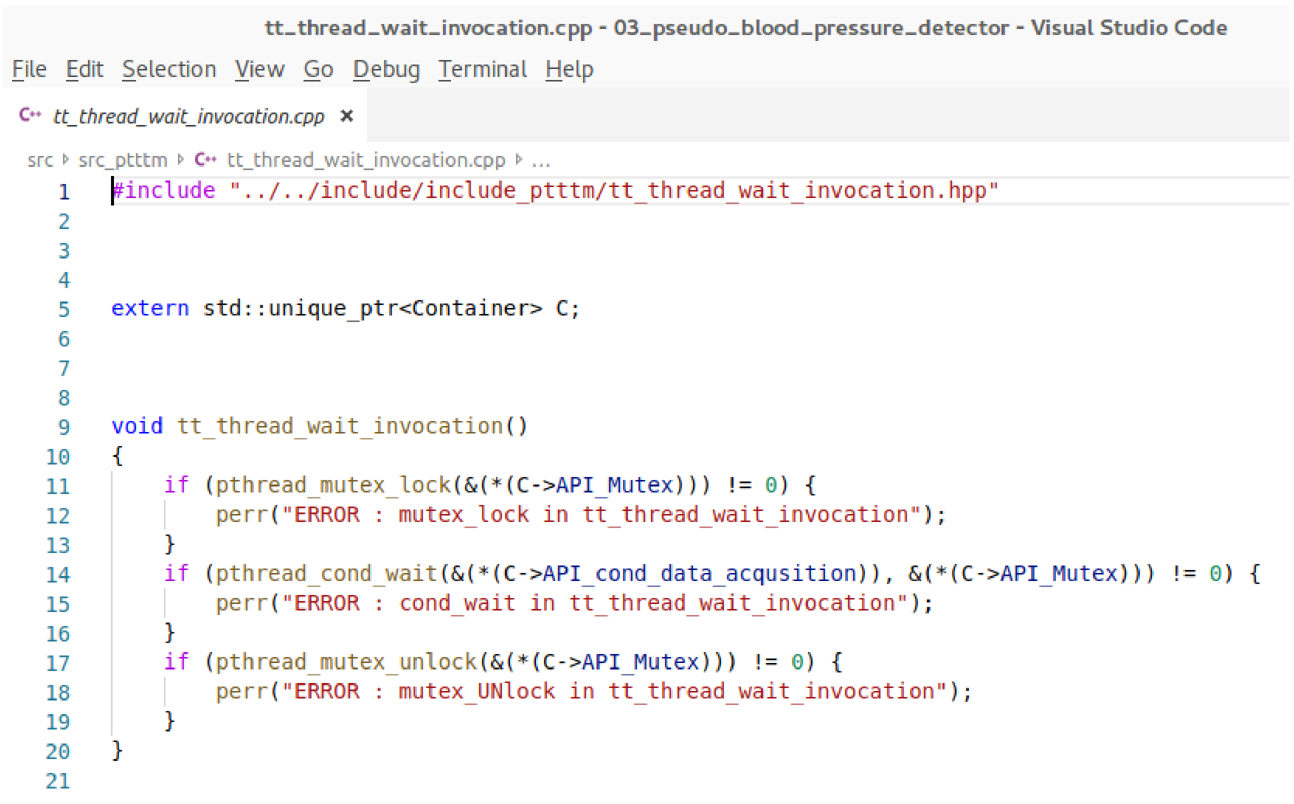
```cpp
#include "../../include/include_ptttm/tt_thread_register.hpp"



extern std::unique_ptr<Container> C;



void tt_thread_register(size_t new_period)
{
    C->TCB_storage->period = new_period;
    C->TCB_storage->time_left_to_invoke = new_period;
}

```

tt_thread_wait_invocation.hpp


tt_thread_wait_invocation.hpp ×
include › include_ptttm › tt_thread_wait_invocation.hpp › ...

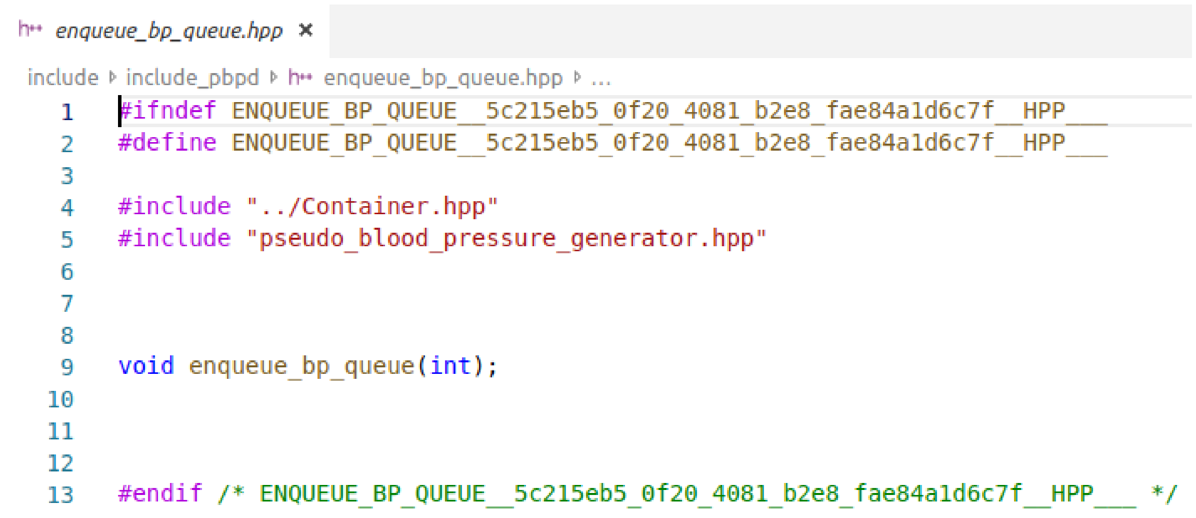
```cpp
#ifndef TT_THREAD_WAIT_INVOCATION__1f52cc61_3fcd_4500_94f8_da44462a0f9a__HPP___
#define TT_THREAD_WAIT_INVOCATION__1f52cc61_3fcd_4500_94f8_da44462a0f9a__HPP___

#include "../Container.hpp"



void tt_thread_wait_invocation();


#endif /* TT_THREAD_WAIT_INVOCATION__1f52cc61_3fcd_4500_94f8_da44462a0f9a__HPP___ */
```

tt_thread_wait_invocation.cpp


tt_thread_wait_invocation.cpp - 03_pseudo_blood_pressure_detector - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
tt_thread_wait_invocation.cpp ×
src › src_ptttm › tt_thread_wait_invocation.cpp › ...

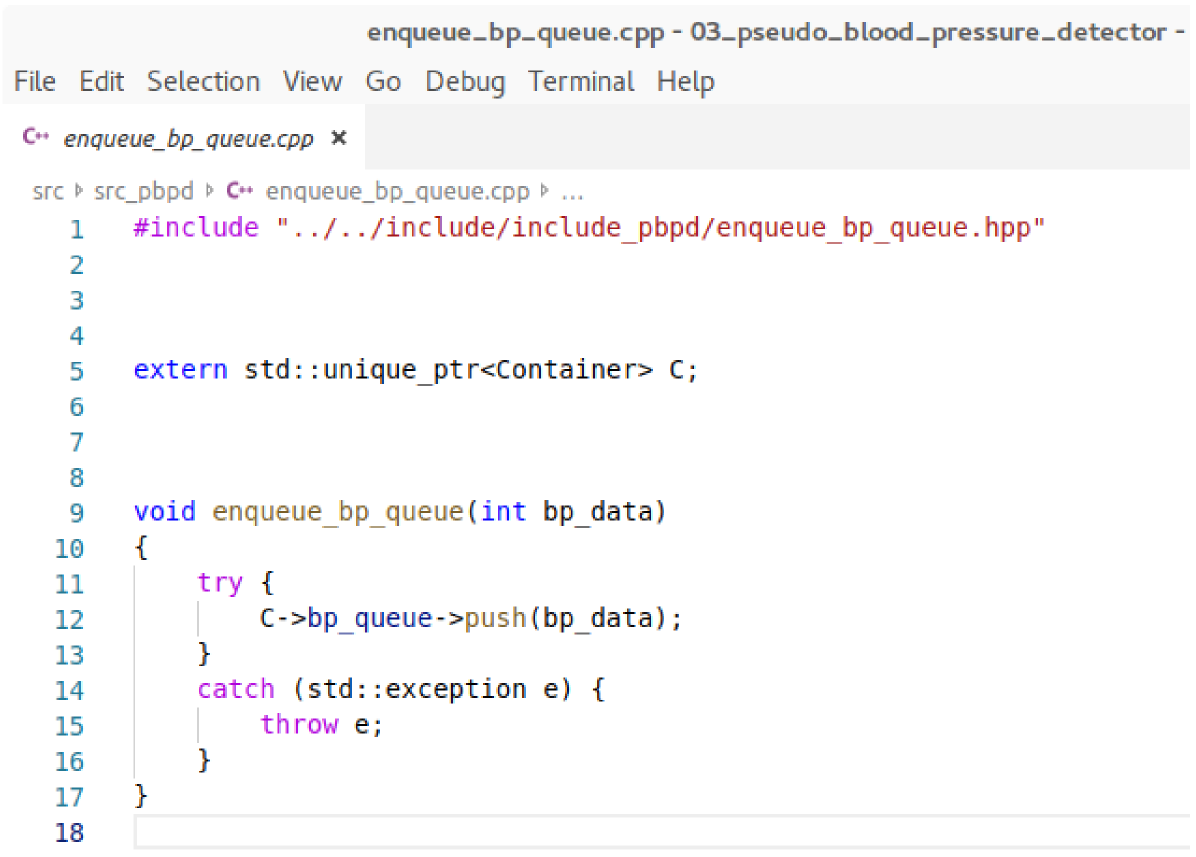
```cpp
#include "../../include/include_ptttm/tt_thread_wait_invocation.hpp"



extern std::unique_ptr<Container> C;



void tt_thread_wait_invocation()
{
    if (pthread_mutex_lock(&(*(C->API_Mutex))) != 0) {
        perr("ERROR : mutex_lock in tt_thread_wait_invocation");
    }
    if (pthread_cond_wait(&(*(C->API_cond_data_acqusition)), &(*(C->API_Mutex))) != 0) {
        perr("ERROR : cond_wait in tt_thread_wait_invocation");
    }
    if (pthread_mutex_unlock(&(*(C->API_Mutex))) != 0) {
        perr("ERROR : mutex_UNlock in tt_thread_wait_invocation");
    }
}

```

enqueue_bp_queue.hpp



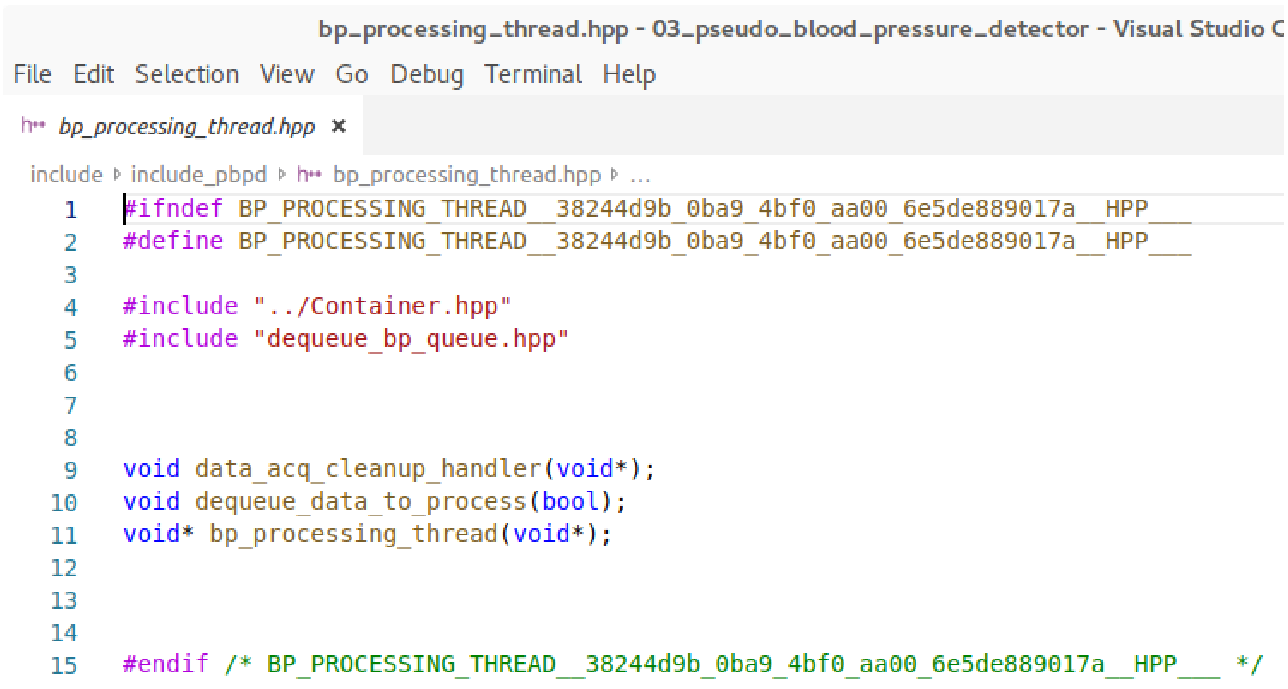
```
#ifndef ENQUEUE_BP_QUEUE__5c215eb5_0f20_4081_b2e8_fae84a1d6c7f__HPP___
#define ENQUEUE_BP_QUEUE__5c215eb5_0f20_4081_b2e8_fae84a1d6c7f__HPP___

#include "../Container.hpp"
#include "pseudo_blood_pressure_generator.hpp"



void enqueue_bp_queue(int);



#endif /* ENQUEUE_BP_QUEUE__5c215eb5_0f20_4081_b2e8_fae84a1d6c7f__HPP___ */
```

enqueue_bp_queue.cpp


enqueue_bp_queue.cpp - 03_pseudo_blood_pressure_detector -
File Edit Selection View Go Debug Terminal Help
enqueue_bp_queue.cpp ×
src › src_pbpd › enqueue_bp_queue.cpp › ...

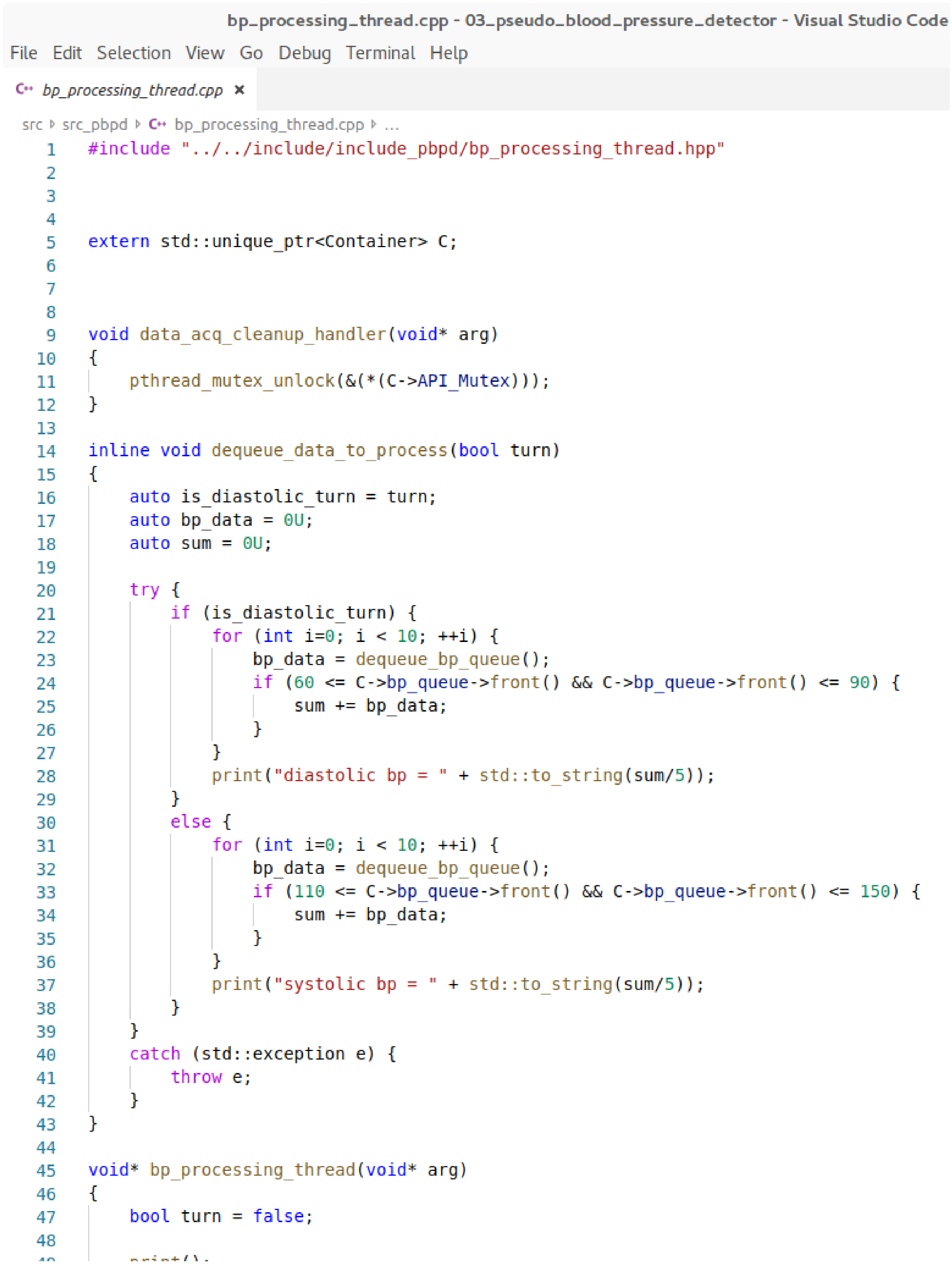
```
#include "../../include/include_pbpd/enqueue_bp_queue.hpp"



extern std::unique_ptr<Container> C;



void enqueue_bp_queue(int bp_data)
{
    try {
        C->bp_queue->push(bp_data);
    }
    catch (std::exception e) {
        throw e;
    }
}

```

bp_processing_thread.hpp


bp_processing_thread.hpp - 03_pseudo_blood_pressure_detector - Visual Studio C
File Edit Selection View Go Debug Terminal Help
bp_processing_thread.hpp
include › include_pbpd › bp_processing_thread.hpp › ...

```
#ifndef BP_PROCESSING_THREAD__38244d9b_0ba9_4bf0_aa00_6e5de889017a__HPP___
#define BP_PROCESSING_THREAD__38244d9b_0ba9_4bf0_aa00_6e5de889017a__HPP___

#include "../Container.hpp"
#include "dequeue_bp_queue.hpp"



void data_acq_cleanup_handler(void*);
void dequeue_data_to_process(bool);
void* bp_processing_thread(void*);



#endif /* BP_PROCESSING_THREAD__38244d9b_0ba9_4bf0_aa00_6e5de889017a__HPP___ */
```

bp_processing_thread.cpp



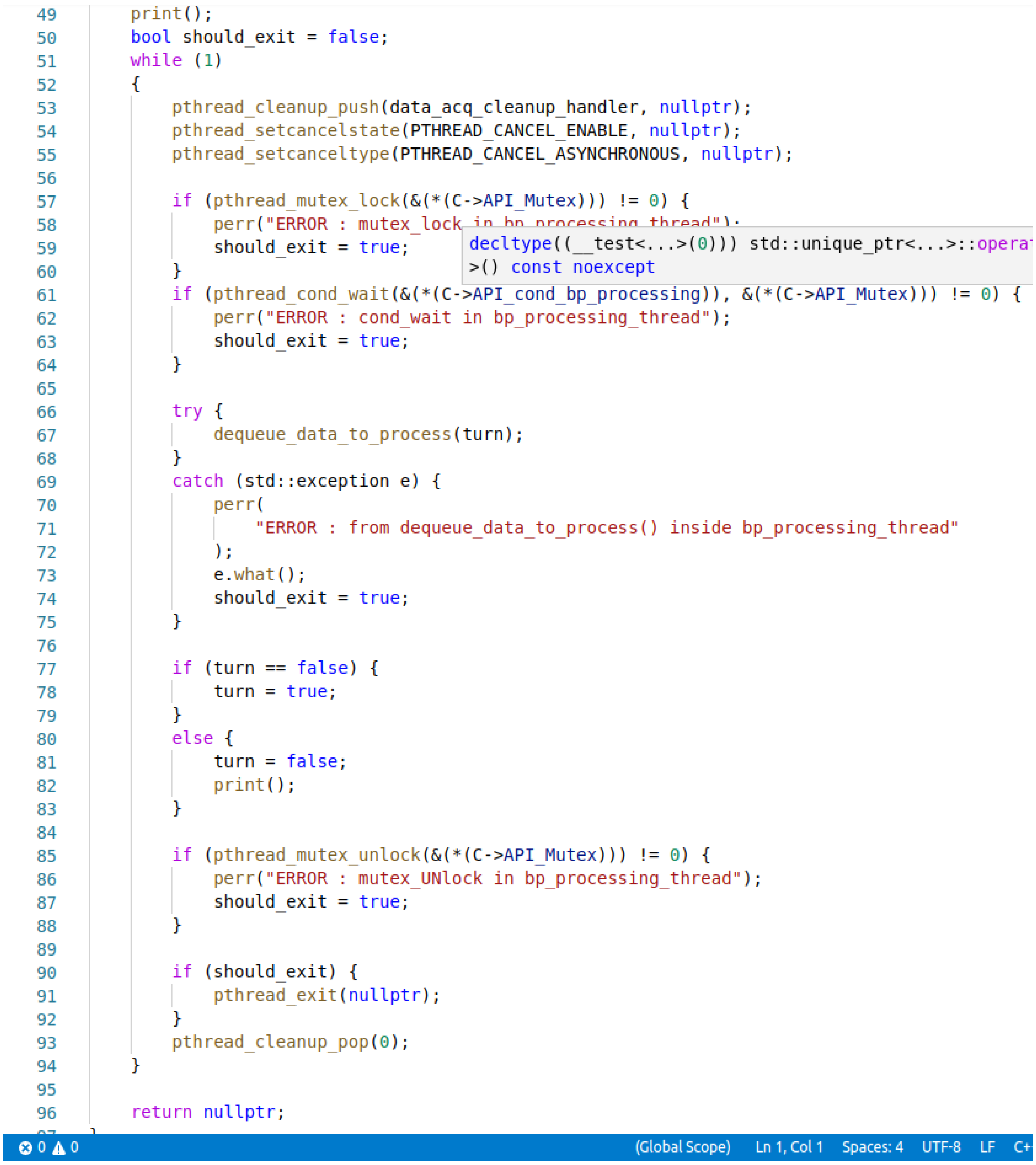
bp_processing_thread.cpp - 03_pseudo_blood_pressure_detector - Visual Studio Code

File Edit Selection View Go Debug Terminal Help

bp_processing_thread.cpp

src › src_pbpd › bp_processing_thread.cpp › ...

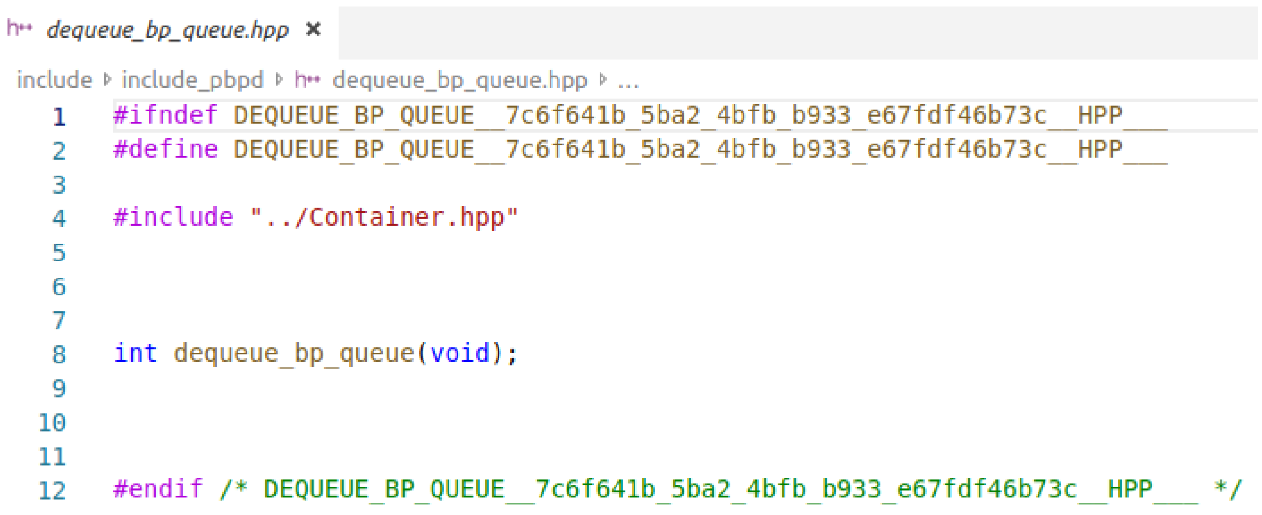
```cpp
#include "../../include/include_pbpd/bp_processing_thread.hpp"



extern std::unique_ptr<Container> C;



void data_acq_cleanup_handler(void* arg)
{
    pthread_mutex_unlock(&(*(C->API_Mutex)));
}

inline void dequeue_data_to_process(bool turn)
{
    auto is_diastolic_turn = turn;
    auto bp_data = 0U;
    auto sum = 0U;

    try {
        if (is_diastolic_turn) {
            for (int i=0; i < 10; ++i) {
                bp_data = dequeue_bp_queue();
                if (60 <= C->bp_queue->front() && C->bp_queue->front() <= 90) {
                    sum += bp_data;
                }
            }
            print("diastolic bp = " + std::to_string(sum/5));
        }
        else {
            for (int i=0; i < 10; ++i) {
                bp_data = dequeue_bp_queue();
                if (110 <= C->bp_queue->front() && C->bp_queue->front() <= 150) {
                    sum += bp_data;
                }
            }
            print("systolic bp = " + std::to_string(sum/5));
        }
    }
    catch (std::exception e) {
        throw e;
    }
}

void* bp_processing_thread(void* arg)
{
    bool turn = false;

```



```
    print();
    bool should_exit = false;
    while (1)
    {
        pthread_cleanup_push(data_acq_cleanup_handler, nullptr);
        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, nullptr);
        pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, nullptr);

        if (pthread_mutex_lock(&(*(C->API_Mutex))) != 0) {
            perr("ERROR : mutex_lock in bp_processing_thread");
            should_exit = true;
        }
        if (pthread_cond_wait(&(*(C->API_cond_bp_processing)), &(*(C->API_Mutex))) != 0) {
            perr("ERROR : cond_wait in bp_processing_thread");
            should_exit = true;
        }

        try {
            dequeue_data_to_process(turn);
        }
        catch (std::exception e) {
            perr(
                "ERROR : from dequeue_data_to_process() inside bp_processing_thread"
            );
            e.what();
            should_exit = true;
        }

        if (turn == false) {
            turn = true;
        }
        else {
            turn = false;
            print();
        }

        if (pthread_mutex_unlock(&(*(C->API_Mutex))) != 0) {
            perr("ERROR : mutex_UNlock in bp_processing_thread");
            should_exit = true;
        }

        if (should_exit) {
            pthread_exit(nullptr);
        }
        pthread_cleanup_pop(0);
    }

    return nullptr;
}
```

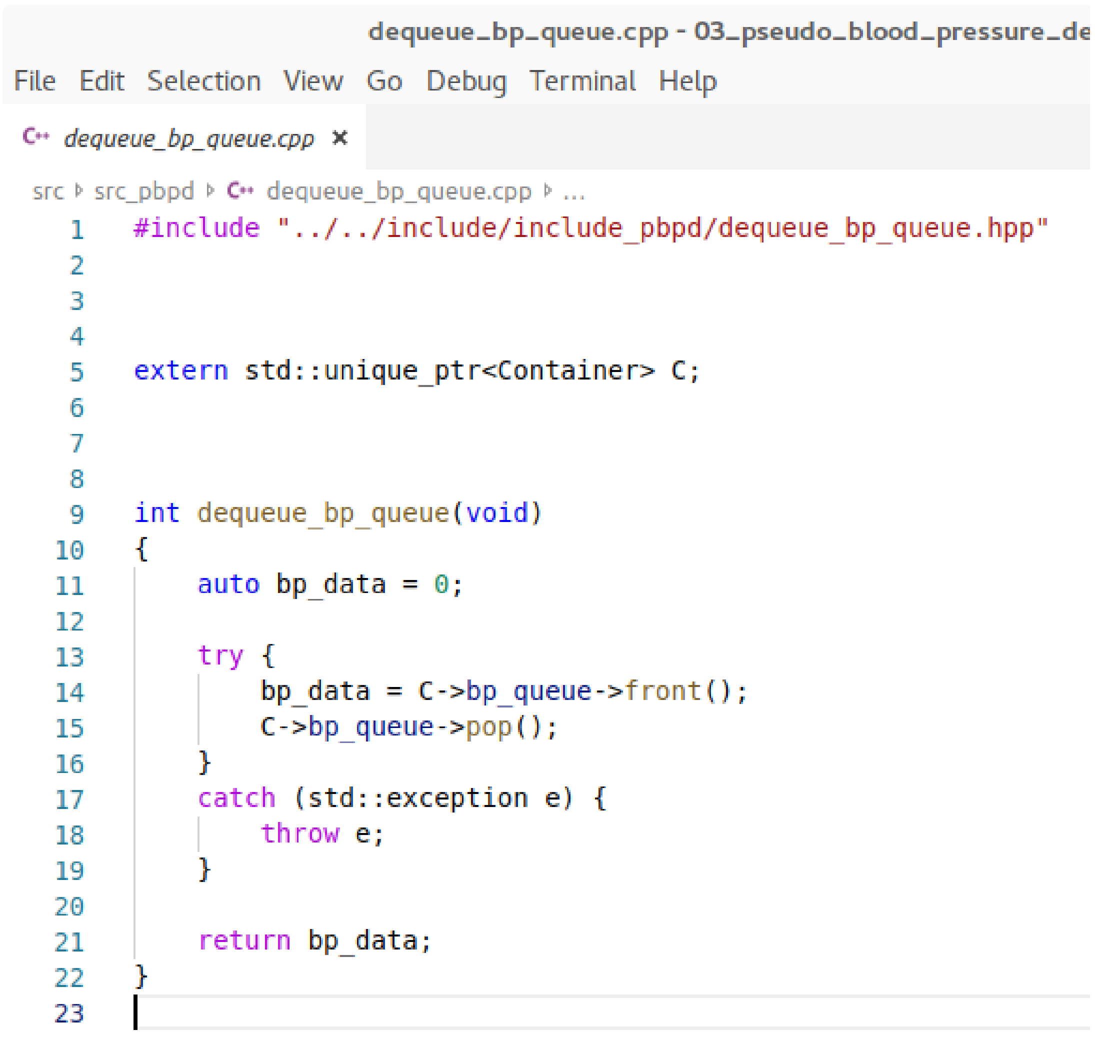
```
decltype((__test<...>(0))) std::unique_ptr<...>::opera
>() const noexcept
```

dequeue_bp_queue.hpp



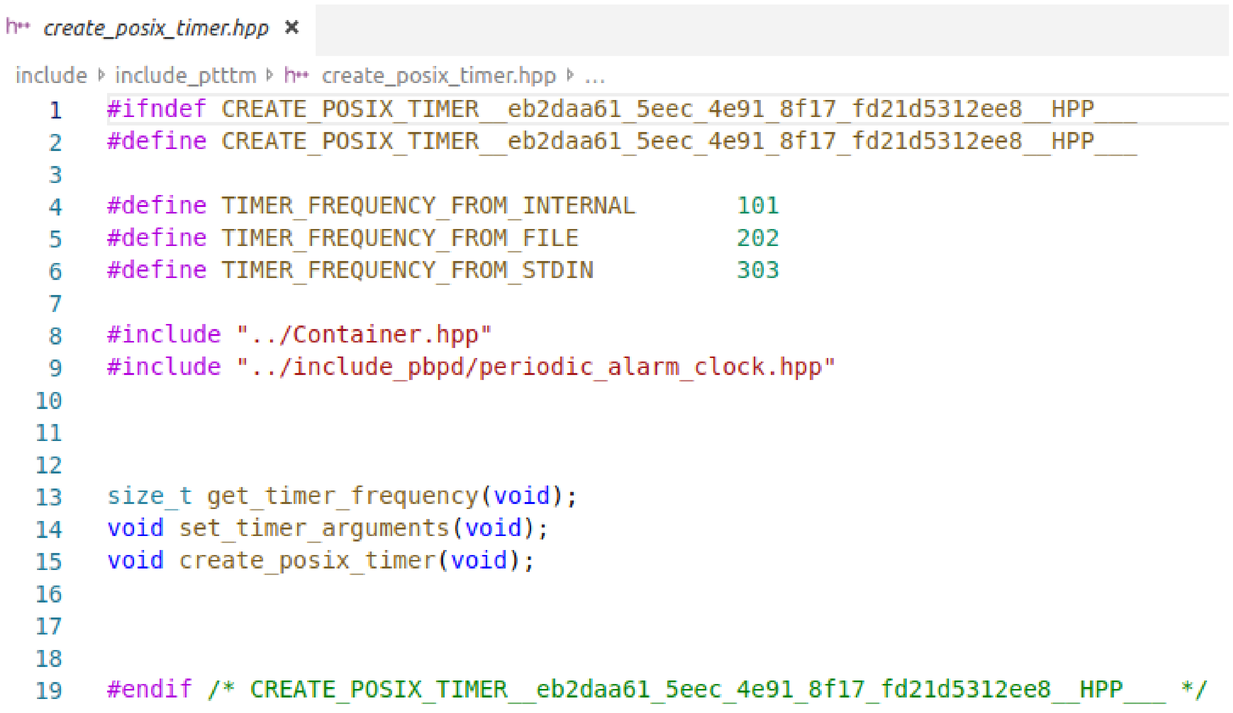
```
#ifndef DEQUEUE_BP_QUEUE__7c6f641b_5ba2_4bfb_b933_e67fdf46b73c__HPP___
#define DEQUEUE_BP_QUEUE__7c6f641b_5ba2_4bfb_b933_e67fdf46b73c__HPP___

#include "../Container.hpp"



int dequeue_bp_queue(void);



#endif /* DEQUEUE_BP_QUEUE__7c6f641b_5ba2_4bfb_b933_e67fdf46b73c__HPP___ */
```

dequeue_bp_queue.cpp


dequeue_bp_queue.cpp - 03_pseudo_blood_pressure_de
File Edit Selection View Go Debug Terminal Help
dequeue_bp_queue.cpp
src › src_pbpd › dequeue_bp_queue.cpp › ...

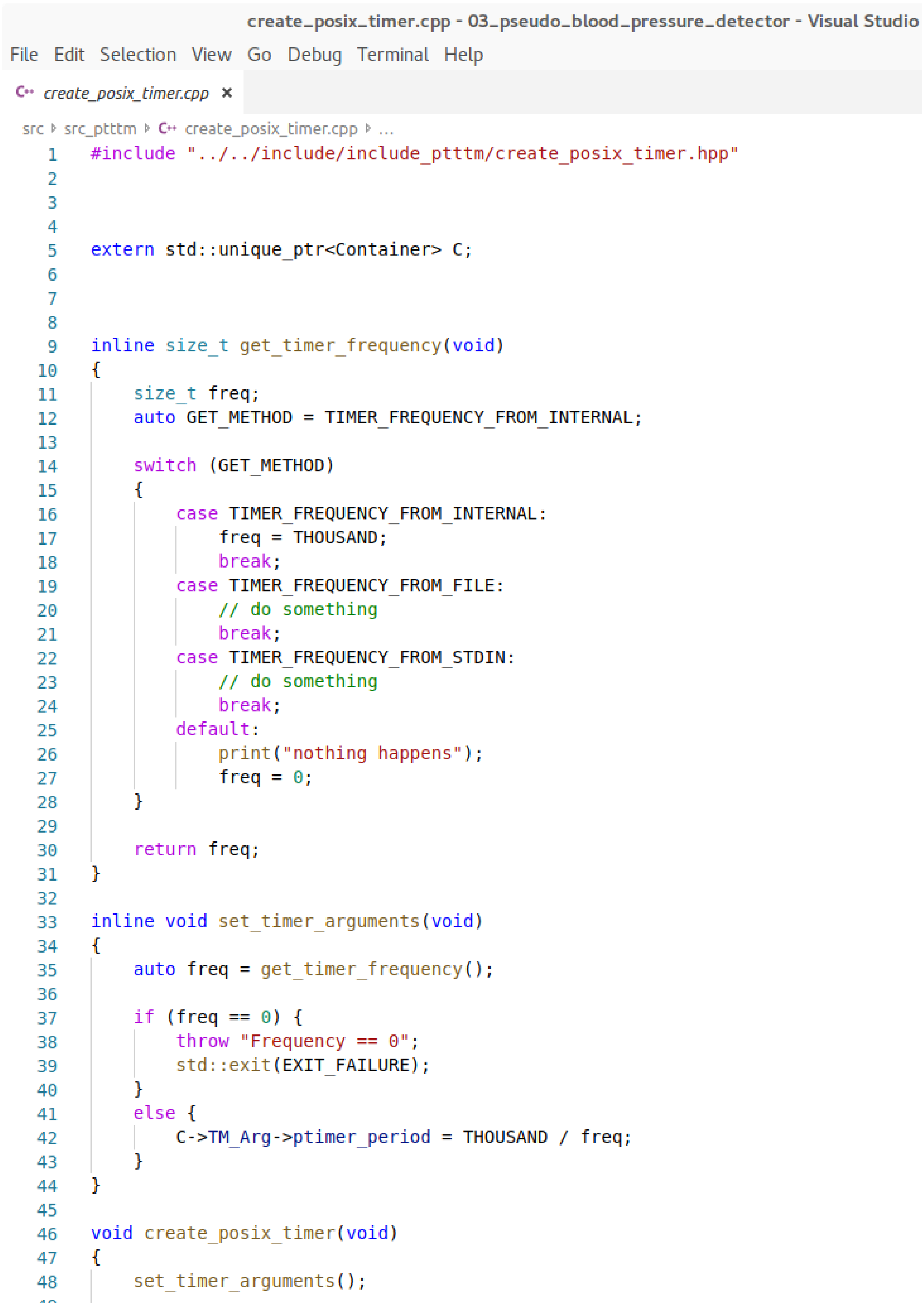
```
#include "../../include/include_pbpd/dequeue_bp_queue.hpp"



extern std::unique_ptr<Container> C;



int dequeue_bp_queue(void)
{
    auto bp_data = 0;

    try {
        bp_data = C->bp_queue->front();
        C->bp_queue->pop();
    }
    catch (std::exception e) {
        throw e;
    }

    return bp_data;
}

```

create_posix_timer.hpp

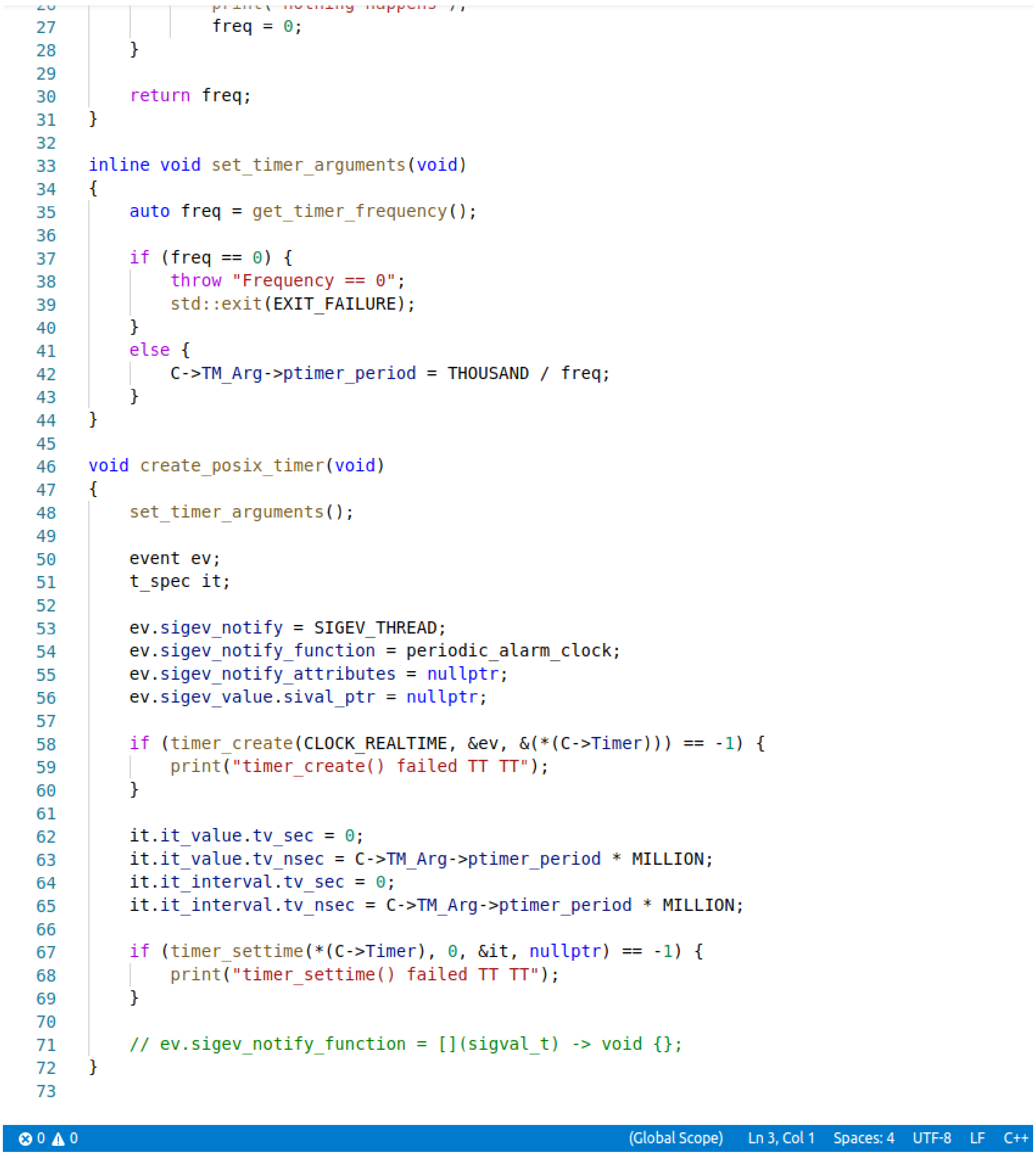


```
#ifndef CREATE_POSIX_TIMER__eb2daa61_5eec_4e91_8f17_fd21d5312ee8__HPP___
#define CREATE_POSIX_TIMER__eb2daa61_5eec_4e91_8f17_fd21d5312ee8__HPP___

#define TIMER_FREQUENCY_FROM_INTERNAL       101
#define TIMER_FREQUENCY_FROM_FILE           202
#define TIMER_FREQUENCY_FROM_STDIN          303

#include "../Container.hpp"
#include "../include_pbpd/periodic_alarm_clock.hpp"



size_t get_timer_frequency(void);
void set_timer_arguments(void);
void create_posix_timer(void);



#endif /* CREATE_POSIX_TIMER__eb2daa61_5eec_4e91_8f17_fd21d5312ee8__HPP___ */
```

create_posix_timer.cpp

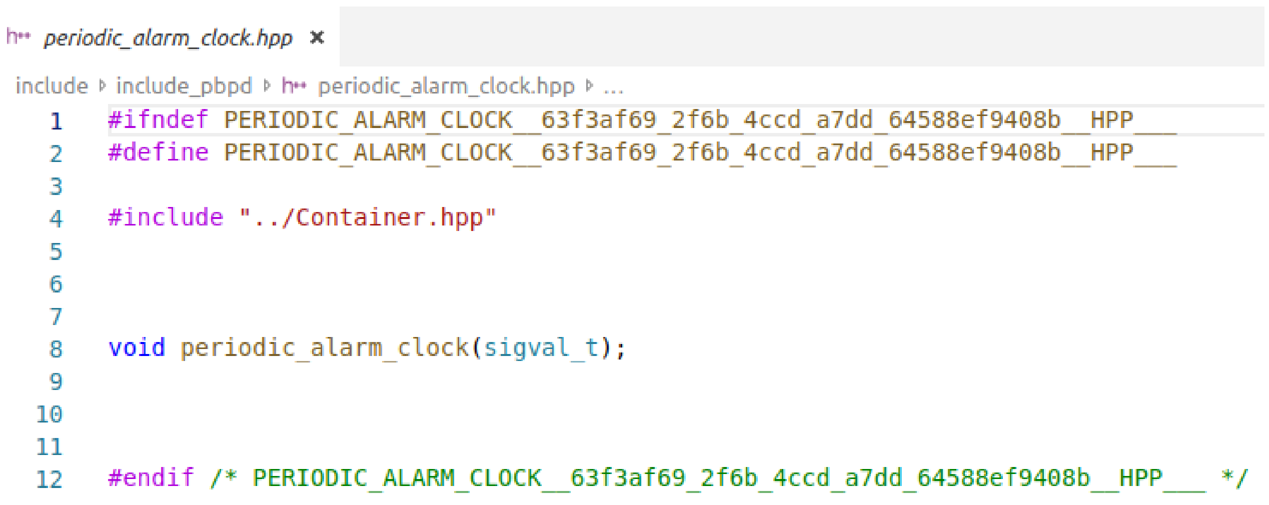

create_posix_timer.cpp - 03_pseudo_blood_pressure_detector - Visual Studio
File Edit Selection View Go Debug Terminal Help
create_posix_timer.cpp
src › src_ptttm › create_posix_timer.cpp › ...

```cpp
#include "../../include/include_ptttm/create_posix_timer.hpp"



extern std::unique_ptr<Container> C;



inline size_t get_timer_frequency(void)
{
    size_t freq;
    auto GET_METHOD = TIMER_FREQUENCY_FROM_INTERNAL;

    switch (GET_METHOD)
    {
        case TIMER_FREQUENCY_FROM_INTERNAL:
            freq = THOUSAND;
            break;
        case TIMER_FREQUENCY_FROM_FILE:
            // do something
            break;
        case TIMER_FREQUENCY_FROM_STDIN:
            // do something
            break;
        default:
            print("nothing happens");
            freq = 0;
    }

    return freq;
}

inline void set_timer_arguments(void)
{
    auto freq = get_timer_frequency();

    if (freq == 0) {
        throw "Frequency == 0";
        std::exit(EXIT_FAILURE);
    }
    else {
        C->TM_Arg->ptimer_period = THOUSAND / freq;
    }
}

void create_posix_timer(void)
{
    set_timer_arguments();
```



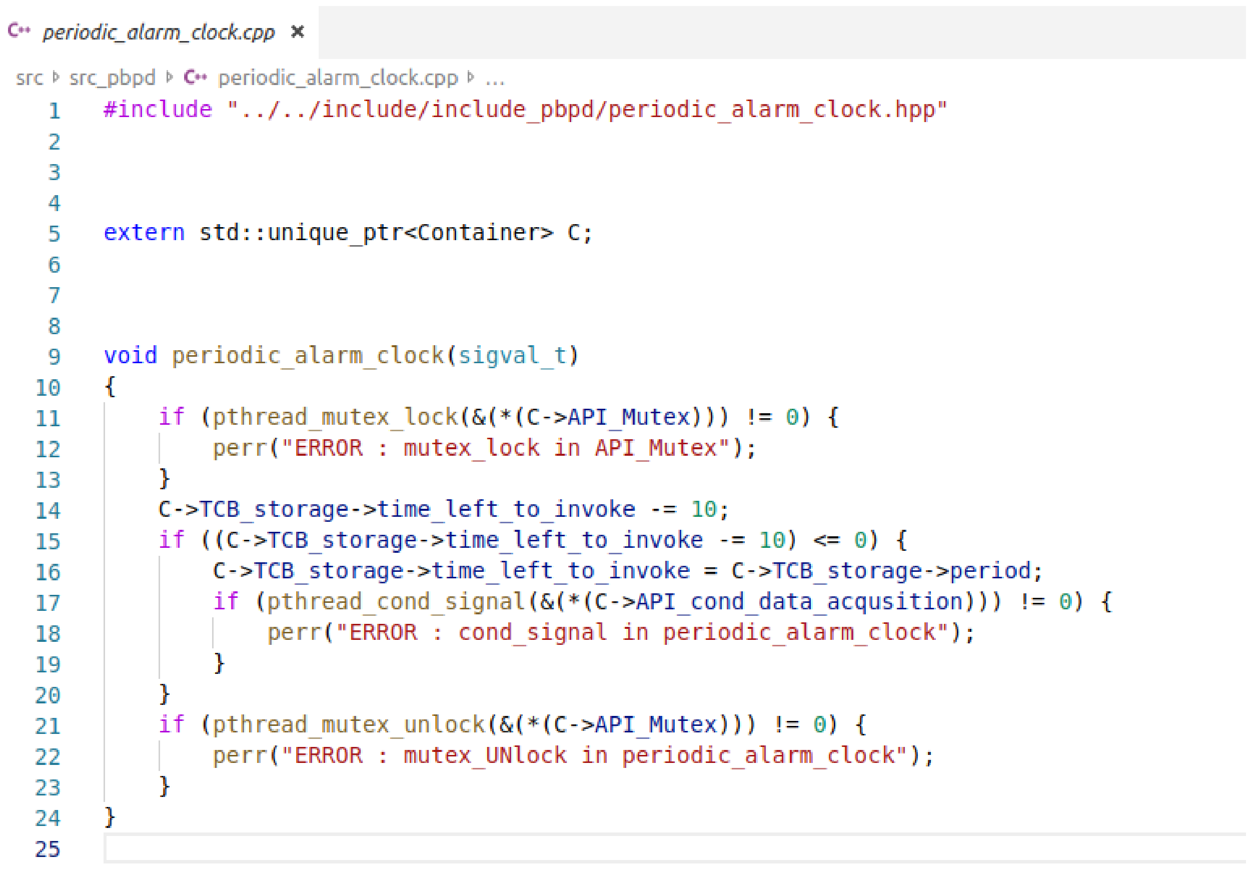
```cpp
            print("nothing happens");
            freq = 0;
    }

    return freq;
}

inline void set_timer_arguments(void)
{
    auto freq = get_timer_frequency();

    if (freq == 0) {
        throw "Frequency == 0";
        std::exit(EXIT_FAILURE);
    }
    else {
        C->TM_Arg->ptimer_period = THOUSAND / freq;
    }
}

void create_posix_timer(void)
{
    set_timer_arguments();

    event ev;
    t_spec it;

    ev.sigev_notify = SIGEV_THREAD;
    ev.sigev_notify_function = periodic_alarm_clock;
    ev.sigev_notify_attributes = nullptr;
    ev.sigev_value.sival_ptr = nullptr;

    if (timer_create(CLOCK_REALTIME, &ev, &(*(C->Timer))) == -1) {
        print("timer_create() failed TT TT");
    }

    it.it_value.tv_sec = 0;
    it.it_value.tv_nsec = C->TM_Arg->ptimer_period * MILLION;
    it.it_interval.tv_sec = 0;
    it.it_interval.tv_nsec = C->TM_Arg->ptimer_period * MILLION;

    if (timer_settime(*(C->Timer), 0, &it, nullptr) == -1) {
        print("timer_settime() failed TT TT");
    }

    // ev.sigev_notify_function = [](sigval_t) -> void {};
}
```

periodic_alarm_clock.hpp



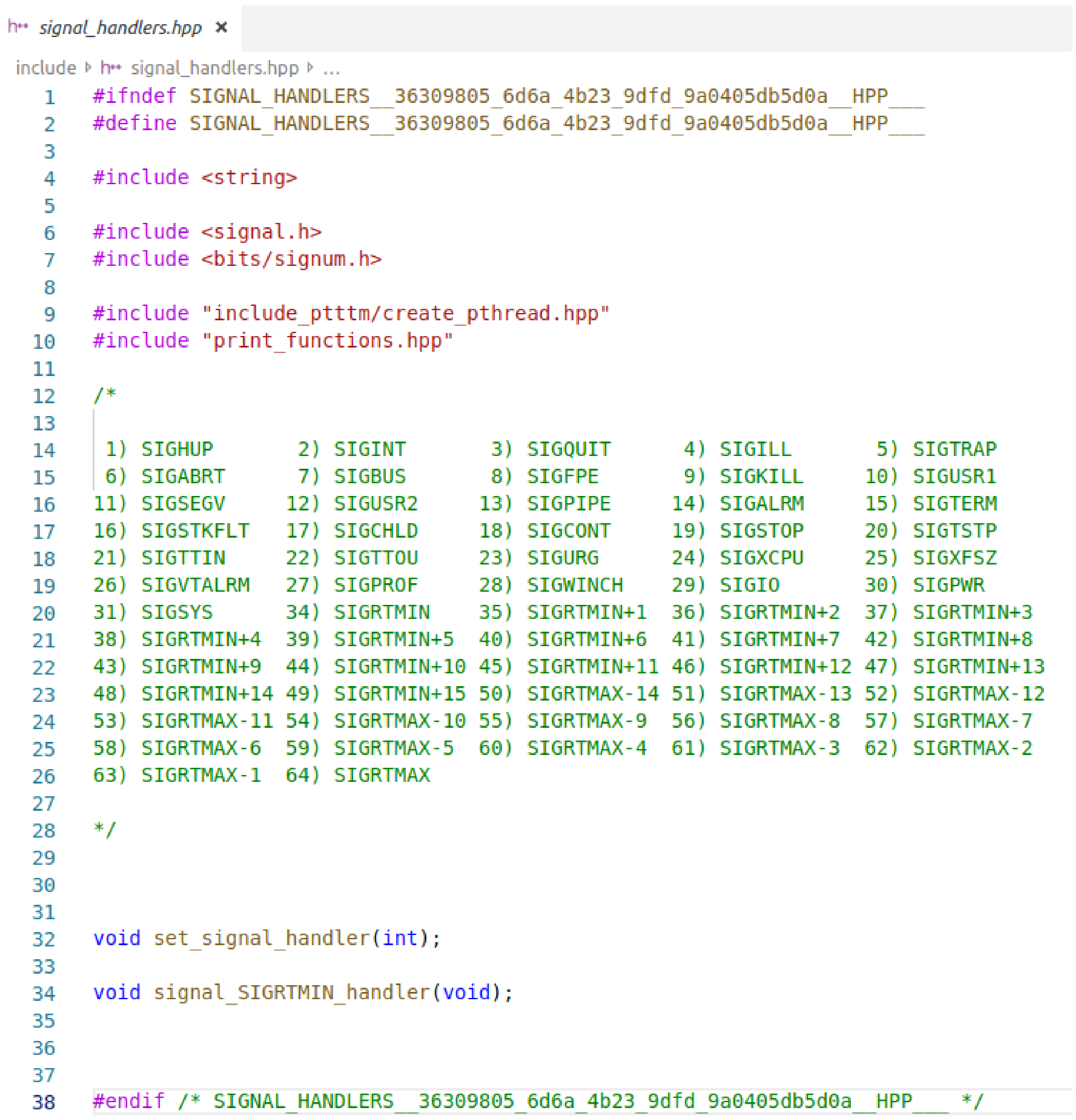
```cpp
#ifndef PERIODIC_ALARM_CLOCK__63f3af69_2f6b_4ccd_a7dd_64588ef9408b__HPP___
#define PERIODIC_ALARM_CLOCK__63f3af69_2f6b_4ccd_a7dd_64588ef9408b__HPP___

#include "../Container.hpp"



void periodic_alarm_clock(sigval_t);



#endif /* PERIODIC_ALARM_CLOCK__63f3af69_2f6b_4ccd_a7dd_64588ef9408b__HPP___ */
```

periodic_alarm_clock.cpp



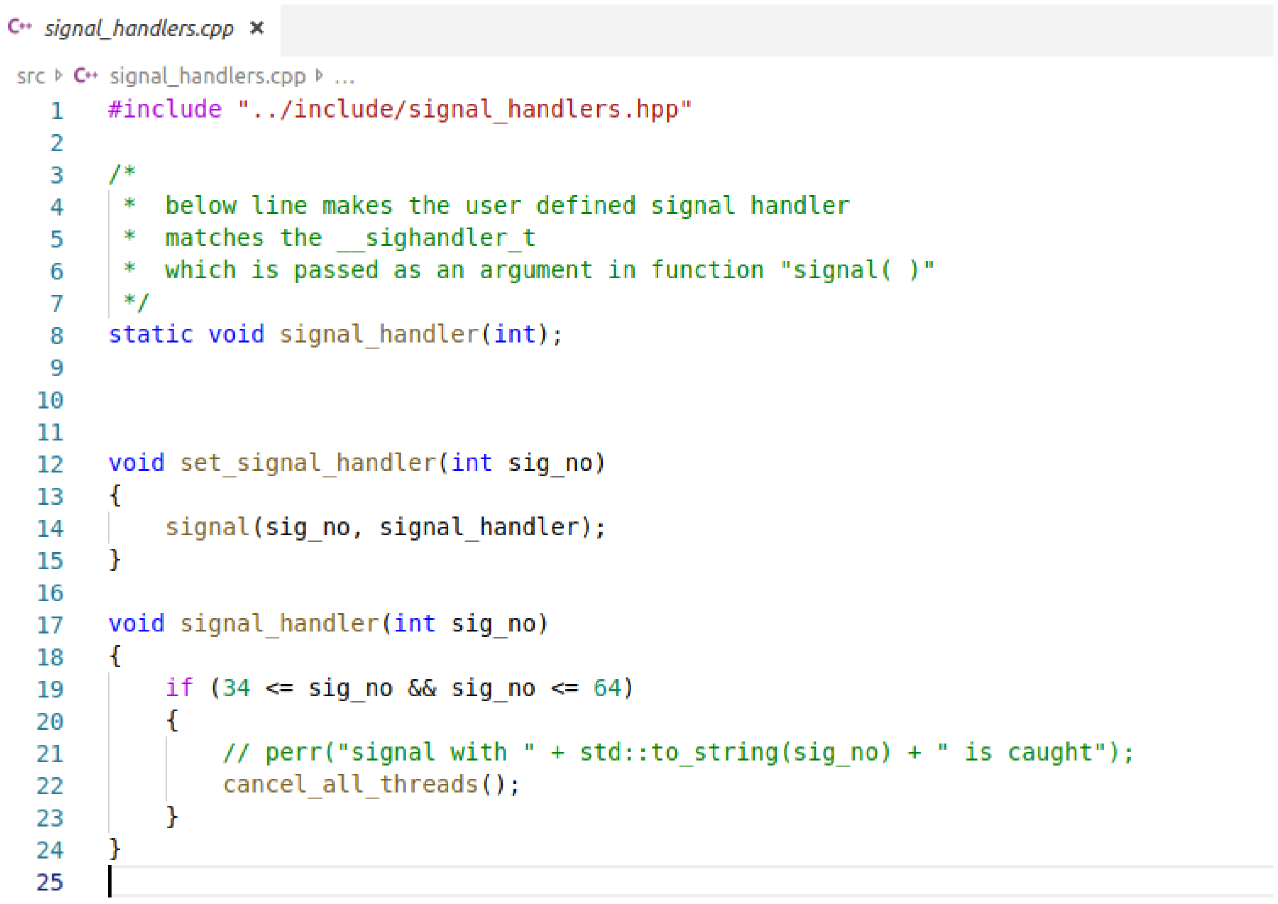
```cpp
#include "../../include/include_pbpd/periodic_alarm_clock.hpp"



extern std::unique_ptr<Container> C;



void periodic_alarm_clock(sigval_t)
{
    if (pthread_mutex_lock(&(*(C->API_Mutex))) != 0) {
        perr("ERROR : mutex_lock in API_Mutex");
    }
    C->TCB_storage->time_left_to_invoke -= 10;
    if ((C->TCB_storage->time_left_to_invoke -= 10) <= 0) {
        C->TCB_storage->time_left_to_invoke = C->TCB_storage->period;
        if (pthread_cond_signal(&(*(C->API_cond_data_acqusition))) != 0) {
            perr("ERROR : cond_signal in periodic_alarm_clock");
        }
    }
    if (pthread_mutex_unlock(&(*(C->API_Mutex))) != 0) {
        perr("ERROR : mutex_UNlock in periodic_alarm_clock");
    }
}

```

signal_handler.hpp



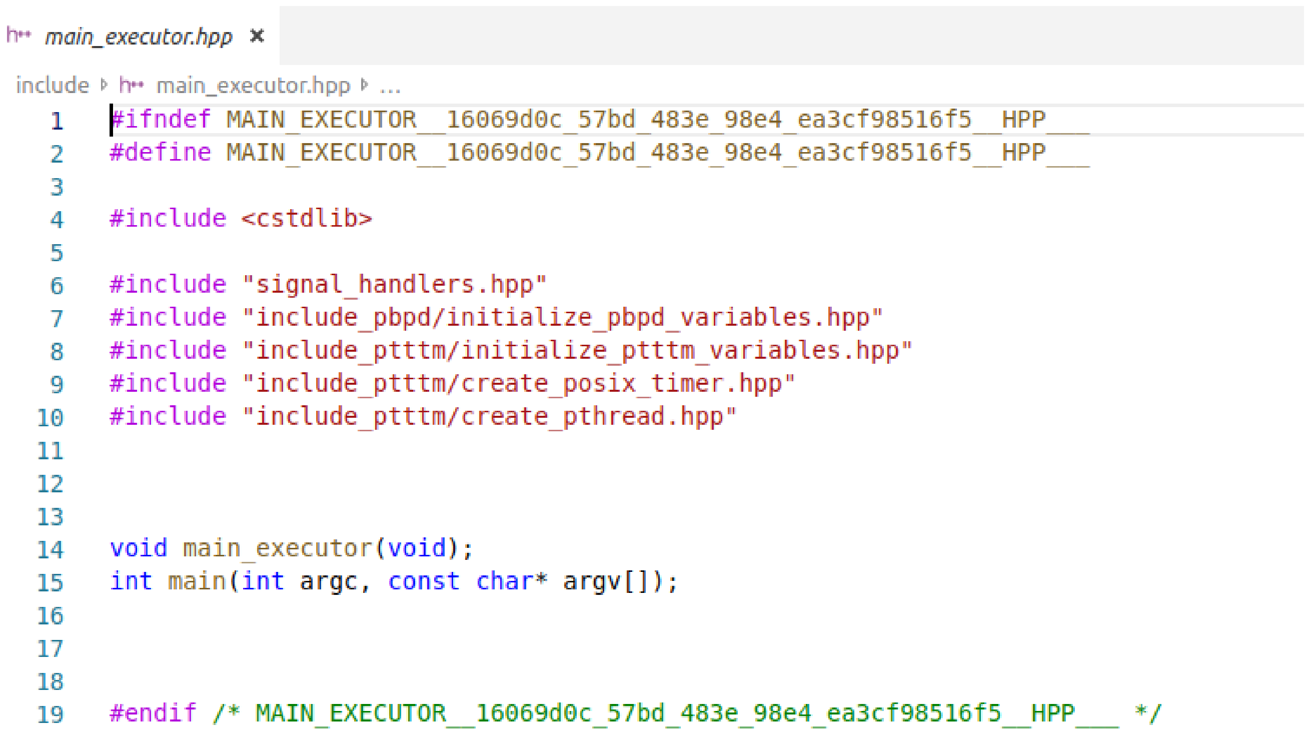
```cpp
#ifndef SIGNAL_HANDLERS__36309805_6d6a_4b23_9dfd_9a0405db5d0a__HPP___
#define SIGNAL_HANDLERS__36309805_6d6a_4b23_9dfd_9a0405db5d0a__HPP___

#include <string>

#include <signal.h>
#include <bits/signum.h>

#include "include_ptttm/create_pthread.hpp"
#include "print_functions.hpp"

/*

 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX

*/



void set_signal_handler(int);

void signal_SIGRTMIN_handler(void);



#endif /* SIGNAL_HANDLERS__36309805_6d6a_4b23_9dfd_9a0405db5d0a__HPP___ */
```

signal_handler.cpp



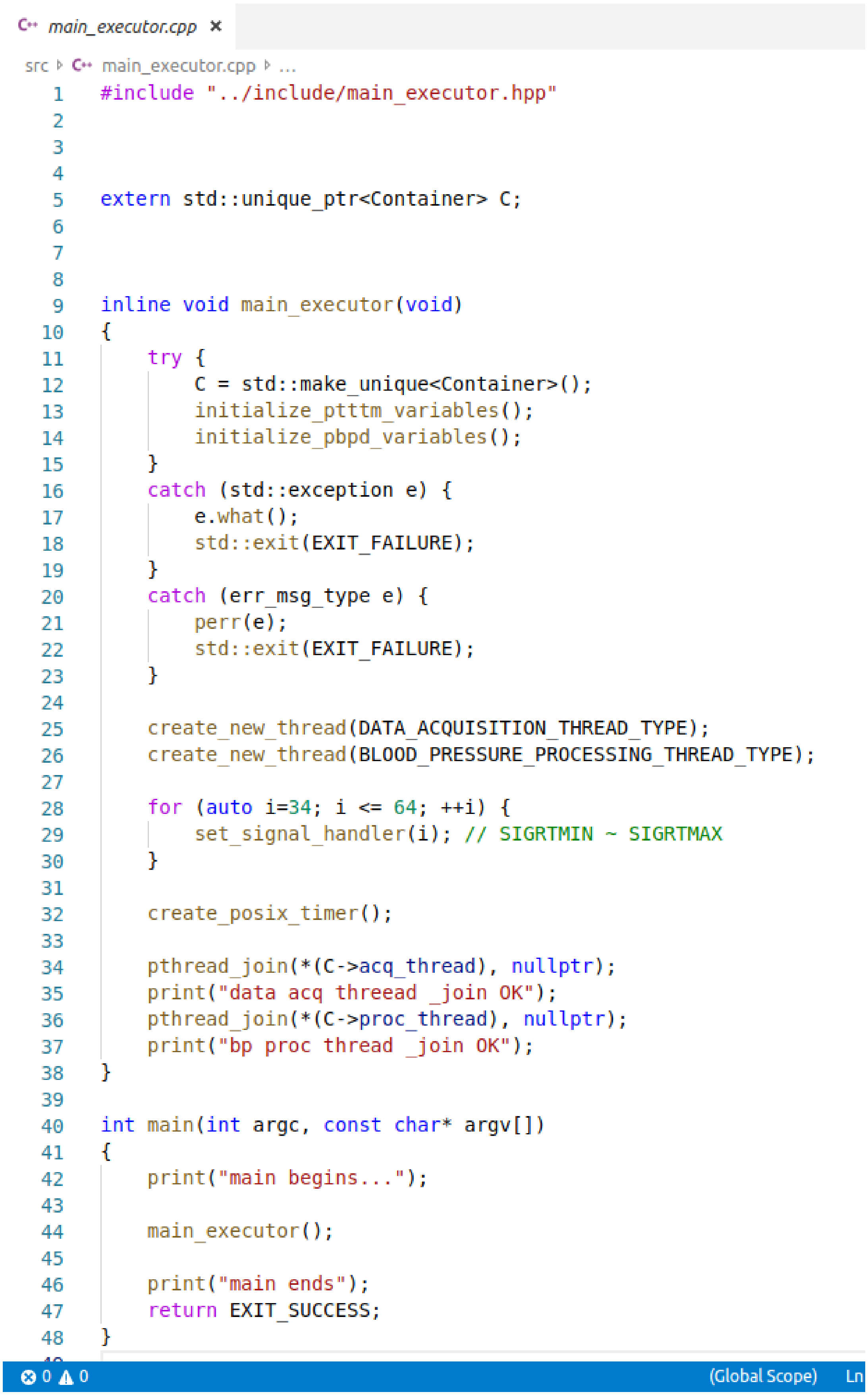
```cpp
#include "../include/signal_handlers.hpp"

/*
 *  below line makes the user defined signal handler
 *  matches the __sighandler_t
 *  which is passed as an argument in function "signal( )"
 */
static void signal_handler(int);



void set_signal_handler(int sig_no)
{
    signal(sig_no, signal_handler);
}

void signal_handler(int sig_no)
{
    if (34 <= sig_no && sig_no <= 64)
    {
        // perr("signal with " + std::to_string(sig_no) + " is caught");
        cancel_all_threads();
    }
}

```

main_executor.hpp



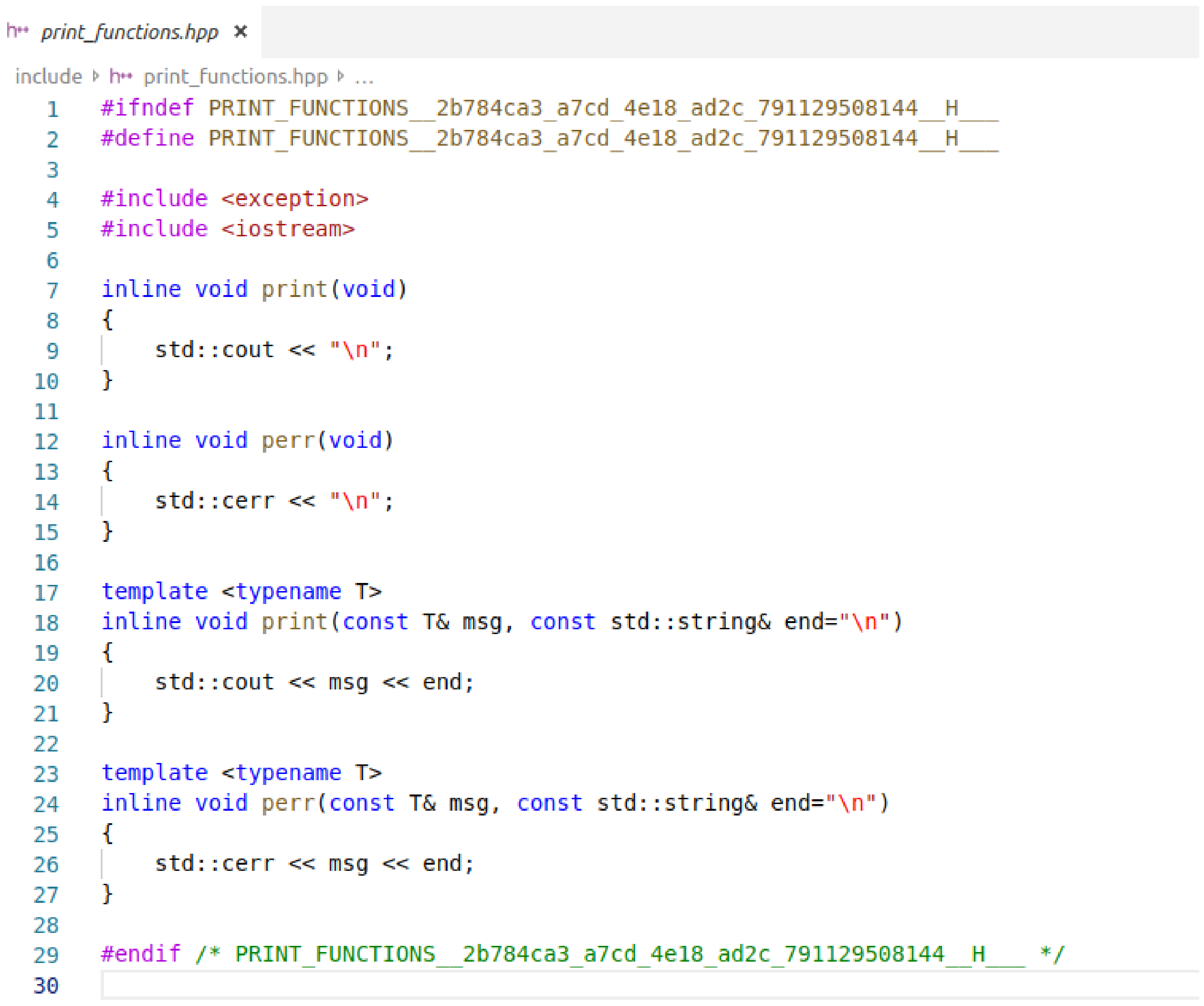
```cpp
#ifndef MAIN_EXECUTOR__16069d0c_57bd_483e_98e4_ea3cf98516f5__HPP___
#define MAIN_EXECUTOR__16069d0c_57bd_483e_98e4_ea3cf98516f5__HPP___

#include <cstdlib>

#include "signal_handlers.hpp"
#include "include_pbpd/initialize_pbpd_variables.hpp"
#include "include_ptttm/initialize_ptttm_variables.hpp"
#include "include_ptttm/create_posix_timer.hpp"
#include "include_ptttm/create_pthread.hpp"



void main_executor(void);
int main(int argc, const char* argv[]);



#endif /* MAIN_EXECUTOR__16069d0c_57bd_483e_98e4_ea3cf98516f5__HPP___ */
```

main_executor.cpp



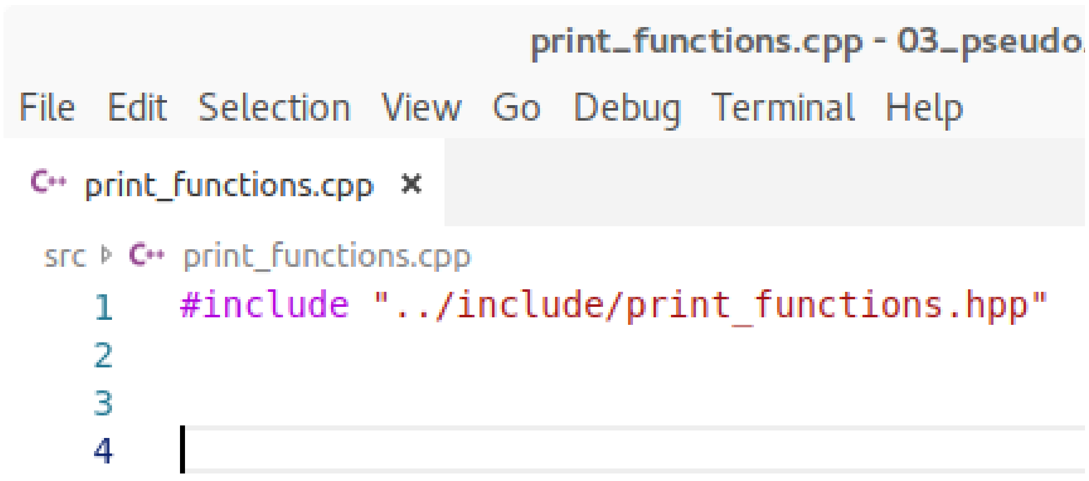
```cpp
#include "../include/main_executor.hpp"



extern std::unique_ptr<Container> C;



inline void main_executor(void)
{
    try {
        C = std::make_unique<Container>();
        initialize_ptttm_variables();
        initialize_pbpd_variables();
    }
    catch (std::exception e) {
        e.what();
        std::exit(EXIT_FAILURE);
    }
    catch (err_msg_type e) {
        perr(e);
        std::exit(EXIT_FAILURE);
    }

    create_new_thread(DATA_ACQUISITION_THREAD_TYPE);
    create_new_thread(BLOOD_PRESSURE_PROCESSING_THREAD_TYPE);

    for (auto i=34; i <= 64; ++i) {
        set_signal_handler(i); // SIGRTMIN ~ SIGRTMAX
    }

    create_posix_timer();

    pthread_join(*(C->acq_thread), nullptr);
    print("data acq threead _join OK");
    pthread_join(*(C->proc_thread), nullptr);
    print("bp proc thread _join OK");
}

int main(int argc, const char* argv[])
{
    print("main begins...");

    main_executor();

    print("main ends");
    return EXIT_SUCCESS;
}
```

print_functions.hpp



```cpp
#ifndef PRINT_FUNCTIONS__2b784ca3_a7cd_4e18_ad2c_791129508144__H___
#define PRINT_FUNCTIONS__2b784ca3_a7cd_4e18_ad2c_791129508144__H___

#include <exception>
#include <iostream>

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

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

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

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

#endif /* PRINT_FUNCTIONS__2b784ca3_a7cd_4e18_ad2c_791129508144__H___ */

```

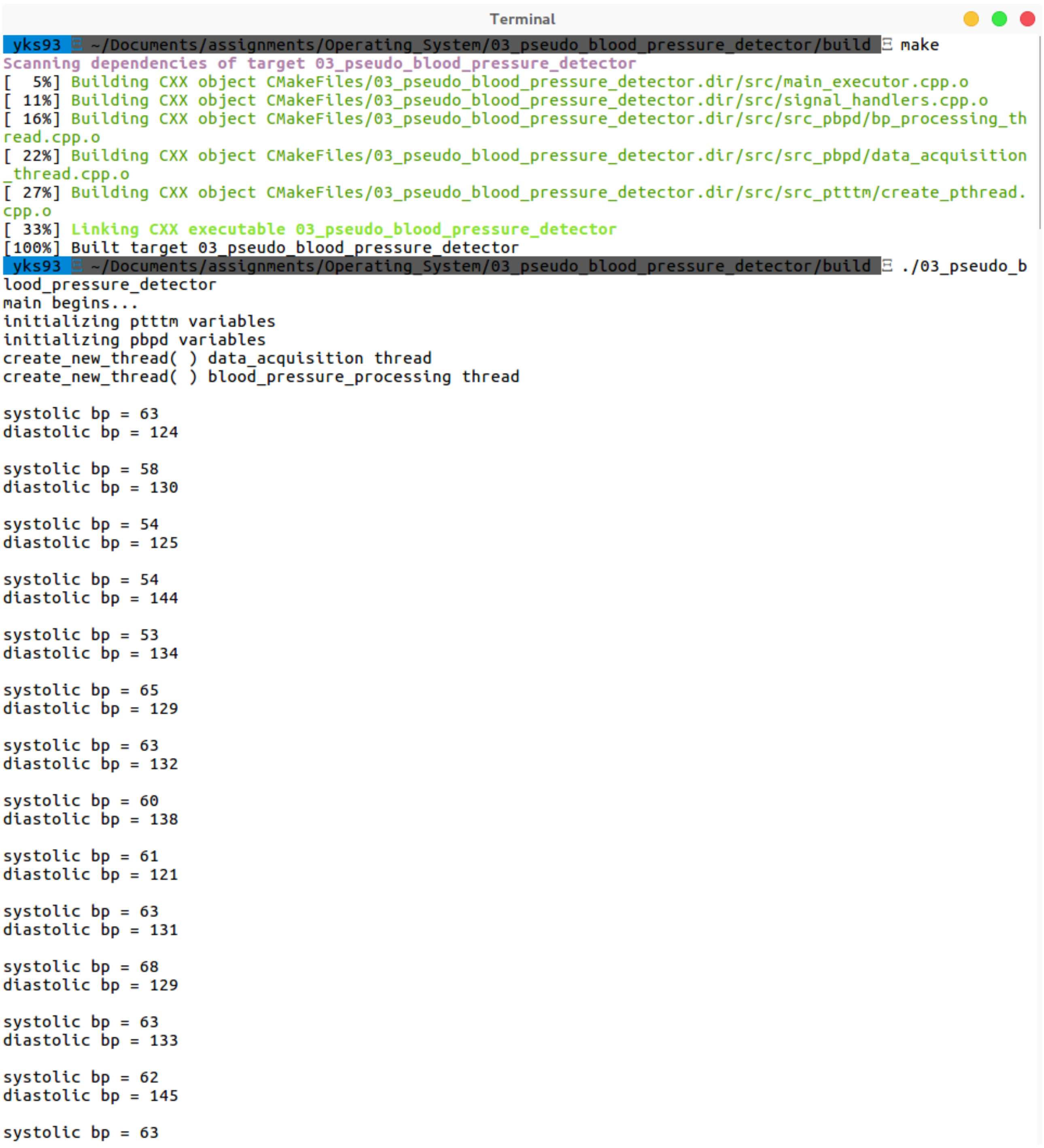
print_functions.cpp


print_functions.cpp - 03_pseudo
File Edit Selection View Go Debug Terminal Help
print_functions.cpp
src › print_functions.cpp

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


```

5. Results Capture

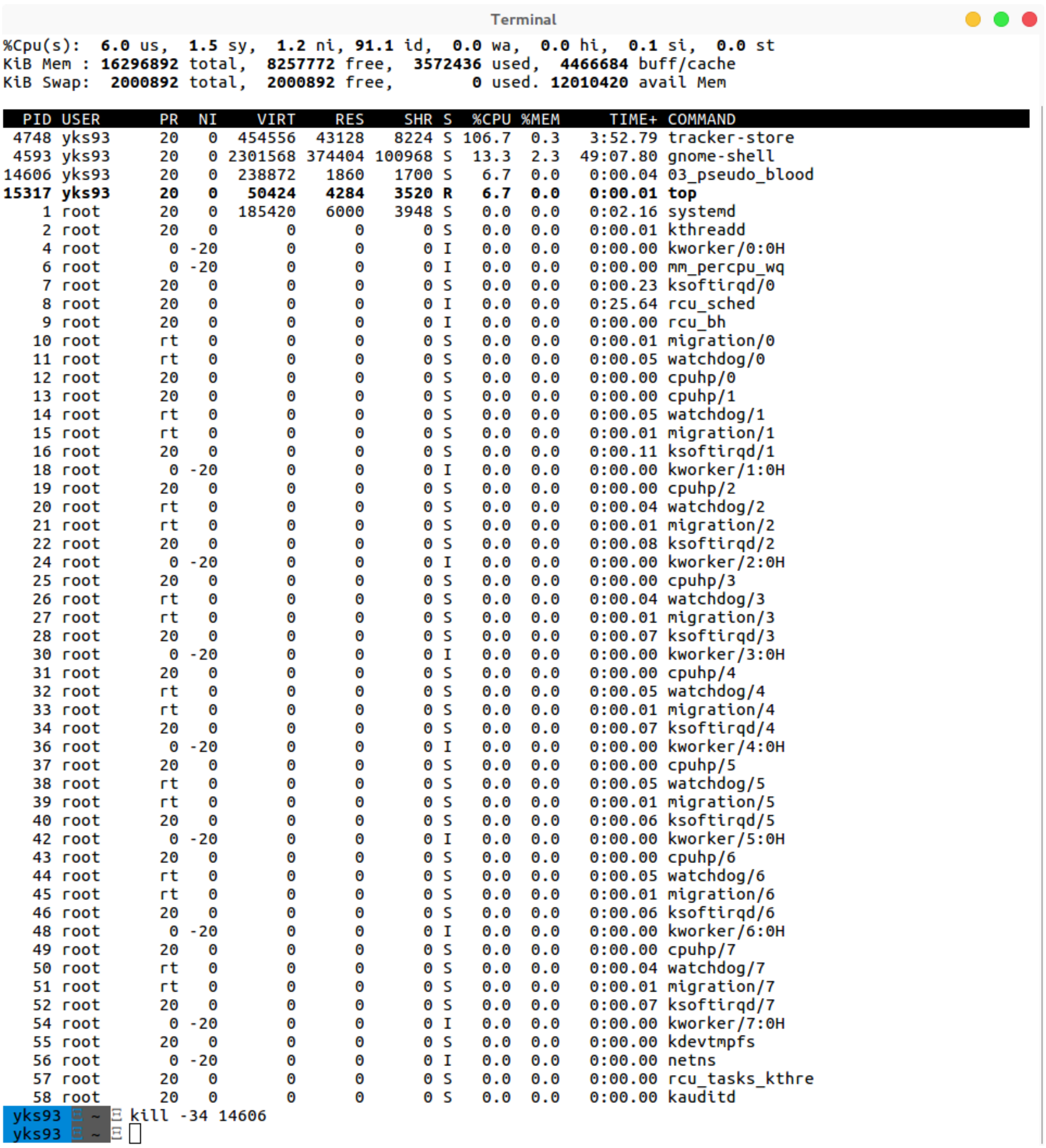
Starting pace of the program



Terminal

```
yks93  ~/Documents/assignments/Operating_System/03_pseudo_blood_pressure_detector/build  make
Scanning dependencies of target 03_pseudo_blood_pressure_detector
[  5%] Building CXX object CMakeFiles/03_pseudo_blood_pressure_detector.dir/src/main_executor.cpp.o
[ 11%] Building CXX object CMakeFiles/03_pseudo_blood_pressure_detector.dir/src/signal_handlers.cpp.o
[ 16%] Building CXX object CMakeFiles/03_pseudo_blood_pressure_detector.dir/src/src_pbpd/bp_processing_th
read.cpp.o
[ 22%] Building CXX object CMakeFiles/03_pseudo_blood_pressure_detector.dir/src/src_pbpd/data_acquisition
_thread.cpp.o
[ 27%] Building CXX object CMakeFiles/03_pseudo_blood_pressure_detector.dir/src/src_ptttm/create_pthread.
cpp.o
[ 33%] Linking CXX executable 03_pseudo_blood_pressure_detector
[100%] Built target 03_pseudo_blood_pressure_detector
yks93  ~/Documents/assignments/Operating_System/03_pseudo_blood_pressure_detector/build  ./03_pseudo_b
lood_pressure_detector
main begins...
initializing ptttm variables
initializing pbpd variables
create_new_thread( ) data_acquisition thread
create_new_thread( ) blood_pressure_processing thread

systolic bp = 63
diastolic bp = 124

systolic bp = 58
diastolic bp = 130

systolic bp = 54
diastolic bp = 125

systolic bp = 54
diastolic bp = 144

systolic bp = 53
diastolic bp = 134

systolic bp = 65
diastolic bp = 129

systolic bp = 63
diastolic bp = 132

systolic bp = 60
diastolic bp = 138

systolic bp = 61
diastolic bp = 121

systolic bp = 63
diastolic bp = 131

systolic bp = 68
diastolic bp = 129

systolic bp = 63
diastolic bp = 133

systolic bp = 62
diastolic bp = 145

systolic bp = 63
```

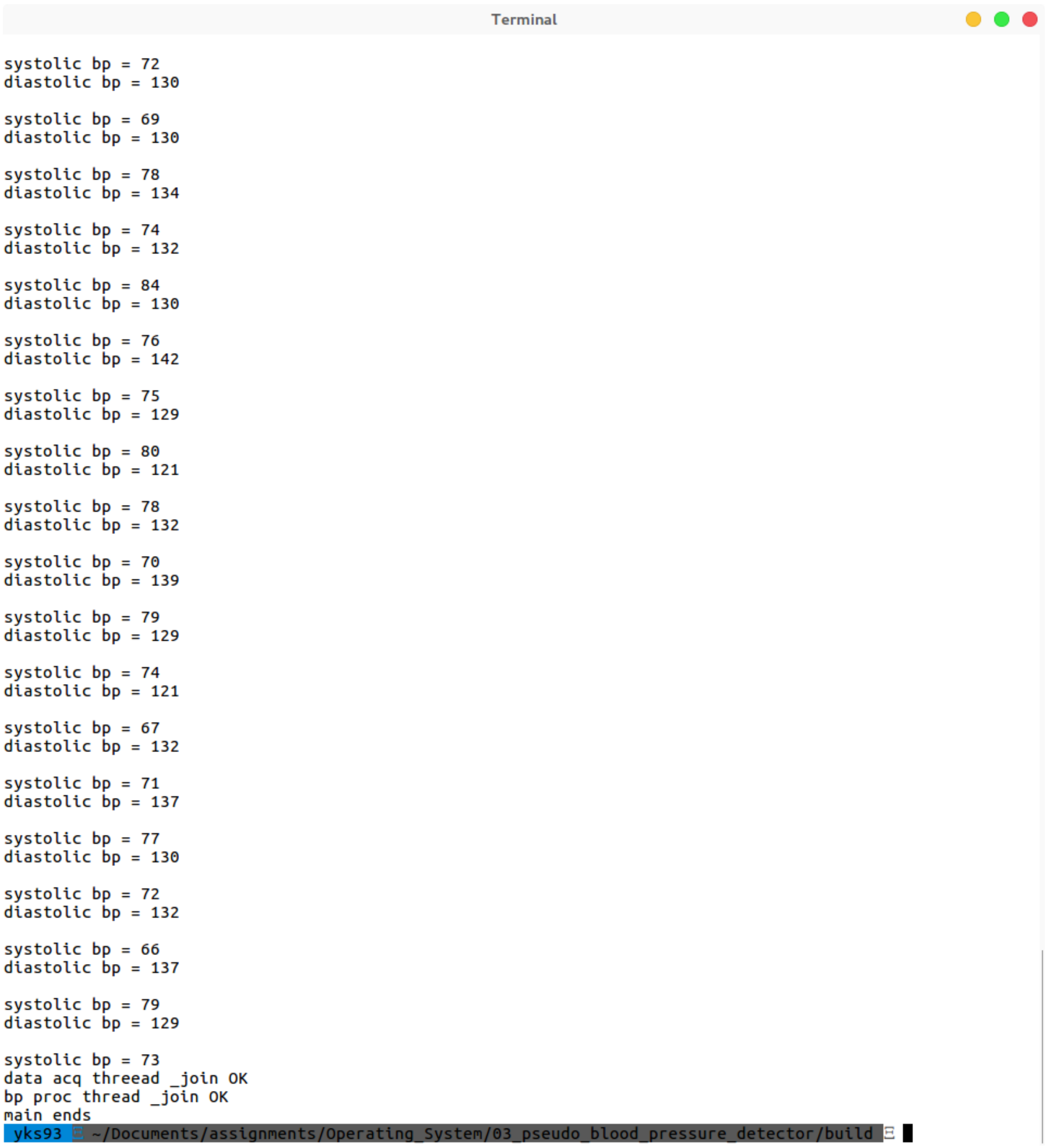
After sending signal (between 34(= SIGRTMIN) to 64(= SIGRTMAX)) **to the program**



Terminal

```
%Cpu(s):  6.0 us,  1.5 sy,  1.2 ni, 91.1 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem : 16296892 total,  8257772 free,  3572436 used,  4466684 buff/cache
KiB Swap:  2000892 total,  2000892 free,        0 used. 12010420 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4748 yks93     20   0  454556  43128   8224 S 106.7  0.3   3:52.79 tracker-store
 4593 yks93     20   0 2301568 374404 100968 S  13.3  2.3  49:07.80 gnome-shell
14606 yks93     20   0  238872   1860   1700 S   6.7  0.0   0:00.04 03_pseudo_blood
15317 yks93     20   0   50424   4284   3520 R   6.7  0.0   0:00.01 top
    1 root      20   0  185420   6000   3948 S   0.0  0.0   0:02.16 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.01 kthreadd
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_wq
    7 root      20   0       0      0      0 S   0.0  0.0   0:00.23 ksoftirqd/0
    8 root      20   0       0      0      0 I   0.0  0.0   0:25.64 rcu_sched
    9 root      20   0       0      0      0 I   0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:00.05 watchdog/0
   12 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/0
   13 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/1
   14 root      rt   0       0      0      0 S   0.0  0.0   0:00.05 watchdog/1
   15 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/1
   16 root      20   0       0      0      0 S   0.0  0.0   0:00.11 ksoftirqd/1
   18 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/1:0H
   19 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/2
   20 root      rt   0       0      0      0 S   0.0  0.0   0:00.04 watchdog/2
   21 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/2
   22 root      20   0       0      0      0 S   0.0  0.0   0:00.08 ksoftirqd/2
   24 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/2:0H
   25 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/3
   26 root      rt   0       0      0      0 S   0.0  0.0   0:00.04 watchdog/3
   27 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/3
   28 root      20   0       0      0      0 S   0.0  0.0   0:00.07 ksoftirqd/3
   30 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/3:0H
   31 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/4
   32 root      rt   0       0      0      0 S   0.0  0.0   0:00.05 watchdog/4
   33 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/4
   34 root      20   0       0      0      0 S   0.0  0.0   0:00.07 ksoftirqd/4
   36 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/4:0H
   37 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/5
   38 root      rt   0       0      0      0 S   0.0  0.0   0:00.05 watchdog/5
   39 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/5
   40 root      20   0       0      0      0 S   0.0  0.0   0:00.06 ksoftirqd/5
   42 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/5:0H
   43 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/6
   44 root      rt   0       0      0      0 S   0.0  0.0   0:00.05 watchdog/6
   45 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/6
   46 root      20   0       0      0      0 S   0.0  0.0   0:00.06 ksoftirqd/6
   48 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/6:0H
   49 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/7
   50 root      rt   0       0      0      0 S   0.0  0.0   0:00.04 watchdog/7
   51 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/7
   52 root      20   0       0      0      0 S   0.0  0.0   0:00.07 ksoftirqd/7
   54 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/7:0H
   55 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kdevtmpfs
   56 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 netns
   57 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_tasks_kthre
   58 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kauditd
yks93  ~  kill -34 14606
yks93  ~ 
```

Ending pace of the program



Terminal

```
systolic bp = 72
diastolic bp = 130

systolic bp = 69
diastolic bp = 130

systolic bp = 78
diastolic bp = 134

systolic bp = 74
diastolic bp = 132

systolic bp = 84
diastolic bp = 130

systolic bp = 76
diastolic bp = 142

systolic bp = 75
diastolic bp = 129

systolic bp = 80
diastolic bp = 121

systolic bp = 78
diastolic bp = 132

systolic bp = 70
diastolic bp = 139

systolic bp = 79
diastolic bp = 129

systolic bp = 74
diastolic bp = 121

systolic bp = 67
diastolic bp = 132

systolic bp = 71
diastolic bp = 137

systolic bp = 77
diastolic bp = 130

systolic bp = 72
diastolic bp = 132

systolic bp = 66
diastolic bp = 137

systolic bp = 79
diastolic bp = 129

systolic bp = 73
data acq threead _join OK
bp proc thread _join OK
main ends
 yks93  ~/Documents/assignments/Operating_System/03_pseudo_blood_pressure_detector/build 
```