

# 発明の巻

スレッドとは、コンピュータープログラミング上の、並列処理の機能です。

スレッドをサポートしているプログラミング言語と、スレッドをサポートしていないプログラミング言語があります。

高度な機能のスレッドをわざとサポートしない言語があるのは、パソコン以外の環境で実行するために、低レベルマイコン対応のプログラミング言語でいるから(例えば C 言語)です。

代わりにマイコンにはインターバルタイマーがあります。

ただし、C 言語からインターバルタイマーを使用するには、そのマイコンのマニュアルだけではなく、そのマイコンの C 言語のマニュアルも必要です。

H 社さんの社員さんは、H 社さんの製品のマイコンのインターバルタイマーを呼び出せるでしょう。

H 社さんの社員さん以外の方は、情報を揃えるのが困難なので、H 社さんの製品のマイコンのインターバルタイマーを呼び出すのは困難でしょう。

M 社さんの社員さんは、M 社さんの製品のマイコンのインターバルタイマーを呼び出せるでしょう。

M 社さんの社員さん以外の方は、情報を揃えるのが困難なので、M 社さんの製品のマイコンのインターバルタイマーを呼び出すのは困難でしょう。

したがって、大手の社員さんは、その大手の会社の製品のマイコンのインターバルタイマーを呼び出せるでしょう。

一般の方は、情報を揃えるのが困難なので、マイコンのインターバルタイマーを呼び出すのは困難でしょう。

今回作成しました、並行処理タイプのスレッド(疑似スレッドと命名)は、インターバルタイマーの使い方が習得できない一般の方には、朗報かも知れません。

発明アルゴリズムに疑似スレッドと名付けました。本物に偽物と名付けました。特許申請はしないで、無料公開をいたしました。発明という手段で人類としてのノルマを達成し、また、生きた証が発生いたしました。篠宮氏を逆賊に祭り上げようとする人種がいます。篠宮氏は辞退いたします。秋篠宮家様は本物の皇族ですが、篠宮氏は偽物です。

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.

```
/* C.h */

#include <stdio.h> /* printf() */
#include <time.h> /* time(), clock() */
#include <stdlib.h> /* calloc(), free(), rand() */
#include <conio.h> /* kbhit(), getche() */
#define CLEAR system("cls")
#define OK 1
#define NG 0
#define USE_THREAD
#define SLEEP_PER_SEC 100000000.0
```

```
/* Timer.h */
```

```
/* 時間に使用する定数の宣言 */
```

```
#define WOVICLOCKSIZE 1000000.0
```

```
/* 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); /* main.cで内容を定義します */  
  
void Init(Thread *This); /* main.cで内容を定義します */  
  
void Destroy(Thread *This); /* 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 Thread_Toggle(unsigned short int id);  
  
void woviRun(void);  
  
void wovilnit(void);  
  
/* タイマ関数 */  
  
void wovi(double threadPerSec);  
  
/* タイマ初期化関数 */  
  
void initWOVI(void);  
  
  
#endif  
  
  
/* ミリ秒待ち関数 */  
  
void SleepMSec(double sleepPerSec, unsigned int ms);
```

```
/* ミリ秒待ち関数 */
```

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

```
/* 現在日時表示 */
```

```
void PrintCurrentTime(void);
```



```
/* Timer.c */
```

```
#include "C.h"
```

```
#include "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;
}
```

```
void Thread_Toggle(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);
    }
    else
    {
        delete_(th);
    }
    return;
}
```

```
/* 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 wovilnit(void)
{
    woviThreadFirst.previous = NULL;
    woviThreadFirst.next = &woviThreadLast;
    woviThreadLast.previous = &woviThreadFirst;
    woviThreadLast.next = NULL;
    return;
}

/* タイマ関数 */
void wovi(double threadPerSec)
{
    woviClock += 1.0 / threadPerSec;
    if(woviClock >= WOVICLOCKSIZE)
    {
        printf("¥nOVERWOVI");
        woviClock -= WOVICLOCKSIZE;
    }
    woviRun(); /* スレッドのためのRunを呼ぶタイミング */
    return;
}

/* タイマ初期化関数 */
void initWOVI(void)
{
    woviClock = 0.0;
    wovilnit(); /* スレッドのための指定開始時OFF */
    return;
}

```



```
#endif
```

```
/* ミリ秒待ち関数 */
```

```
void SleepMSec(double sleepPerSec, unsigned int ms)
```

```
{  
    double cnt;  
    double set;  
    cnt = 0.0;  
    set = ((double) ms) / 1000;  
    while(cnt < set)  
    {  
        cnt += 1.0 / sleepPerSec;  
    }  
    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;
```

```
}
```

```
/* main.h */
```

```
#ifndef Timer_h  
#define Timer_h  
#include "Timer.h"  
#endif
```

```
#define THREADSIZE 4
```

```
/* main.c */

#include "C.h"

#include "main.h"

#ifdef USE_THREAD

void Thread1(void)

{

    unsigned short int i;

    /* 疑似スレッドの疑似インスタンス宣言 */

    Thread *th[THREADSIZE];

    printf("%nCountUp");

    /* 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);

    }

    /* 2秒待機 */

    SleepMSec(SLEEP_PER_SEC, 2000);

    /* 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(25000000.0);  
if(Thread_checkAllDelete() == OK)  
{  
    break;  
}  
}  
return;  
}
```

```
void Thread2(void)
```

```
{  
    char key;  
    Thread *th20;  
  
    th20 = new_Thread(20);  
    printf("¥n");  
    Start(th20);  
  
    for(;;)  
    {  
        /* タイマー呼び出し */  
        wovi(50000.0);  
        key = ' ';  
        if(kbhit())  
        {  
            key = getche();  
        }  
        if(key == '1')  
        {
```

```
    Thread_Toggle(21);
}
else if(key == '2')
{
    Thread_Toggle(22);
}
else if(key == '3')
{
    Thread_Toggle(23);
}
else if(key == '4')
{
    Thread_Toggle(24);
}
else if(key == '5')
{
    Thread_Toggle(25);
}
else if(key == '6')
{
    Thread_Toggle(26);
}
else if(key == '7')
{
    Thread_Toggle(27);
}
else if(key == '8')
{
    Thread_Toggle(28);
```



```
    }  
    else if(key == '9')  
    {  
        Thread_Toggle(29);  
    }  
    else if(key == '0')  
    {  
        Thread_Toggle(20);  
    }  
  
    if(Thread_checkAllDelete() == OK)  
    {  
        break;  
    }  
}  
return;
```

```
}  
#endif
```

```
void main(void)
```

```
{  
    printf("¥nHello BCC");  
#ifdef USE_THREAD  
    /* タイマー初期化 */  
    initWOVI();  
    /* 前半部分 */  
    Thread1();  
#endif  
    printf("¥nNEXT");
```

```

    /* 2秒待機 */
    SleepMSec(SLEEP_PER_SEC, 2000);
#ifdef USE_THREAD
    /* 後半部分 */
    Thread2();
#endif

    printf("¥nEND");

    /* 10秒待機 */
    SleepMSec(SLEEP_PER_SEC, 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->ID == 2) || (This->ID == 3))
{
    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);
    }
}
else
{
    printf("<%d>", This->ID);
    nextRun(This,2000);
}
return;
}

```

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

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

```

{
    if((This->ID == 1) || (This->ID == 2) || (This->ID == 3) || (This->ID == 4))
    {
        printf("%n<%d>Init", This->ID);
    }
}

```

```

        countUpNextRun(This, (((rand() % 9) + 10) * 100 * This->ID));
    }
    else
    {
        printf("<%d>Init", This->ID);
        nextRun(This,2000);
    }
    return;
}

```

/\* 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 if((This->ID == 2) || (This->ID == 3) || (This->ID == 4))
    {
        printf(" <%d>Destroy Time = ", (This->ID));
        PrintCurrentTime();
    }
    else
    {
        printf("<%d>Destroy", This->ID);
    }
    return;
}

```

#endif

```
# makefile.mak
CC = bcc32
main.exe : Timer.obj main.obj
    $(CC) main.obj Timer.obj
Timer.obj : Timer.c Timer.h C.h
    $(CC) -c Timer.c
Toggle_main.obj : main.c main.h Timer.h C.h
    $(CC) -c main.c
clean:
    del *.obj
    del *.tds
```



```
@rem build.bat
C:
set path=C:\borland\bcc55\Bin;%path%
D:
cd D:\Hidemine_Shinomiya\C_CountUp\CountUp_Work
del error.txt
make -f makefile.mak >> error.txt
make -f makefile.mak clean >> error.txt
error.txt
exit
```

```
MAKE Version 5.2 Copyright (c) 1987, 2000 Borland
    bcc32 -c Timer.c
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
Timer.c:
    bcc32 /c main.c
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
main.c:
    bcc32 main.obj 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 CountUp について、

main.c の関数 Thread1 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 `Timer.c` に記述しました。

`makefile.mak` `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。

## 使い方

NEXT以降の後半部分の、スレッド20が走り始めたら、

0以外の数字キーを押してみてください。

その数字に20を加えた番号のスレッドが、キーを押す度毎に、

起動・消去を繰り返します。

20を含めて、全部スレッドが消去されると、終了です。

著作者:

しのみや ひでみね

篠宮 英峰