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.

```
/* Toggle.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
```

```
/* Toggle_Timer.h */

/* 時間に使用する定数の宣言 */
#define WOVICLOCKSIZE 1000000.0
#define WOVI_PER_1S 25000.0

/* 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); /* Toggle_main.cで内容を定義します */
void Init(Thread *This); /* Toggle_main.cで内容を定義します */
void Destroy(Thread *This); /* Toggle_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 woviInit(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);
```

```
/* Toggle_Timer.c */

#include "Toggle.h"
#include "Toggle_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 woviInit(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
	woviInit(); /* 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;
}
```

```
/* Toggle_main.h */

#ifndef Toggle_Timer_h
#define Toggle_Timer_h
#include "Toggle_Timer.h"
#endif
```

```
/* Toggle_main.c */

#include "Toggle.h"
#include "Toggle_main.h"

#ifdef USE_THREAD
void Thread0(void)
{
	char key;
	Thread *th0;

	th0 = new_Thread(0);
	printf("¥n");
	Start(th0);

	for(;;)
	{
		/* タイマー呼び出し */
		wovi();
		key = ' ';
		if(kbhit())
		{
			key = getche();
		}
		if(key == '1')
		{
			Thread_Toggle(1);
		}
```

```
else if(key == '2')
{
    Thread_Toggle(2);
}
else if(key == '3')
{
    Thread_Toggle(3);
}
else if(key == '4')
{
    Thread_Toggle(4);
}
else if(key == '5')
{
    Thread_Toggle(5);
}
else if(key == '6')
{
    Thread_Toggle(6);
}
else if(key == '7')
{
    Thread_Toggle(7);
}
else if(key == '8')
{
    Thread_Toggle(8);
}
else if(key == '9')
```

```
            {
                Thread_Toggle(9);
            }
            else if(key == '0')
            {
                Thread_Toggle(0);
            }

            if(Thread_checkAllDelete() == OK)
            {
                break;
            }
        }
        return;
}
#endif

void main(void)
{
        printf("¥nHello BCC");
        /* タイマー初期化 */
        initWOVI();
#ifdef USE_THREAD
        printf("¥nToggle");
#endif
        /* 2秒待機 */
        setSleep(2000);
#ifdef USE_THREAD
        Thread0();
```

```
#endif
	printf("¥nEND");
	/* 2秒待機 */
	setSleep(2000);
	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)
{
	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)
{
	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)
```

```
{
	printf("<%d>Destroy", This->ID);
	return;
}

#endif
```

```
# Toggle_makefile.mak
CC = bcc32
Toggle_main.exe : Toggle_Timer.obj Toggle_main.obj
	$(CC) Toggle_main.obj Toggle_Timer.obj
Toggle_Timer.obj : Toggle_Timer.c Toggle_Timer.h Toggle.h
	$(CC) -c Toggle_Timer.c
Toggle_main.obj : Toggle_main.c Toggle_main.h Toggle_Timer.h Toggle.h
	$(CC) -c Toggle_main.c
clean:
	del *.obj
	del *.tds
```

```
@rem Toggle_build.bat
C:
set path=C:\borland\bcc55\Bin;%path%
D:
cd D:\Hidemine_Shinomiya\C_Toggle\Toggle_Work
del Toggle_error.txt
make -f Toggle_makefile.mak >> Toggle_error.txt
make -f Toggle_makefile.mak clean >> Toggle_error.txt
Toggle_error.txt
exit
```

```
MAKE Version 5.2  Copyright (c) 1987, 2000 Borland
        bcc32 -c Toggle_Timer.c
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
Toggle_Timer.c:
        bcc32 -c Toggle_main.c
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
Toggle_main.c:
        bcc32 Toggle_main.obj Toggle_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 Toggle について、

Toggle_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 *th0; でオブジェクト宣言しています。
th0 = new_Thread(0); 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 th0 = new Thread(0);

Start(th0); 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。
th0.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()
{
    ...
}
```

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

Toggle_Timer.c の 関数 Thread_Toggle を見てください。

delete_(th); 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 th;

Toggle_makefile.mak Toggle_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
スレッド0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。
0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