

タイトル(title):

C言語の疑似スレッド

サブタイトル(subtitle):

C言語の偽物のスレッド

The thread at the imitation of the C language.

似ているが独創的な別物

The resembling but original singleton.

C言語によるスレッドという概念の模倣

Copying a concept, the thread, by the C language.

=====

/* Imitative4.h */

#include <stdio.h> /* printf() */

#include <time.h> /* time(), clock() */

#include <stdlib.h> /* calloc(), free(), rand() */

#define CLEAR system("cls")

#define OK 1

#define NG 0

#define USE_THREAD

=====

/* Imitative4_Timer.h */

```
/* 時間に使用する定数の宣言 */
```

```
#define WOVICLOCKSIZE 1000000.0
```

```
#define WOVI_PER_1S 25000000.0
```

```
/* 疑似スレッド定義 */
```

```
#ifdef USE_THREAD
```

```
/* 疑似スレッドに使用する定数の宣言 */
```

```
#define INITCLOCKNO 1000001
```

```
#define STOPCLOCKNO 1000002
```

```
/* 構造体宣言 */
```

```
typedef struct tag_Thread
```

```
{
```

```
    /* 疑似スレッドID */
```

```
    int ID;
```

```
    /* 指定開始時 */
```

```
    double preClock;
```

```
    /* woviClockがpreClockからsetClock秒増えたらRunを呼ぶ */
```

```
    double setClock;
```

```
    /* Runが呼ばれた回数を調べるために使用(countUpNextRunが呼ばれた回数) */
```

```
    long count;
```

```
    /* List機能 */
```

```
    struct tag_Thread *previous;
```

```
    struct tag_Thread *next;
```

```
}Thread;
```

```
/* 疑似メソッドとwovi用関数のプロトタイプ宣言 */
```

```

/* 宣言の順番は以下の通り */
#endif

double getClock(void);

#ifdef USE_THREAD

void nextRun(Thread *This, unsigned int ms);

void countUpNextRun(Thread *This, unsigned int ms);

void Run(Thread *This); /* Imitative4_main.cで内容を定義します */
void Init(Thread *This); /* Imitative4_main.cで内容を定義します */
void Destroy(Thread *This); /* Imitative4_main.cで内容を定義します */

Thread *new_Thread(unsigned short int id);

void delete_(Thread *This);

void Start(Thread *This);

void Stop(Thread *This);

int Thread_checkAllDelete(void);

Thread *Thread_getThread(unsigned short int id);

Thread *Thread_Start(unsigned short int id);

void woviRun(void);

void wovilnit(void);

#endif

/* タイマ関数 */

void wovi(void);

/* タイマ初期化関数 */

void initWOVI(void);

/* ミリ秒待ち関数 */

void setSleep(unsigned int ms);

/* 現在日時表示 */

void PrintCurrentTime(void);

```

```
=====

/* Imitative4_Timer.c */

#include "Imitative4.h"
#include "Imitative4_Timer.h"

/* 時間を表す外部変数宣言 */
double woviClock;

/* 疑似スレッド定義 */
#ifdef USE_THREAD

/* wovi用疑似インスタンス宣言 */
Thread woviThreadFirst;
Thread woviThreadLast;

#endif

/* 時刻取得 */
double getClock(void)
{
    return woviClock;
}

#ifdef USE_THREAD
```

```
/* スレッドのvoid Sleep(int ms)の代用 */
```

```
void nextRun(Thread *This, unsigned int ms)
```

```
{  
    This->preClock = woviClock;  
    This->setClock = (((double) ms) / 1000);  
    return;  
}
```

```
/* スレッドのvoid Sleep(int ms)の代用 */
```

```
void countUpNextRun(Thread *This, unsigned int ms)
```

```
{  
    nextRun(This, ms);  
    This->count++;  
}
```

```
/* スレッドのコンストラクタの代用 */
```

```
Thread *new_Thread(unsigned short int id)
```

```
{  
    Thread *List;  
    Thread *new_List;  
    List = &woviThreadFirst;  
    while(List->next->next != NULL)  
    {  
        List = List->next;  
    }  
    new_List = (Thread *)calloc(1, sizeof(Thread));  
    if(new_List == NULL)  
    {
```

```
    printf("¥ncalloc failed");
    return NULL;
}
new_List->previous = List;
new_List->next = List->next;
new_List->next->previous = new_List;
List->next = new_List;
new_List->preClock = INITCLOCKNO;
new_List->setClock = 0;
new_List->ID = id;
new_List->count = 0;
/* スレッドのvoid init(void)の代用 */
Init(new_List);
return new_List;
}
```

```
/* スレッドのデストラクタの代用 */
```

```
void delete_(Thread *This)
{
    Destroy(This);
    This->previous->next = This->next;
    This->next->previous = This->previous;
    free(This);
    return;
}
```

```
/* スレッドのvoid start(void)の代用 */
```

```
void Start(Thread *This)
{
```

```
woviClock = getClock();
This->preClock = woviClock;
return;
}
```

```
/* スレッドのvoid stop(void)の代用 */
```

```
void Stop(Thread *This)
{
    This->preClock = STOPCLOCKNO;
    return;
}
```

```
int Thread_checkAllDelete(void)
{
    if(woviThreadFirst.next->next == NULL)
    {
        return OK;
    }
    else
    {
        return NG;
    }
}
```

```
Thread *Thread_getThread(unsigned short int id)
{
    Thread *th;

    if(woviThreadFirst.next->next == NULL)
```

```
{
    return NULL;
}
else
{
    th = woviThreadFirst.next;
    do
    {
        if(th->ID == id)
        {
            return th;
        }
        else
        {
            th = th->next;
        }
    }while(th->next != NULL);
}
return NULL;
}
```

Thread *Thread_Start(unsigned short int id)

```
{
    Thread *th;

    th = Thread_getThread(id);
    if(th == NULL)
    {
        th = new_Thread(id);
    }
}
```



```

        Start(th);
    }
    else if(th->preClock == STOPCLOCKNO)
    {
        Start(th);
    }
    return th;
}

```

/ Runを呼ぶタイミング */*

```

void woviRun(void)
{
    double woviClockCompare;
    Thread *List;
    Thread *next_List;
    List = &woviThreadFirst;
    List = List->next;
    while(List->next != NULL)
    {
        next_List = List->next;
        if((List->preClock != INITCLOCKNO) && (List->preClock != STOPCLOCKNO))
        {
            woviClockCompare = List->preClock + List->setClock;
            if(woviClock < List->preClock)
            {
                woviClockCompare -= WOVICLOCKSIZE;
            }
            if(woviClock >= woviClockCompare)

```

```

        {
            List->preClock = woviClock;
            /* スレッドのvoid run(void)の代用 */
            Run(List);
        }
    }
    List = next_List;
}
return;
}

```

/* 指定開始時OFF */

void wovlnit(void)

```

{
    woviThreadFirst.previous = NULL;
    woviThreadFirst.next = &woviThreadLast;
    woviThreadLast.previous = &woviThreadFirst;
    woviThreadLast.next = NULL;
    return;
}

```

#endif

/* タイマ関数 */

void wovi(void)

```

{
    woviClock += 1.0 / WOVI_PER_1S;
    if(woviClock >= WOVICLOCKSIZE)
    {

```

```

        printf("¥nOVERWOVI");
        woviClock -= WOVICLOCKSIZE;
    }
#ifdef USE_THREAD
    woviRun(); /* スレッドのためのRunを呼ぶタイミング */
#endif
    return;
}

```

/* タイマ初期化関数 */

```
void initWOVI(void)
```

```

{
    woviClock = 0.0;
#ifdef USE_THREAD
    wovilnit(); /* スレッドのための指定開始時OFF */
#endif
    return;
}

```

/* ミリ秒待ち関数 */

```
void setSleep(unsigned int ms)
```

```

{
    double start;
    double set;
    start = clock() / CLOCKS_PER_SEC;
    set = ((double) ms) / 1000;
    while(woviClock < start + set)
    {
        woviClock = clock() / CLOCKS_PER_SEC;
    }
}

```

```

    }

    return;
}

/* 現在日時表示 */
void PrintCurrentTime(void)
{
    time_t timer;

    struct tm *t_st;

    /* 現在時刻の取得 */
    time(&timer);

    /* 現在時刻を構造体に変換 */
    t_st = localtime(&timer);

    printf("%d",t_st->tm_year+1900);
    if(t_st->tm_mon+1 < 10)
    {
        printf("0%d",t_st->tm_mon+1);
    }
    else
    {
        printf("%d",t_st->tm_mon+1);
    }
    if(t_st->tm_mday < 10)
    {
        printf("0%d",t_st->tm_mday);
    }
}

```

```
else
{
    printf("%d",t_st->tm_mday);
}
printf(" ");
if(t_st->tm_hour < 10)
{
    printf("0%d",t_st->tm_hour);
}
else
{
    printf("%d",t_st->tm_hour);
}
if(t_st->tm_min < 10)
{
    printf("0%d",t_st->tm_min);
}
else
{
    printf("%d",t_st->tm_min);
}
if(t_st->tm_sec < 10)
{
    printf("0%d",t_st->tm_sec);
}
else
{
    printf("%d",t_st->tm_sec);
}
```

```
}

return;

}
```

```
/* Imitative4_main.h */
```

```
#ifndef Imitative4_Timer_h
#define Imitative4_Timer_h
#include "Imitative4_Timer.h"
#endif
```

```
#define THREADSIZE 4
```

```
/* Imitative4_main.c */
```

```
#include "Imitative4.h"
#include "Imitative4_main.h"
```

```
void main(void)
{
    unsigned short int i;
#ifdef USE_THREAD
```

```

/* 疑似スレッドの疑似インスタンス宣言 */
Thread *th[THREADSIZE];

#endif

printf("\nHello BCC");

/* タイマー初期化 */
initWOVI();

#ifdef USE_THREAD

printf("\nImitative");

/* 疑似スレッドの疑似インスタンス初期化 */
for(i = 0; i < THREADSIZE; i++)
{
    th[i] = new_Thread(i + 1);
}

#endif

/* 10秒待機 */
setSleep(10000);

#ifdef USE_THREAD

/* 疑似スレッド開始 */
printf("\nStart");

for(i = 0; i < THREADSIZE; i++)
{
    Start(th[i]);
}

for(;;)
{
    /* タイマー呼び出し */
    wovi();

    if(Thread_checkAllDelete() == OK)
    {

```

```

        break;
    }
}

#endif

    printf("¥nEND");
    /* 10秒待機 */
    setSleep(10000);
    return;
}

#ifdef USE_THREAD

/*
 * 疑似スレッドの疑似メソッド関数
 */
/* スレッドのpublic void run()の代用 */
void Run(Thread *This)
{
    Thread *th1;

    if(This->ID == 1)
    {
        if(This->count == 1)
        {
            printf("¥n<1>1回目 Time = ");
            PrintCurrentTime();
            countUpNextRun(This, (((rand() % 9) + 10) * 100 * 1));
        }
        else if(This->count == 2)
        {

```



```

    printf("¥n<1>2回目");
    printf(" <1>Stop Time = ");
    PrintCurrentTime();
    Stop(This);
}
else if(This->count == 3)
{
    printf("¥n<1>3回目 Time = ");
    PrintCurrentTime();
    countUpNextRun(This, (((rand() % 9) + 10) * 100 * 1));
}
else if(This->count == 4)
{
    printf("¥n<1>Stop Time = ");
    PrintCurrentTime();
    Stop(This);
}
}
else if(This->ID == 4)
{
    if(This->count == 1)
    {
        printf("¥n<4>1回目 Time = ");
        PrintCurrentTime();
        countUpNextRun(This, (((rand() % 9) + 10) * 100 * 4));
    }
    else if(This->count == 2)
    {

```

```

printf("¥n<4>2回目");
countUpNextRun(This, (((rand() % 9) + 10) * 100 * 4));

printf(" <1>Start Time = ");
PrintCurrentTime();
th1 = Thread_Start(1);
countUpNextRun(th1, (((rand() % 9) + 10) * 100 * 1));
}
else if(This->count == 3)
{
printf("¥n<4>3回目 Time = ");
PrintCurrentTime();
countUpNextRun(This, (((rand() % 9) + 10) * 100 * 4));
}
else if(This->count == 4)
{
th1 = Thread_getThread(1);
if(th1 != NULL)
{
delete_(th1);
}

printf(" <4>Stop");
Stop(This);
delete_(This);
}
}
else
{

```

```

if(This->count <= 3)
{
    printf("¥n<%d>%d回目 Time = ", This->ID, This->count);
    PrintCurrentTime();
    countUpNextRun(This, (((rand() % 9) + 10) * 100 * This->ID));
}
else
{
    printf("¥n<%d>Stop", This->ID);
    Stop(This);
    delete_(This);
}
}
return;
}

```

/* スレッドのコンストラクタのpublic void init()の代用 */

```
void Init(Thread *This)
```

```

{
    printf("¥n<%d>Init", This->ID);
    countUpNextRun(This, (((rand() % 9) + 10) * 100 * This->ID));

    return;
}

```

```
#endif
```

/* スレッドのデストラクタの代用 */

```
void Destroy(Thread *This)
```

```
{
```

```
if(This->ID == 1)
{
    printf("¥n<1>Destroy");
}
else
{
    printf(" <%d>Destroy Time = ", (This->ID));
    PrintCurrentTime();
}

return;
}
```

=====

Imitative4_makefile.mak

CC = bcc32

Imitative4_main.exe : Imitative4_Timer.obj Imitative4_main.obj

\$(CC) Imitative4_main.obj Imitative4_Timer.obj

Imitative4_Timer.obj : Imitative4_Timer.c Imitative4_Timer.h Imitative4.h

\$(CC) -c Imitative4_Timer.c

Imitative4_main.obj : Imitative4_main.c Imitative4_main.h Imitative4_Timer.h Imitative4.h

\$(CC) -c Imitative4_main.c

clean:

del *.obj

del *.tds

=====
@rem Imitative4_build.bat

C:

set path=C:\borland\bcc55\Bin;%path%

D:

cd D:\Electronics\C_Imitative4\Imitative4_Work

del Imitative4_error.txt

make -f Imitative4_makefile.mak >> Imitative4_error.txt

make -f Imitative4_makefile.mak clean >> Imitative4_error.txt

Imitative4_error.txt

exit

=====
MAKE Version 5.2 Copyright (c) 1987, 2000 Borland

 bcc32 -c Imitative4_Timer.c

Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland

Imitative4_Timer.c:

 bcc32 -c Imitative4_main.c

Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland

Imitative4_main.c:

 bcc32 Imitative4_main.obj Imitative4_Timer.obj

Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland

Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland

MAKE Version 5.2 Copyright (c) 1987, 2000 Borland

 del *.obj

```
del *.tds
```

=====

C言語のプロジェクト Imitative4 について、

Imitative4_main.c の関数 main を見てください。

スレッドを使用しています。

Thread *th[THREADSIZE]; でオブジェクト宣言しています。

th[i] = new_Thread(i + 1); で初期値設定しています。

この2行は Java で次と同じ意味です。

```
Thread th[] = new Thread[THREADSIZE];
```

```
th[i] = new Thread(i + 1);
```

Start(th[i]); でスレッドを開始しています。

この1行は Java で次と同じ意味です。

```
th[i].start();
```

```
void Run(Thread *This)
```

```
{
```

```
    ...
```

```
}
```

```
void Init(Thread *This)
```

```
{
```

```
    ...
```

```
}
```

はそれぞれ Java で次と同じ意味です。

```
public void run()
```

```
{
```

```
    ...
```

```
}
```

```
public void init()
```

```
{
```

```
    ...
```

```
}
```

`delete_(This);` でオブジェクトを消去しています。

この1行は C++ で次と同じ意味です。

```
delete this;
```

これらのスレッドに関する仕様は `Imitative4_Timer.c` に記述しました。

`Imitative4_makefile.mak` `Imitative4_build.bat` は複数のファイルを1個のプロジェクトとしてコンパイルするためのファイルです。

=====

著作者:

しのみや ひでみね

篠宮 英峰

```
Hello BCC
Imitative
<1>Init
<2>Init
<3>Init
<4>Init
Start
<1>1回目 Time = 20140823 011917
<2>1回目 Time = 20140823 011919
<1>2回目 <1>Stop Time = 20140823 011920
<3>1回目 Time = 20140823 011920
<4>1回目 Time = 20140823 011922
<2>2回目 Time = 20140823 011922
<2>3回目 Time = 20140823 011927
<3>2回目 Time = 20140823 011928
<2>Stop <2>Destroy Time = 20140823 011931
<4>2回目 <1>Start Time = 20140823 011933
<1>3回目 Time = 20140823 011935
<3>3回目 Time = 20140823 011936
<1>Stop Time = 20140823 011937
<3>Stop <3>Destroy Time = 20140823 011941
<4>3回目 Time = 20140823 011943
<1>Destroy <4>Stop <4>Destroy Time = 20140823 011948
END
```