

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.

=====

/* Imitative.h */

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

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

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

#define CLEAR system("cls")

#define OK 1

#define NG 0

#define USE_THREAD

=====

/* Imitative_Timer.h */

```
/* 時間に使用する定数の宣言 */
```

```
#define WOVICLOCKSIZE 1000000.0
```

```
#define UNSIGNEDINTMAX 65535
```

```
/* 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); /* Imitative_main.cで内容を定義します */
void Init(Thread *This); /* Imitative_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);

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);

=====

/* Imitative_Timer.c */
```

```
#include "Imitative.h"
```

```
#include "Imitative_Timer.h"
```

```
/* 時間を表す外部変数宣言 */
```

```
double woviClock;
```

```
double woviClockCompare;
```

```
/* 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)
```

```
{
```

```
    double d;
```

```
    long l;
```

```
    d = clock() / CLOCKS_PER_SEC;
```

```
    d *= 1000;
```

```
    l = (long) d;
```

```
    l %= (((long) WOVICLOCKSIZE) - UNSIGNEDINTMAX) * 1000;
```

```
    d = (double) l;
```

```
    d /= 1000.0;
```

```
    return d;
```

```
}
```

```
#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)
{
    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;  
    }  
}
```

```
/* Runを呼ぶタイミング */
```

```
void woviRun(void)
```

```

{
    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))
        {
            woviClock = getClock();
            woviClockCompare = list->preClock + list->setClock;
            if(woviClock < list->preClock)
            {
                woviClockCompare -= (WOVICLOCKSIZE - UNSIGNEDINTMAX);
            }
            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 wovlinit(void)
{
    woviThreadFirst.previous = NULL;
    woviThreadFirst.next = &woviThreadLast;
    woviThreadLast.previous = &woviThreadFirst;
    woviThreadLast.next = NULL;
    return;
}

#endif

/* タイマ関数 */
void wovi(void)
{
    woviClock = getClock();
#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 = -WOVICLOCKSIZE;
#ifdef USE_THREAD
    wovlinit(); /* 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;
}
```

=====

```
/* Imitative_main.h */
```

```
#ifndef Imitative_Timer_h
#define Imitative_Timer_h
#include "Imitative_Timer.h"
#endif
```

```
#define THREADSIZE 4
```

```
=====

/* Imitative_main.c */

#include "Imitative.h"
#include "Imitative_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("%nlmitative");
    /* 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)
{

```

```

int i;

int j;

for(i = 0; i < THREADSIZE; i++)
{
    if(This->ID == (i + 1))
    {
        for(j = 1; j <= 3; j++)
        {
            if(This->count == j)
            {
                printf("¥n<%d>%d回目", (i + 1), j);
                countUpNextRun(This, (((rand() % 9) + 10) * 100 * (i + 1)));
                return;
            }
        }
        printf("¥n<%d>StopDestroy", (i + 1));
        Stop(This);
        delete_(This);
        return;
    }
}

```

/* スレッドのpublic void init()の代用 */

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

```

{
    int i;

    for(i = 0; i < THREADSIZE; i++)

```

```
{
    if(This->ID == (i + 1))
    {
        printf("¥n<%d>Init", (i + 1));
        countUpNextRun(This, (((rand() % 9) + 10) * 100 * (i + 1)));
        return;
    }
}
#endif
```

=====
Imitative_makefile.mak

CC = bcc32

Imitative_main.exe : Imitative_Timer.obj Imitative_main.obj

\$(CC) Imitative_main.obj Imitative_Timer.obj

Imitative_Timer.obj : Imitative_Timer.c Imitative_Timer.h Imitative.h

\$(CC) -c Imitative_Timer.c

Imitative_main.obj : Imitative_main.c Imitative_main.h Imitative_Timer.h Imitative.h

\$(CC) -c Imitative_main.c

clean:

del *.obj

del *.tds

=====

@rem Imitative_build.bat

C:

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

D:

cd D:\Electronics\C_Imitative\Imitative_Work

del Imitative_error.txt

make -f Imitative_makefile.mak >> Imitative_error.txt

make -f Imitative_makefile.mak clean >> Imitative_error.txt

Imitative_error.txt

exit

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

bcc32 -c Imitative_Timer.c

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

Imitative_Timer.c:

bcc32 -c Imitative_main.c

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

Imitative_main.c:

bcc32 Imitative_main.obj Imitative_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 Imitative について、

Imitative_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 `Imitative_Timer.c` に記述しました。

`Imitative_makefile.mak` `Imitative_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
<2>1回目
<1>2回目
<3>1回目
<2>2回目
<4>1回目
<1>3回目
<1>StopDestroy
<2>3回目
<3>2回目
<4>2回目
<2>StopDestroy
<3>3回目
<3>StopDestroy
<4>3回目
<4>StopDestroy
END
```