國 立 高 雄 師 範 大 學
高雄師大學報 49, 2020. 12, 頁 73-136

即時作業系統 tinyOS：完整功能版

蔡長達*

摘 要

在作業系統領域，很少發現有國人可以提供完整的原始程式來作為授課教材。本文參考 uC/OS-II 的理論、ARMv6-M 指令集與 context switch 的前人研究並延續已發表的 tinyOS1 精簡架構，成功開發出一個功能完備可運作施行的即時作業系統命名為 tinyOS。實驗是以 ARM Cortex-M0 微控制器來做為執行平台，展示了信號量、互斥信號量、訊息郵箱、旗標、中斷控制、佇列與記憶體管理之功能。

關鍵詞：即時作業系統、微控制器

投稿日期：2020/03/03；接受日期：2020/04/28

* 東南科技大學資訊科技系助理教授

74 Kaohsiung Normal University Journal
No.49, Dec. 2020, pp. 73-136

The TinyOS Real-Time Operating System Provided with Integrative Service

Chang-Da Tsai*

Abstract

In the field of operating system, it is exceedingly rare to find out a complete teaching material including source code proposed by Taiwanese. After investigating the uC/OS-II theory, ARMv6-M instruction set, and previous research in context switch, a real-time operating system provided with integrative service naming tinyOS inherited published simplified tinyOS1 was developed and successfully implemented on ARM Cortex-M0 processor based microcontroller to demonstrate the functionality of semaphore, mutual exclusion semaphore, mail box, flag, interrupt control, queue and memory manage, and so on.

Keywords: real-time operating system, microcontroller

Submitted: 2020/03/03；Accepted: 2020/04/28

* Assistant Professor, Department of Information Technology, Tungnan University

壹、前言

tinyOS1 是作者開發 tinyOS 過程中唯一公開發表過的子版本[1]，tinyOS1 最主要的優點是低耗記憶體及簡易設定程序容易啟動，而最主要的缺點是僅提供信號量一種事件服務，tinyOS1 原本的設計動機是要在 4K bytes RAM 的微控制器實現電腦平台高階語言多執行緒的同步程序控制，也就是要將電腦平台複雜的多執行緒程式移植至微控制器平台上執行。tinyOS1 雖然是以無與倫比極低耗記憶體(1K bytes RAM 即可執行)的優異特性問世，但是產業界的專家反應低記憶體消耗並非是即時作業系統最重要的考量，以產業界的角度而言，即時作業系統能否提供完備且親善的功能服務是遠比低記憶體消耗更重要。作者接納產業界的意見，持續開發完備且親善的功能服務，除了保留已發表的 tinyOS1 版本之外不再另名子版本，完備且親善版本的 tinyOS 其名就直接稱為 tinyOS。

tinyOS 的理論與功能服務是參考 uC/OS-II 即時作業系統[2]，指令集是使用 Cortex-M0 的 ARMv6-M [3]，內文交換(context switch)是參考 Adam Heinrich 的研究[4]，除了繼承 tinyOS1 極優異的記憶體消耗技術，tinyOS 亦具備諸多獨創技術來成就完備且親善的功能服務，包括信號量、互斥信號量、訊息郵箱、旗標、佇列與動態記憶體管理。

在 tinyOS 的設計中，除了互斥信號量，任務(task)可以同時等待多個相同事件或是不同事件，中斷服務程式(interrupt service routine)可以發布除了互斥信號量以外的任何事件，但是中斷服務程式不能等待任何事件，核心將不處理中斷服務程式的等待事件。

貳、聲明

本文作者即為 tinyOS 作者，應用 tinyOS 作任何開發皆不需付費，應用程式使用 tinyOS 也不必公開原始程式碼。

參、特色與理論

tinyOS 繼承 tinyOS1 諸多特色架構，以下僅說明相異之處，相同處請參考 tinyOS1 的說明[1]。

一、啟動

tinyOS 一貫的理念就是要非常容易啟動，不需要繁瑣的設定程序與使用規則，因此只須將 SIZE.h 檔案中應用程式會使用到的常數設定好就可以成功啟動，連 main()函數都可以使用模版來啟動核心，SIZE.h 檔案中有八個常數，只有二個是必須設定，分別是任務的數量與任務堆疊緩衝空間，另六個常數是事件與服務相關，依應用程式用途而選擇性設定(預設為 0)，啟動程序將會獨立說明於後，所以應用程式與作業系統核心之間的溝通介面非常簡潔。

二、狀態

同 tinyOS1。

三、優先權與就緒表

tinyOS 不限制任務數量，故就緒表相較於 tinyOS1 就會比較複雜一些，其餘與 tinyOS1 相同。

四、任務

同 tinyOS1。

五、任務函數指標陣列

同 tinyOS1。

六、等待函數

tinyOS 規定每一個使用者任務都必須至少呼叫一個等待函數以使自己進入等待狀態而讓其他任務有機會獲得 CPU 的執行權，tinyOS 所提供的等待函數有以下七個：pendSemOS()、pendMailOS()、pendFlagOS()、pendQOS()、pendMutexOS()、delayTimeOS()與 delayTickOS()。

七、事件與任務及中斷服務程式

tinyOS 的事件都是以整數作為名稱號碼，除了互斥信號量之外，所有任務都可以同時等待多個事件，事件號碼是在應用程式以整數陣列型式傳遞給核心，該陣列必須以-1 作為元素號碼的終止。只有任務自己才可以刪除自己所等待的事件，一個任務不能夠刪除其他任務的等待事件。中斷服務程式(ISR)無法等待事件，但可以發布除了互斥信號量之外的所有事件號碼。

八、動態記憶體管理

tinyOS 所提供的記憶體管理技術是非常獨特且超高效率，核心所管理的記憶體是以 8 bytes 作為基本單位稱為塊(bulk)，任務是以 byte 為單位向核心申請的記憶體，申請數量若除以 8 的餘數非為 0 或不足 8 bytes，核心都會以一個基本單位塊來計算。核心實際分配給任務的記憶體會比任務所申請的數量再多一個基本塊單位以作為緩衝空間，任務仍然可以合法安全使用該緩衝空間(8 bytes)但必須注意記憶體邊界，核心會再提供合法安全記憶體以外的剩餘記憶體資訊稱為不安全記憶體空間，只要未有其他任務再提出申請，該不安全記憶體空間仍然是可以正常使用，精準掌握任務程序而善用不安全空間可以大幅提高記憶體的使用效率。tinyOS 對於記憶體的歸還是非常和善可以盡量降低人為的錯誤，不會有記憶體歸還錯誤的問題，且核心提供記憶體使用狀況查詢的功能可方便應用程式規畫追蹤流程與除錯。如果記憶體的申請與歸還程序不同步將會耗盡記憶體池(memory pool)，tinyOS 允許清除記憶體池，但所有的任務就都必須再重新開始申請與歸還的程序。

九、佇列

tinyOS 的佇列是搭配動態記憶體管理技術，記憶體池的最前段是保留給佇列使用，任務若不再使用佇列亦可以歸還記憶體，歸還之後若要再使用佇列就必須先初始化記憶體池與佇列，此程序是核心啟動時會自動初始化，tinyOS 開放記憶體池與佇列的初始化程序給應用程式來調用，初始化記憶體池與佇列將會影響到其他有關的任務之運作，故必須小心為之。佇列的內容是記憶體位址而非具體資料數值，因此若是佇列內有相同的內容就表示有資料數值被覆蓋，就應該調整任務讀取或發布佇列的時間間隔。

十、void、void*與 void**

在 C 語言中 void 主要是用來宣告函數沒有輸入參數或是沒有輸出參數，例如：

```
void    xyz(void)
```

是表示 xyz 函數並不需要輸入參數也沒有輸出參數。

void*的本質是指標，指標即位址，絕大部分的應用場合指標都是代表記憶體位址，但對 CPU 而言，指標也可能是代表其他硬體裝置而非記憶體。void*是一種特殊指標，C 程式不能對 void*指標的內容資料作存取，必須將 void*指標強制轉換為確定的資料形態指標(例如 int*)之後才可以作存取，任何確定的資料形態指標也可以指定給 void*指標來作傳遞，故 void*指標比較像是容器的功用，void*指標可以接受任何資料形態指標的指定，也可以將 void*指標指定給任何資料形態的指標，例如將函數的輸入與輸出皆宣告為 void*如下

```
void*    xyz(void*    p)
```

則可以將任何資料形態的指標傳給 xyz 函數的輸入 p，而 xyz 函數的輸出則可以用任何資料形態的指標來接收。

void**可以理解為元素資料型態是 void*的陣列，在 C 語言中陣列名稱其本質就是記憶體位址，故陣列名稱可當作指標來使用，也就是說 void** q 與 void* q[]是同義，而 void* q[]即表示 q 陣列中的元素其資料形態是 void*。

C 語言中指標的另一種神奇的功能是函數的輸入指標是可以作為輸出功能，例如：

```
void    xyz(int*    p)
```

則 xyz 函數是不能夠以 return 指令來輸出資料，但是卻能夠以輸入參數 p 位址來輸出資料，如果指標 p 的資料僅是呼叫 xyz 函數的任務在單獨使用，則 p 可以是區域變數位址，如果指標 p 的資料將會提供給眾多任務來存取，則 p 就必須是全域變數位址。

肆、程式內容

tinyOS 包含三個檔案：SIZE.h、OS.h 與 os.c，以及實現內文交換的中斷向量程式 PendSV_Handler，SIZE.h 是程式設計師惟一必須指定數值的檔案，tinyOS 所有原始程式碼如下所示：

一、SIZE.h

以下 8 常數視應用程式之需求而設定。

```
#define   TASKSIZE          (int) 13
#define   PADDING           (char) 19
#define   PENDSIZE          (int) 12
#define   MEMORYPOOLBYTES   (int) 0
#define   QSIZE             (int) 0
#define   QBYTES            (int) 0
#define   FLAGSIZE          (int) 0
#define   MUTEXSIZE         (int) 0
```

二、OS.h

```
#define   OSCLOCK_1S          SystemCoreClock   //define in system_LPC11xx.c
#define   OSCLOCK_500mS       SystemCoreClock/2
#define   OSCLOCK_200mS       SystemCoreClock/5
#define   OSCLOCK_100mS       SystemCoreClock/10 //10 OS tick clock per second
#define   OSCLOCK_10mS        SystemCoreClock/100
#define   OSCLOCK_1mS         SystemCoreClock/1000
#define   OSCLOCK_100uS       SystemCoreClock/10000
#define   OSCLOCK_10K         10000
#define   OSCLOCK_100K        100000
#define   OSCLOCK_1M          1000000
#define   clockOS             (unsigned int) OSCLOCK_100Ms
#define   TIMEOUT_INFINITE    (int) -1
#define   SEM                 (char) 0
#define   MAIL                (char) 1
#define   FLAG                (char) 2
#define   Q                   (char) 3
#define   MUTEX               (char) 4
```

```
#define   FLAG_CLEAR          (char) 0
#define   FLAG_SET            (char) 1
#define   FLAG_MATCH_ANY (char) 0
#define   FLAG_MATCH_ALL (char) 1
                  // application function
char      startOS(void (*[])(void), int, int, unsigned int);
int       findOptimalPaddingOS(void);
void      delayTickOS(int);
void      delayTimeOS(int, int, int, int);
char      chechOwnEventOS(void);
void      postSemOS(int);
void      postMailOS(int, void*);
void      postFlagOS(int, unsigned int, char);
void      postQOS(int, void*);
void      postMutexOS(void);
void      deleteEventNumberSelfOS(char);
in        pendSemOS(int*, int);
int       pendMailOS(int*, int);
int       pendFlagOS(int*, unsigned int*, char, int);
int       pendQOS(int*, int);
int       pendMutexOS(int*, int);
void*     readMailOS(void);
int       readQOS(int, void**);
void      setQOS(void);
void*     getMemoryOS(int);
int       getMemory3OS(int bytes, void**, int*);
char      putMemoryOS(void*);
int       queryFreeBulkNoOS(char*);
void      resetMemoryPoolOS(char);
char      checkPublicFlagBitOS(int, char);
unsigned int queryPublicFlagOS(int);
```

三、startup_LPC11xx.s

同 tinyOS1。

四、os.c

使用 NXP LPC1114 系列的 Cortex-M0 平台作測試。

```
#include <LPC11xx.h>
#include "SIZE.h"
#include "OS.h"
#define   DISABLE_INTERRUPT __ASM{cpsid i}
#define   ENABLE_INTERRUPT  __ASM{cpsie i}
#define   TABLELENGTH          (int) ((TASKSIZE+1)/33+1)
#define   BULKBYTES            (int) sizeof(long double)
#define GLOBALBULKLENGTH (int) (MEMORYPOOLBYTES%BULKBYTES?
          MEMORYPOOLBYTES/BULKBYTES+1:MEMORYPOOLBYTES/BULKBYTES)
#define   FREEBULKLENGTH (int) (GLOBALBULKLENGTH?GLOBALBULKLENGTH/33
          +1:0)
#define   QLENGTH              (int) (QBYTES/sizeof(void*))

typedef struct
{
   unsigned int      sp;          //sp must be the first element
   int               pack[PADDING];
   unsigned int      registerStack[16];
} stackOS;           //the same as SP address

typedef struct
{
   void  **q;
   Int   inIndex;   //the only entry of array index to push
   int   outIndex;
   int   items;
}qbodyOS;

typedef struct
{
   int               priority;          //pending task
   int               *numberArray;
   int               readyNumber;
   char              eventType;
   unsigned int      *privateFlag;      //flag
   void              *receiveMessage;   //box
} eventOS;
```

```
    stackOS         *CurrentTaskOS;             //the same as SP address
    stackOS         *NextTaskOS;                //the same as SP address
    stackOS         TaskOS[TASKSIZE+1];         //include idleTaskOS()
    int             WaitTickOS[TASKSIZE];
    unsigned int    ReadyTableOS[TABLELENGTH];  //bit 1 is ready, priority is bit index
    int             CurrentPriorityOS; //priority<=TASKSIZE-1, priority begin from 0, priority>=0
    unsigned int    TickPerSecondOS;
    eventOS         EventNumberTaskOS[PENDSIZE];
    qbodyOS         QBodyOS[QSIZE];
    long double     PoolOS[GLOBALBULKLENGTH];
    unsigned int    FreeBulkNoOS[FREEBULKLENGTH];
    int             MutexOwnerOS[MUTEXSIZE];    //owner priority
    unsigned int    PublicFlagOS[FLAGSIZE];  //flags used, initialize public flag value
    unsigned int    FlagAllOrAnyOS[TABLELENGTH];
    unsigned int    PriorityOwnEventOS[TABLELENGTH];
    char            SysTickCountOS=0;
    char            ReadSysTickCountOS=0;

void setTableOS(unsigned int* Table, int priority)          //Table can be returned
{
   int    index;        //active priority=32*IndexOS+BitOS
   char bit;
   index=priority/32;
   bit=priority%32;
   Table[index] | =(1<<bit);
}

void clearTableOS(unsigned int* Table, int priority)       //Table can be returned
{
   int    index;        //active priority=32*IndexOS+BitOS
   char bit;
   index=priority/32;
   bit=priority%32;
   Table[index] &=~(1<<bit);
}
   //return 1 or 0 only
```

```
char checkSetBitOS(unsigned int *Table, int priority)
{
    unsigned int priorityBit=0x0;
    int    index;
    char bit;
    index=priority/32;
    bit=priority%32;
    priorityBit | =(1<<bit);
    priorityBit &=Table[index];
    if (priorityBit==0x0)
    {
        return 0;
    }
    return 1;
}
        //Table must not be zero

__ASM int findLeastBitOS (unsigned int Table)
{
    MOVS R3, #0
loop1
    LSRS R0, R0, #1
    BCS return1
    ADDS R3, R3, #1
    CMP R3, #32
    BLT loop1
return1
    MOV R0, R3
    BX LR
}

__ASM int interruptNumberOS()
{
    MRS R0, IPSR
    BX LR
}
```

```
void executeHighestPriorityTaskOS(void)
{
    int highestPriority;
    int index;
    int base;
    int i;
    char setBit;
    setBit=checkSetBitOS(PriorityOwnEventOS, CurrentPriorityOS);
    if((!setBit) || (CurrentPriorityOS==TASKSIZE))
    {
        index=0;
        while (ReadyTableOS[index]==0)
        {
            index++;
        }
        highestPriority=findLeastBitOS(ReadyTableOS[index]);
        if(index>=1)
        {
            base=0;
            for (i=index; i>0; i--)
            {
                base+=32;
            }
            highestPriority+=base;
        }
        if(highestPriority!=CurrentPriorityOS)
        {
            DISABLE_INTERRUPT;
            CurrentTaskOS=&TaskOS[CurrentPriorityOS];
            NextTaskOS=&TaskOS[highestPriority];
            CurrentPriorityOS=highestPriority;
            SCB->ICSR | =1<<28;
            ENABLE_INTERRUPT;
        }
    } //if((!setBit) || (CurrentPriorityOS==TASKSIZE))
}
```

```
void initializeTaskOS(void (*handler)(void), int priority)
{
    if(handler)
    {
        setTableOS(ReadyTableOS, priority);
        TaskOS[priority].sp=(unsigned int)(&TaskOS[priority].registerStack[0]);
        TaskOS[priority].registerStack[15]=0x01000000;
        TaskOS[priority].registerStack[14]=(unsigned int)handler;
    }
}

void idleTaskOS(void)
{
    while(1)
    {
    }
}

void setQOS(void)
{
    int i;
    for(i=0; i<FREEBULKLENGTH; i++)  //must before getMemoryOS()
    {
        FreeBulkNoOS[i]=0xffffffff;        //1 is available
    }
    for(i=0; i<QSIZE; i++)        //must after setting FreeBulkNoOS[]
    {
        QBodyOS[i].q=(void**)getMemoryOS(QBYTES);
        QBodyOS[i].inIndex=0;
        QBodyOS[i].outIndex=0;
        QBodyOS[i].items=0;
    }
}

char startOS(void (*taskName[])(void), int arraySize, int startPriority, unsigned int OS_clock)
{
    int i;
    if(arraySize! =TASKSIZE)
```

```
{
	return 1;	//error will stop OS
}
			//Q need MEMORYPOOLBYTES
if(MEMORYPOOLBYTES<QSIZE*(QBYTES+sizeof(long double)))
{
	return 2;	//error will stop OS
}
for(i=0; i<TABLELENGTH; i++)
{
	PriorityOwnEventOS[i]=0x0;
	FlagAllOrAnyOS[i]=0x0;
}
for(i=0; i<PENDSIZE; i++)
{
		EventNumberTaskOS[i].priority=-1;
		EventNumberTaskOS[i].numberArray=0x0;
		EventNumberTaskOS[i].readyNumber=-1;
		EventNumberTaskOS[i].eventType=(char) -1;
		EventNumberTaskOS[i].privateFlag=0x0;
		EventNumberTaskOS[i].receiveMessage=0x0;
}
for(i=0; i<FLAGSIZE; i++)
{
			PublicFlagOS[i]=0x0;
}
for(i=0; i<MUTEXSIZE; i++)
{
		MutexOwnerOS[i]=-1;
}
setQOS();
			//initialize user's tasks
for(i=0; i<=(TASKSIZE-1); i++)  //idleTaskOS's priority=TASKSIZE
{
			WaitTickOS[i]=0;
			initializeTaskOS(taskName[i], i);
}
```

```
    initializeTaskOS(idleTaskOS, TASKSIZE);    //create system's idleTaskOS()
                    // successfully start OS
    NVIC_SetPriority(PendSV_IRQn, 0xff);        //lowest priority
    NVIC_SetPriority(SysTick_IRQn,0x0);//highest priority
    SysTick_Config(OS_clock);//OS_clock is defined in OS.h
    TickPerSecondOS=SystemCoreClock/OS_clock;
    CurrentPriorityOS=startPriority;
    CurrentTaskOS=&TaskOS[CurrentPriorityOS];
    __set_PSP((int)&CurrentTaskOS->registerStack[15]+4);
    __set_CONTROL(0x03);/*Switch to PSP,unprivilleged mode*/
    __ISB();/*Exec. ISB after changing CONTORL(recommended)*/
    taskName[CurrentPriorityOS]();
    return 0;    //never reach if start successfully
}

void resumeTaskOS(int priority)
{
    DISABLE_INTERRUPT;
            WaitTickOS[priority]=0;
                    setTableOS(PriorityOwnEventOS, priority);
                    setTableOS(ReadyTableOS, priority);
            ENABLE_INTERRUPT;
}

int currentPriorityMapEventIndexOS(char eventType)
{
    int i;
    int index=-1;
    for(i=0; i<PENDSIZE; i++)
    {
            if(((EventNumberTaskOS[i].priority==CurrentPriorityOS) &&
           (EventNumberTaskOS[i].eventType==eventType)) || (EventNumberTaskOS[i].priority==-1))
            {
                    index=i;
                    break;
            }
    }
    if(index>=PENDSIZE)
```

```
    {
            index=-1;          //error
    }
    return index;
}

char IsPending(char eventType)
{
        int i;
        char pend=0;
    for(i=0; i<PENDSIZE; i++)
    {
        if((EventNumberTaskOS[i].numberArray!=0x0) && (EventNumberTaskOS[i].eventType==
        eventType))
        {
            pend=1;
            break;
        }
    }
    return pend;
}

void postSemOS(int number)
{
    int i;
    int priority;
    int *array;
    int previousValue;
    int k;
    int index;
    if(number>=0)
    {
        DISABLE_INTERRUPT;
        clearTableOS(PriorityOwnEventOS, CurrentPriorityOS);
        ENABLE_INTERRUPT;
        index=currentPriorityMapEventIndexOS(SEM);
        if(index>=0)
        {
```

```
        DISABLE_INTERRUPT;
        EventNumberTaskOS[index].numberArray=0x0;
        EventNumberTaskOS[index].readyNumber=-1;
        ENABLE_INTERRUPT;
    }
    if(IsPending(SEM))
    {
        for(i=0; i<PENDSIZE; i++)
        {
            if(EventNumberTaskOS[i].eventType==SEM)
            {
                priority=EventNumberTaskOS[i].priority;
                if(priority!=CurrentPriorityOS)
                {
                    array=EventNumberTaskOS[i].numberArray;
                    previousValue=-3;
                    k=0;
                    while((array[k]>=0) && (array[k]!=previousValue))
                    {
                        if( array[k]==number)
                        {
                            DISABLE_INTERRUPT;
                            EventNumberTaskOS[i].numberArray=0x0;
                            EventNumberTaskOS[i].readyNumber=number;
                            ENABLE_INTERRUPT;
                            resumeTaskOS(priority);
                        }
                        else
                        {
                            previousValue=array[k];
                        }
                        k++;
                    } //while
                } //if(priority!=CurrentPriorityOS)
            } //if(EventNumberTaskOS[i].eventType==SEM)
        } //for
    } //if(IsPending(SEM))
```

```
    } //if(number>=0)
    if(interruptNumberOS()==0)
    {
        executeHighestPriorityTaskOS();        //successful
    }
}
        //message must be global address

void postMailOS(int number, void *message)
{
    int i;
    int priority;
    int *array;
    int previousValue;
    int k;
    int index;
    if(number>=0)
    {
        DISABLE_INTERRUPT;
        clearTableOS(PriorityOwnEventOS, CurrentPriorityOS);
        ENABLE_INTERRUPT;
        index=currentPriorityMapEventIndexOS(MAIL);
        if(index>=0)
        {
            DISABLE_INTERRUPT;
            EventNumberTaskOS[index].numberArray=0x0;
            EventNumberTaskOS[index].readyNumber=-1;
            ENABLE_INTERRUPT;
        }
        if(IsPending(MAIL))
        {
            for(i=0; i<PENDSIZE; i++)
            {
                if(EventNumberTaskOS[i].eventType==MAIL)
                {
                    priority=EventNumberTaskOS[i].priority;
                    if ( priority!=CurrentPriorityOS)
```

```
                    {
                        array=EventNumberTaskOS[i].numberArray;
                        previousValue=-3;
                        k=0;
                        while((array[k]>=0) && (array[k]!=previousValue))
                        {
                            if(array[k]==number)
                            {
                                DISABLE_INTERRUPT;
                                EventNumberTaskOS[i].numberArray=0x0;
                                EventNumberTaskOS[i].readyNumber=number;
                                EventNumberTaskOS[i].receiveMessage=message;
                                ENABLE_INTERRUPT;
                                resumeTaskOS(priority);
                            }
                            else
                            {
                                previousValue=array[k];
                            }
                            k++;
                        } //while
                    } //if(priority!=CurrentPriorityOS)
                } //if(EventNumberTaskOS[i].eventType==BOX)
            } //for
        } //if(IsPending(BOX))
    } //if(number>=0)
    if(interruptNumberOS()==0)
    {
        executeHighestPriorityTaskOS();
    }
}

void postFlagOS(int number, unsigned int modifyPublicFlag, char setOrClear)
{
    int i;
    int priority;
    unsigned int interestBits;
```

```
char AllOrAny;
char match;
int *array;
int previousValue;
int k;
int index;
if((number>=0) && (number<FLAGSIZE))
{
    DISABLE_INTERRUPT;
    clearTableOS(PriorityOwnEventOS, CurrentPriorityOS);
    switch (setOrClear)     //calculate PublicFlagOS[]
    {
        case FLAG_CLEAR:          //0
            PublicFlagOS[number] &=~modifyPublicFlag;    // clear the interest bits
            break;
        case FLAG_SET:   //flagNumber is the index of PublicFlagOS[] and must start from 0
            PublicFlagOS[number] | =modifyPublicFlag;    //set the interest bits
            break;
    }
    ENABLE_INTERRUPT;
    index=currentPriorityMapEventIndexOS(FLAG);
    if(index>=0)
    {
        DISABLE_INTERRUPT;
        EventNumberTaskOS[index].numberArray=0x0;
        EventNumberTaskOS[index].readyNumber=-1;
        ventNumberTaskOS[index].privateFlag=0x0;        //reset private flag
        ENABLE_INTERRUPT;
    }
    if(IsPending(FLAG))
    {
        for(i=0; i<PENDSIZE; i++)
        {
            if(EventNumberTaskOS[i].eventType==FLAG)
            {
                priority=EventNumberTaskOS[i].priority;
                if(priority!=CurrentPriorityOS)
```

```
{
    array=EventNumberTaskOS[i].numberArray;
    previousValue=-3;
    k=0;
    while((array[k]>=0) && (array[k]!=previousValue))
    {
        if(array[k]==number)
        {
            interestBits=PublicFlagOS[number] &
            *(EventNumberTaskOS[i].privateFlag); //compare bit value 1
            match=0;
            AllOrAny=checkSetBitOS(FlagAllOrAnyOS, priority);
            switch(AllOrAny)
            {
                case FLAG_MATCH_ALL:
                    if(interestBits==
                       *EventNumberTaskOS[i].privateFlag)
                    {
                        match=1;
                    }
                    break;
                case FLAG_MATCH_ANY:
                    if(interestBits!=(unsigned int)0)
                    {
                        match=1;
                    }
                    break;
                } //switch
                if(match)
                {
                    DISABLE_INTERRUPT;
                    EventNumberTaskOS[i].numberArray=0x0;
                    EventNumberTaskOS[i].readyNumber=number;
                    EventNumberTaskOS[i].privateFlag=0x0;
                    ENABLE_INTERRUPT;
                    resumeTaskOS(priority);
                }
```

```
                                        } //if(array[arrayIndex]==number)
                                        else
                                        {
                                            previousValue=array[k];
                                        }
                                        k++;
                                    } //while
                                } //if(priority!=CurrentPriorityOS)
                            } //if(EventNumberTaskOS[i].eventType==FLAG)
                        } //for
                    } //if(IsPending(FLAG))
                } //if(number<FLAGSIZE)
                if(interruptNumberOS()==0)
                {
                    executeHighestPriorityTaskOS();
                }
}

void postMutexOS(void)
{
    int number;
    int index;
    if(interruptNumberOS()==0)        //ISR can not call postMutexOS()
    {
        index=currentPriorityMapEventIndexOS(MUTEX);
        if(index>=0)
        {
            number=EventNumberTaskOS[index].readyNumber;
            if((number>=0) && (number<MUTEXSIZE))
            {
                if(MutexOwnerOS[number]==CurrentPriorityOS)
                {
                    DISABLE_INTERRUPT;
                    MutexOwnerOS[number]=-1;       //mutex is free
                    clearTableOS(PriorityOwnEventOS, CurrentPriorityOS);
                    EventNumberTaskOS[index].numberArray=0x0;
                    EventNumberTaskOS[index].readyNumber=-1;
```

```
                    ENABLE_INTERRUPT;
                }
            } //if((number>=0) && (number<MUTEXSIZE))
        } //if(index>=0)
        executeHighestPriorityTaskOS();        //successful
    } //if( interruptNumberOS()==0)
}

void postQOS(int number, void *message)
{
    int i;
    int priority;
    int inIndex;
    int *array;
    int k;
    int previousValue;
    if((number>=0) && (number<QSIZE) && (QBodyOS[number].q!=0x0))
    {
        DISABLE_INTERRUPT;
        clearTableOS(PriorityOwnEventOS, CurrentPriorityOS);
        inIndex=QBodyOS[number].inIndex++;
        QBodyOS[number].q[inIndex]=message; //user must manage data type.
        QBodyOS[number].items++;
        ENABLE_INTERRUPT;
        if(QBodyOS[number].inIndex>=QLENGTH)
        {
            DISABLE_INTERRUPT;
            QBodyOS[number].inIndex=0;
            ENABLE_INTERRUPT;
        }
        if(IsPending(Q))
        {
            for(i=0; i<PENDSIZE; i++)
            {
                if(EventNumberTaskOS[i].eventType==Q)
                {
                    priority=EventNumberTaskOS[i].priority;
```

```
                        if(priority!=CurrentPriorityOS)
                        {
                            array=EventNumberTaskOS[i].numberArray;
                            previousValue=-3;
                            k=0;
                            while((array[k]>=0) && (array[k]!=previousValue))
                            {
                                if( array[k]==number)
                                {
                                    DISABLE_INTERRUPT;
                                    EventNumberTaskOS[i].readyNumber=number;
                                    ENABLE_INTERRUPT;
                                    resumeTaskOS(priority);
                                }
                                else
                                {
                                    previousValue=array[k];
                                }
                                k++;
                            } //while
                        } //if(priority!=CurrentPriorityOS)
                    } //if(EventNumberTaskOS[i].eventType==Q)
                } //for
            } //if(IsPending(Q))
        } //if(number>=0)
        if(interruptNumberOS()==0)
        {
            executeHighestPriorityTaskOS();          //successful
        }
}

void deleteEventNumberSelfOS(char eventType)
{
    int index;
    index=currentPriorityMapEventIndexOS(eventType);
    if(index>=0)
    {
```

```
            DISABLE_INTERRUPT;
            EventNumberTaskOS[index].numberArray=0x0;    //does not pend Sem
            ENABLE_INTERRUPT;
        }
    }

    void pauseTaskOS(int timeout)
    {
        DISABLE_INTERRUPT;
        WaitTickOS[CurrentPriorityOS]=timeout;
        clearTableOS(PriorityOwnEventOS, CurrentPriorityOS);
        clearTableOS(ReadyTableOS, CurrentPriorityOS);
        ENABLE_INTERRUPT;
    }
                    //use for synchronized tasks with slow OS tick

    void ensureOneTickPassOS(void)
    {
        while(SysTickCountOS==ReadSysTickCountOS)
        {
        }
        DISABLE_INTERRUPT;
        ReadSysTickCountOS=SysTickCountOS;
        ENABLE_INTERRUPT;
    }

    char justifyNumberArray(int *array)    //terminate sign
    {
        int previousNumber=-3;
        int i;
        char error;
        i=0;
        while((array[i]>=0) && (array[i]!=previousNumber))
        {
            previousNumber=array[i];
            i++;
        }
        error=1;
```

```
    if(array[i]<0)
    {
        error=0;
    }
    return error;
}

int readReadyNumberOS(char eventType)
{
    char setBit;
    int index;
    int readyNumber=-1;
    setBit=checkSetBitOS(PriorityOwnEventOS, CurrentPriorityOS);
    if(setBit>0)
    {
        index=currentPriorityMapEventIndexOS(eventType);
        readyNumber=EventNumberTaskOS[index].readyNumber;
    }
    return readyNumber;
}

int pendErrorCode(int *array, char eventType)
{
    int terminate;
    int readyNumber;
    terminate=justifyNumberArray(array);  //error:terminate=1
    readyNumber=readReadyNumberOS(eventType);  //error:readyNumber=-1
    if(terminate==1)
    {
        readyNumber=-2;       //No terminate sign -1
    }
    return readyNumber;
}

int pendSemOS(int *array, int timeout)
{
    int index;
    int readyNumber;
```

```
    if(interruptNumberOS()==0)
    {
        ensureOneTickPassOS();
        index=currentPriorityMapEventIndexOS(SEM);
        if(index>=0)
        {
            DISABLE_INTERRUPT;
            EventNumberTaskOS[index].priority=CurrentPriorityOS;
            EventNumberTaskOS[index].numberArray=array;
            EventNumberTaskOS[index].eventType=SEM;
            ENABLE_INTERRUPT;
            pauseTaskOS(timeout);
            executeHighestPriorityTaskOS();
        }
        readyNumber=pendErrorCode(array, SEM);
    } //if(interruptNo==0)
    return readyNumber;
}

int pendMailOS(int *array, int timeout)
{
    int index;
    int readyNumber;
    if(interruptNumberOS()==0)
    {
        ensureOneTickPassOS();
        index=currentPriorityMapEventIndexOS(MAIL);
        if( index>=0)
        {
            DISABLE_INTERRUPT;
            EventNumberTaskOS[index].priority=CurrentPriorityOS;
            EventNumberTaskOS[index].numberArray=array;
            EventNumberTaskOS[index].eventType=MAIL;
            ENABLE_INTERRUPT;
            pauseTaskOS(timeout);
            executeHighestPriorityTaskOS();
        }
```

```
        readyNumber=pendErrorCode(array, MAIL);
    } //if
    return readyNumber;
}

int pendFlagOS(int *array, unsigned int *privateFlag, char allOrAny, int timeout)
{
    int index;
    int readyNumber;
    if(interruptNumberOS()==0)
    {
        ensureOneTickPassOS();
        index=currentPriorityMapEventIndexOS(FLAG);
        if(index>=0)
        {
            DISABLE_INTERRUPT;
            EventNumberTaskOS[index].priority=CurrentPriorityOS;
            EventNumberTaskOS[index].numberArray=array;
            EventNumberTaskOS[index].privateFlag=privateFlag;
            EventNumberTaskOS[index].eventType=FLAG;
            if(allOrAny==FLAG_MATCH_ALL)
            {
                setTableOS(FlagAllOrAnyOS, CurrentPriorityOS);
            }
            else
            {
                clearTableOS(FlagAllOrAnyOS, CurrentPriorityOS);
            }
            ENABLE_INTERRUPT;
            pauseTaskOS(timeout);
            executeHighestPriorityTaskOS();
        } //if( index>=0)
        readyNumber=pendErrorCode(array, FLAG);
    } //if(interruptNumberOS()==0)
    return readyNumber;
}
```

```
int pendMutexOS(int *array, int timeout)
{
    int index;
    int number;
    int readyNumber;
    if(interruptNumberOS()==0)
    {
        ensureOneTickPassOS();
        index=currentPriorityMapEventIndexOS(MUTEX);
        if(index>=0)
        {
            DISABLE_INTERRUPT;
            EventNumberTaskOS[index].priority=CurrentPriorityOS;
            EventNumberTaskOS[index].numberArray=array;
            EventNumberTaskOS[index].eventType=MUTEX;
            ENABLE_INTERRUPT;
            number=array[0];        //only one mutex
            if((number<MUTEXSIZE) && (MutexOwnerOS[number]<0))
            {
                DISABLE_INTERRUPT;
                MutexOwnerOS[number]=CurrentPriorityOS;
                setTableOS(PriorityOwnEventOS, CurrentPriorityOS);
                EventNumberTaskOS[index].numberArray=0x0;
                EventNumberTaskOS[index].readyNumber=number;
                ENABLE_INTERRUPT;
            } //if(number<MUTEXSIZE)
            else
            {
                pauseTaskOS(timeout);
                executeHighestPriorityTaskOS();
            }
        } //if(index>=0)
        readyNumber=readReadyNumberOS(MUTEX);
    } //if(interruptNumberOS()==0)
    return readyNumber;
}
```

```
int pendQOS(int *array, int timeout)
{
    int index;
    int readyNumber;
    int k;
    int previousValue;
    int number;
    if(interruptNumberOS()==0)
    {
        ensureOneTickPassOS();
        index=currentPriorityMapEventIndexOS(Q);
        if(index>=0)
        {
            DISABLE_INTERRUPT;
            EventNumberTaskOS[index].priority=CurrentPriorityOS;
            EventNumberTaskOS[index].numberArray=array;
            EventNumberTaskOS[index].eventType=Q;
            ENABLE_INTERRUPT;
            previousValue=-3;
            k=0;
            number=array[k];
            while((QBodyOS[number].items==0) && ((number>=0) && (number!=
                previousValue)))
            {
                previousValue=number;
                k++;
                number=array[k];
            }
                if((number<QSIZE) && (QBodyOS[number].items>0))
                {
                    DISABLE_INTERRUPT;
                    setTableOS(PriorityOwnEventOS, CurrentPriorityOS);
                    EventNumberTaskOS[index].numberArray=0x0;
                    EventNumberTaskOS[index].readyNumber=number;
                    ENABLE_INTERRUPT;
                } //if(number<MUTEXSIZE)
                else
```

```
                    {
                        pauseTaskOS(timeout);
                        executeHighestPriorityTaskOS();
                    }
                } //if( index>=0)
                readyNumber=pendErrorCode(array, Q);
    } //if
    return readyNumber;
}

void SysTick_Handler(void)
{
    int i;
    char schedule=0;
    DISABLE_INTERRUPT;
        //one Tick Pass and set ready
    for(i=0; i<=TASKSIZE-1; i++)    //i is task's priority
    {
        if(WaitTickOS[i]>=1)
        {
            WaitTickOS[i]--;
        }
        if(WaitTickOS[i]==0)
        {
            setTableOS(ReadyTableOS, i);
            schedule=1;
        }
    } //for
    SysTickCountOS++;
    ENABLE_INTERRUPT;
    if(schedule)
    {
        executeHighestPriorityTaskOS();
    }
}
```

```
void delayTickOS(int tick)
{
    if(tick>0)
    {
        pauseTaskOS(tick);
    }
    if(interruptNumberOS()==0)
    {
        executeHighestPriorityTaskOS();
    }
}

void delayTimeOS(int hour, int minute, int second, int mS)
{
    int tick;
    short int remainder;
    if((hour>=0) && (minute>=0) && (second>=0) && (mS>=0))
    {
        tick=TickPerSecondOS*(hour*3600+minute*60+second);
        tick+=TickPerSecondOS*mS/1000;
        remainder=TickPerSecondOS*mS%1000;
        if ( remainder>=500)
        {
            tick++;
        }
        if(tick<1)
        {
            tick=1;
        }
        pauseTaskOS(tick);
    }
    if(interruptNumberOS()==0)
    {
        executeHighestPriorityTaskOS();
    }
}
```

```
int findOptimalPaddingOS(void)
{
    int pack;
    int i;
    int maximum=-1;
    for(i=0; i<TASKSIZE; i++)
    {
        pack=PADDING-((int)TaskOS[i].sp-(int)&TaskOS[i])/4;
        if(pack>maximum)
        {
            maximum=pack;
        }
    }
    return maximum;
}

long double* memoryAddress(int bulkNo)
{
    long double*bulkAddress;
    bulkAddress=(long double*)((unsigned int)PoolOS+(unsigned int)(bulkNo*BULKBYTES));
    return bulkAddress;
}
            //bulkAddress=0x0 is error

void* getMemoryOS(int bytes)
{
    int i;
    int k;
    int r;
    int bulkLength;         //available data length
    int frontBulkNo;
    int rearBulkNo;
    int minimum=999999;
    char found;
    char setBit;
    long double *relyAddress=0x0;
    bulkLength=bytes%BULKBYTES?bytes/BULKBYTES+1:bytes/BULKBYTES;
    found=0;
```

```
i=0;
while((i<=GLOBALBULKLENGTH-bulkLength+2) && (!found))
{
	setBit=checkSetBitOS(FreeBulkNoOS, i);
	if(setBit)	//available memory
	{
		k=1;
		setBit=checkSetBitOS(FreeBulkNoOS, i+1);
		while((i+k < GLOBALBULKLENGTH) && setBit)
		{
			k++;
			setBit=checkSetBitOS(FreeBulkNoOS, i+k);
		}
		frontBulkNo=i<1? 0:i+1; // i=0 i is data bulk, i>0 i is stopBulkNo
		if(i+k==GLOBALBULKLENGTH)
		{
			if((k>=bulkLength) && (k<minimum))
			{
				rearBulkNo=i<1? bulkLength-1: i+bulkLength;
				found=1;
				minimum=k;
			}
			else		//memory is too small
			{
				i+=k;
			}
		}
		else if(!setBit)	//FreeBulkNoOS[i+k]=0, FreeBulkNoOS[i+k-1]=1
		{
			if((k-2>=bulkLength) && (k<minimum))
			{
				rearBulkNo=i<1? bulkLength-1: i+bulkLength;
				found=1;
				minimum=k;
			}
			else		//memory is too small
			{
```

```
                        i+=k;
                    }
                }
            } //if(setBit)
            else
            {
                i++;
            }
        } // while
        if(found)
        {
            relyAddress=memoryAddress(frontBulkNo);
            for(r=frontBulkNo; r<=rearBulkNo; r++)      //FreeBulkNoOS[stopBulkNo]=1
            {
                clearTableOS(FreeBulkNoOS, r);          //using
            }
        }
        return relyAddress;
    }

    int getMemory3OS(int bytes, void **relyMarginDanger, int *dangerBytes)
    {
        int i;
        int k;
        int r;
        int bulkLength;   //available data length
        int frontBulkNo;
        int rearBulkNo;
        int relyBytes=0;
        int minimum= 999999;
        char found;
        char setBit;
        *dangerBytes=-1;
        relyMarginDanger[0]=0x0;          //relyAddress
        relyMarginDanger[1]=0x0;          //marginAddress
        relyMarginDanger[2]=0x0;          //dangerAddress
        bulkLength=bytes%BULKBYTES? bytes/BULKBYTES+1: bytes/BULKBYTES;
```

```
found=0;
i=0;
while((i<=GLOBALBULKLENGTH-bulkLength+2) && (!found))
{
	setBit=checkSetBitOS(FreeBulkNoOS, i);
	if(setBit)	//available memory, i is stopBulkNo normally
	{
		k=1;
		setBit=checkSetBitOS(FreeBulkNoOS, i+1); //i+1 is freeBulkNo normally
		while((i+k<GLOBALBULKLENGTH) && setBit)
		{
			k++;
			setBit=checkSetBitOS(FreeBulkNoOS, i+k);
		}
		frontBulkNo=i<1? 0: i+1;
		if(i+k==GLOBALBULKLENGTH)
		{
			if((k>=bulkLength) && (k<minimum))
			{
				rearBulkNo=i<1? bulkLength-1: i+bulkLength;
				found=1;
				minimum=k;
			}
			else		//memory is too little
			{
				i+=k;
			}
		}
		else if(!setBit)	//FreeBulkNoOS[i+k]=0, FreeBulkNoOS[i+k-1]=1
		{
			if((k-2>=bulkLength) && (k<minimum))
			{
				rearBulkNo=i<1? bulkLength-1: i+bulkLength;
				found=1;
				minimum=k;
			}
			else
```

```
                    {
                        i+=k;
                    }
                }
            } //if(setBit)
            else
            {
                i++;
            }
        } //while
            if(found)
            {
                relyMarginDanger[0]=memoryAddress(frontBulkNo); // relyAddress
                for(r=frontBulkNo; r<=rearBulkNo; r++) //FreeBulkNoOS[stopBulkNo]=1
                {
                    clearTableOS(FreeBulkNoOS, r);
                }
                if(rearBulkNo<=GLOBALBULKLENGTH-2)
                {
                    relyMarginDanger[1]=memoryAddress(rearBulkNo+1); //marginAddress
                }
                if(rearBulkNo<=GLOBALBULKLENGTH-3)
                {
                    relyMarginDanger[2]=(void*)((unsigned int)relyMarginDanger[1]+
                                    unsigned int)BULKBYTES);      //dangerAddress
                }
                *dangerBytes=((i+k-1)-(rearBulkNo+2))*BULKBYTES;
                if(*dangerBytes<0)
                {
                    *dangerBytes=-1;
                }
                relyBytes=(bulkLength+1)*BULKBYTES;
            }
            return relyBytes;
    }
                //error=0 is successful
```

```
char putMemoryOS(void *putAddr)
{
    int i;
    int k;
    char error=1;
    char setBit;
    if(putAddr!=0x0)
    {
        for(i=0; i<GLOBALBULKLENGTH; i++)
        {
            if(putAddr==(void*)memoryAddress(i))        //i=frontBulkNo
            {
                for(k=i; k<GLOBALBULKLENGTH; k++)
                {
                    setBit=checkSetBitOS(FreeBulkNoOS, k);
                    if(setBit)
                    {
                        break;
                    }
                    setTableOS(FreeBulkNoOS, k);
                } //for(k
                error=0;
            } //if(putAddr
        } //for(i
    } //if(putAddr!=0x0)
    else
    {
        error=0;
    }
    return error;
}

int queryFreeBulkNoOS(char* result)
{
    int i;
    for(i=0; i<GLOBALBULKLENGTH; i++)
    {
```

```
            result[i]=checkSetBitOS(FreeBulkNoOS, i);
        }
        return GLOBALBULKLENGTH;
    }

    void resetMemoryPoolOS(char includeQ)
    {
        int i;
        int qBytes;
        int qBulks;
        qBytes=QSIZE*(QBYTES+sizeof(long double));
        qBulks=qBytes/BULKBYTES;    //exclude Q defaultly
        if(includeQ)
        {
            qBulks=0;
            for(i=0; i<QSIZE; i++)
            {
                QBodyOS[i].q=(void**)0x0;
            }
        }
        for(i=qBulks; i<GLOBALBULKLENGTH; i++)
        {
            setTableOS(FreeBulkNoOS, i);
        }
    }

    int readQOS(int number, void **retrieve)
    {
        int i;
        int outIndex;
        int items=0;
        char setBit;
        setBit = checkSetBitOS(PriorityOwnEventOS, CurrentPriorityOS);
        if((setBit>0) && (number>=0))
        {
            items=QBodyOS[number].items;
            if((QBodyOS[number].outIndex==0) && (items<QBodyOS[number].inIndex))
            {
```

```
            items=QBodyOS[number].inIndex;
        }
        for(i=0; i<items; i++)
        {
            DISABLE_INTERRUPT;
            outIndex=QBodyOS[number].outIndex++;
            ENABLE_INTERRUPT;
            retrieve[i]=QBodyOS[number].q[outIndex];
            if(QBodyOS[number].outIndex>=QLENGTH)
            {
                QBodyOS[number].outIndex=0;
            }
        } //for
        DISABLE_INTERRUPT;
        QBodyOS[number].items=0;
        ENABLE_INTERRUPT;
    } //if((setBit>0)  && (number>=0))
    return items;
}

void* readMailOS(void)
{
    int index;
    char setBit;
    void *messageAddress=0x0;
    setBit=checkSetBitOS(PriorityOwnEventOS, CurrentPriorityOS);
    if(setBit>0)
    {
        index=currentPriorityMapEventIndexOS(MAIL);
        DISABLE_INTERRUPT;
        messageAddress=EventNumberTaskOS[index].receiveMessage;
        ENABLE_INTERRUPT;
    }
    return messageAddress;
}
            //call chechOwnEventOS() after pend function
```

```
char chechOwnEventOS(void)
{
    return checkSetBitOS(PriorityOwnEventOS, CurrentPriorityOS);
}

unsigned int queryPublicFlagOS(int flagNumber)
{
    return PublicFlagOS[flagNumber];
}
                    //return value is either 1 or 0

char checkPublicFlagBitOS(int flagNumber, char bitNumber)
{
    return checkSetBitOS(&PublicFlagOS[flagNumber], bitNumber);
}
```

伍、啟動步驟

tinyOS 的啟動步驟非常簡潔，只有兩個步驟是必須，其餘皆視應用程式的需求而設定，主要是設定 SIZE.h 中的常數，啟動 tinyOS 的步驟如下所示，除了步驟(一)與(二)是任務相關而必需，步驟(三)至(七)是事件服務相關所以是選擇性設定，步驟(八)是用以更改作業系統的預設時脈也是選擇性的步驟。

(一)建立應用程式專屬 C 檔案，檔案中包含 main()主程式(可用模版 main())，而主體是任務程式的功能設計，對於 C 語言而言每一個任務都是一個函數，將 main()主程式內真正使用到的任務數量指定給 SIZE.h 檔案中的 TASKSIZE。

(二)適當指定 SIZE.h 檔案中的 PADDING，該值若太小則作業系統將誤動作或無法執行，該值若太大則會浪費記憶體空間，測試 PADDING 最佳值可先隨意指定 PADDING 值，然後由最後執行的任務來呼叫 findOptimalPaddingOS()函數，最佳 PADDING 值會在該函數回傳值附近，然後以該回傳 PADDING 值來取代隨意 PADDING 值，所謂的最佳 PADDING 值是指可以讓應用程式正常執行的最小 PADDING 值。

(三)將任務呼叫事件等待函數(pend 開頭的函數)的數量總合指定給 SIZE.h 檔案中的 PENDSIZE，預設為 0，delayTimeOS()與 delayTickOS()並不歸類於 PENDSIZE。

(四)設定動態記憶體管理的記憶體池大小，以 byte 為單位，將該數值指定給 SIZE.h 檔案中的 MEMORYPOOLBYTES，預設為 0。

(五)設定佇列的使用數量並指定給 SIZE.h 檔案中的 QSIZE，預設為 0，且將佇列的長度(以 byte 為單位)指定給 SIZE.h 檔案中的 QBYTES，預設為 0，tinyOS 規定每一個佇列的長度都是相同以簡化設定常數的步驟，且佇列是使用記憶體池，故必須 MEMORYPOOLBYTES>QSIZE×(QBYTES+8)。

(六)設定旗標的使用數量並指定給 SIZE.h 檔案中的 FLAGSIZE，預設為 0。

(七)設定互斥信號量的使用數量並指定給 SIZE.h 檔案中的 MUTEXSIZE，預設為 0。

(八)可在 OS.h 檔案中選擇性地更改作業系統預設時脈 clockOS。

陸、實驗測試

實驗將測試 tinyOS 的各項事件與服務功能，包括信號量、中斷與事件、訊息郵箱、互斥信號量、旗標、記憶體管理與佇列。對於 PADDING 常數的指定，除了信號量以及中斷的實驗是 findOptimalPaddingOS()函數的計算值再加 1 之外，其餘所有實驗的 PADDING 值都直接採用 findOptimalPaddingOS()函數的計算。微控制器的輸出結果將透過 UART 串列協定傳輸至電腦作顯示，電腦上的 UART 通信軟體是採用 AccessPort，微控制器上的 UART 通信函數是作者自行開發的 sendByte(char)函數與 print32bits(unsigned int)函數，sendByte()會傳送一個 byte 資料以便 AccessPort 轉成 ASCII 碼符號顯示，主要是用以鑑別當下是那一個任務正在執行，print32bits()會輸出變數的 16 進制碼資料形如 0x3355aaff，主要是用來顯示任務程式對於資料處理之結果。

主程式 main()函數可作為 tinyOS 啟動的模版函數，其所在的 C 檔案之重點是任務程式的設計，任務程式將依實驗性質個別呈現於後，C 檔案之形式如下：

```
#include <LPC11xx.h>
#include "OS.h"

void task0(void)
{
   . . . . . .
}

void task1(void)
{
   . . . . . .
}
. . . . . .
. . . . . .
. . . . . .

void taskN(void)
{
   . . . . . .
}
```

```
void(*taskName[])(void)={task0, task1,・・・・・・, taskN};
        //ErrorCode: 1-TaskCountOS!=TASKSIZE+1
        //ErrorCode: 2-MEMORYPOOLBYTES too little

int main(void)
{
    char errorCode;
    int startTaskIndex;
    int arraySize;

    initializeUART(9600);       //本文未展示 UART 之初始化設定
    arraySize=sizeof(taskName)/sizeof(taskName[0]);
    startTaskIndex = 0;
    errorCode=startOS(taskName, arraySize, startTaskIndex, clockOS);
    sendByte('0'+errorCode);    //never execute if startOS() successfully
}
```

實驗的過程主要是觀察各個 task 函數使用 tinyOS 所提供的服務之執行結果，所以測試的服務功能皆會展示 SIZE.h 檔案的常數值、任務程式碼與執行結果，微控制器是使用 NXP M0 LPC1114FN28/102(4KB SRAM)，開發環境是使用 Keil uVision5，所測試的核心服務功能依序如下：

一、信號量

範例是展示一個任務可以等待多個事件號碼(task10 與 task11)，且多個任務可以同時等待同一個事件號碼並同時一起被就緒(task10 與 task11 共同等待 task12 所發布的號碼)，task0 到 task4 是 task10 所啟動，task5 到 task9 是 task11 所啟動，task10 與 task11 是 task12 所啟動。

(一)SIZE.h

```
#define   TASKSIZE          (int) 13
#define   PADDING           (char) 19   //findOptimalPaddingOS()+1
#define   PENDSIZE          (int) 12
#define   MEMORYPOOLBYTES   (int) 0
#define   QSIZE             (int) 0
#define   QBYTES            (int) 0
#define   FLAGSIZE          (int) 0
#define   MUTEXSIZE         (int) 0
```

(二)任務程式碼

```
void task0(void)
{  int n[]={10, -1}; while(1)  {  sendByte('a');  sendByte('0');  pendSemOS(n,
   TIMEOUT_INFINITE);  delayTimeOS(0,0,0, 400);  }
}

void task1(void)
{  int n[]={11, -1}; while(1)  {  sendByte('a');  sendByte('1');  pendSemOS(n,
   TIMEOUT_INFINITE  );  delayTimeOS(0,0,0, 400);  }
}

void task2(void)
{  int n[]={12, -1}; while(1)  {  sendByte('a');  sendByte('2');  pendSemOS(n,
   TIMEOUT_INFINITE  );  delayTimeOS(0,0,0, 400);  }
}

void task3(void)
{  int n[]={13, -1}; while(1)  {  sendByte('a');  sendByte('3');  pendSemOS(n,
   TIMEOUT_INFINITE  );  delayTimeOS(0,0,0, 400);  }
}

void task4(void)
{  int n[]={14, -1}; while(1)  {  sendByte('a');  sendByte('4');  pendSemOS(n,
   TIMEOUT_INFINITE  );  delayTimeOS(0,0,0, 400);  }
}

void task5(void)
{  int n[]={20, -1}; while(1)  {  sendByte('b');  sendByte('5');  pendSemOS(n,
   TIMEOUT_INFINITE  );  delayTimeOS(0,0,0, 400);  }
}

void task6(void)
{  int n[]={21, -1}; while(1)  {  sendByte('b');  sendByte('6');  pendSemOS(n,
   TIMEOUT_INFINITE  );  delayTimeOS(0,0,0, 400);  }
}

void task7(void)
{  int n[]={22, -1}; while(1)  {  sendByte('b');  sendByte('7');  pendSemOS(n,
   TIMEOUT_INFINITE  );  delayTimeOS(0,0,0, 400);  }
}
```

```
void task8(void)
{  int n[]={23, -1}; while(1)  {  sendByte('b');  sendByte('8');  pendSemOS(n,
   TIMEOUT_INFINITE  );  delayTimeOS(0,0,0, 400);  }
}

void task9(void)
{  int n[]={24, -1}; while(1)  {  sendByte('b');  sendByte('9');  pendSemOS(n,
   TIMEOUT_INFINITE  );  delayTimeOS(0,0,0, 400);  }
}

void task10(void)
{
   int n[]={0,1,2,3,4, -1};
   int x;
   while(1)
   {
        sendByte('A');
        x=pendSemOS(n, TIMEOUT_INFINITE);
        postSemOS(x+10);
   }
}

void task11(void)
{
   int n[]={0,1,2,3,4, -1};
   int x;
   while(1)
   {
        sendByte('B');
        x=pendSemOS(n, TIMEOUT_INFINITE);
        postSemOS(x+20);
   }
}

void task12(void)
{
   while(1)
   {
        sendByte('Y');
```

```
            postSemOS(0);          delayTimeOS(0,0,1, 400);
            postSemOS(1);          delayTimeOS(0,0,1, 400);
            postSemOS(2);          delayTimeOS(0,0,1, 400);
            postSemOS(3);          delayTimeOS(0,0,1, 400);
            postSemOS(4);          delayTimeOS(0,0,1, 400);
        }
    }

    void (*taskName[])(void)={task0,task1,task2,task3,task4,task5,task6,task7,task8,task9,task10,task11, task12};
```

(三)執行結果

圖 1 是信號量功能之實驗結果，task0 至 task9 依序執行而顯示 a0、a1、a2、a3、a4、b5、b6、b7、b8、b9，再執行 task10、task11 與 task12 而依序顯示 A、B、Y，task12 依序發布信號量號碼 0 至 4 都會同時就緒 task10 與 task11，而 task10 會再就緒 task0 至 task4(顯示 a0 至 a4)、且 task11 會再就緒 task5 至 task9(顯示 b5 至 b9)，但 task10 的優先權比 task11 高，故先執行，會依序顯示 a0b5(task12 之號碼 0 所啟動)、a1b6(task12 之號碼 1 所啟動)、a2b7(task12 之號碼 2 所啟動)、a3b8(task12 之號碼 3 所啟動)、a4b9(task12 之號碼 4 所啟動)，再重新循環。

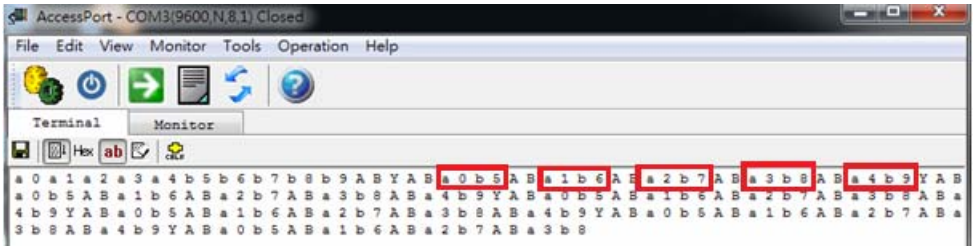

圖 1　信號量功能之實驗結果

二、中斷與事件

微控制器的 UART 中斷服務程式 UART_IRQHandler()接收到電腦 AccessPort 送來的資料後將會發布信號量 0 而就緒 task0，發送端是由電腦上的 AccessPort 以人工操作發送任意字元。

(一)SIZE.h

同信號量實驗之 SIZE.h

(二)任務程式碼

同信號量之任務程式碼，但 C 檔案主程式須再加上 UART 中斷服務程式，當微控制器收到電腦傳送過來的資料後，UART_IRQHandler()會回傳電腦"#"符號並發布信號量號碼 10 去啟動 task0。

```
void UART_IRQHandler(void)
{
.....
.....
        sendByte('#');          //#符號是標示中斷服務程式所執行
        postSemOS(10);          //就緒 task0 而顯示'0'
.....
.....
}
```

(三)執行結果

圖 2 是中斷與事件功能之實驗結果，符號"#"代表是由 UART_IRQHandler()所執行，之後會額外顯示 a0 代表就緒 task0，#與 a0 不必然會連續，因為 UART_IRQHandler()有可能是中斷了正在執行的其他 task，待 UART_IRQHandler()執行結束後，CPU 會繼續執行被中斷的其他 task 後才會去執行 task0 而顯示 a0。

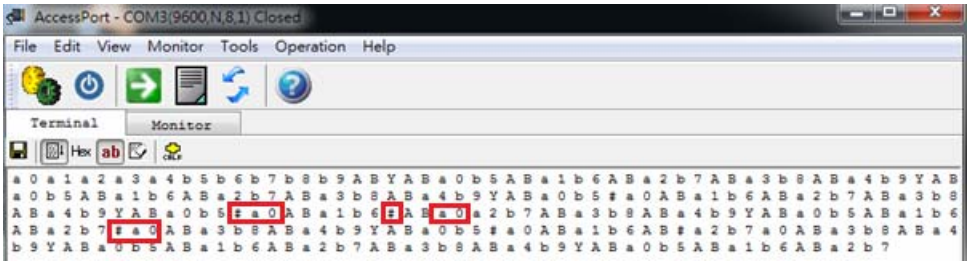

圖 2 中斷與事件功能之實驗結果

三、訊息郵箱

展示不同資料形態(int, char, double, unsigned int)的訊息互相傳遞。

(一)SIZE.h

```
#define     TASKSIZE            (int) 5
#define     PADDING             (char) 18
#define     PENDSIZE            (int) 4
#define     MEMORYPOOLBYTES     (int) 0
#define     QSIZE               (int) 0
#define     QBYTES              (int) 0
#define     FLAGSIZE            (int) 0
#define     MUTEXSIZE           (int) 0
```

(二)任務程式碼

```
    int     box0=0;
    int     box1=1;
    int     box2=2;
    int     box3=3;


    int             mail1[]={1, 2, 3};
    char            mail2[]={'4', '5', '6'};
    double          mail3[]={1.7, 1.8, 1.9};
    unsigned int    mail4[3];

void task0(void)
{
    int i;
    int number[]={box0, -1};
    unsigned int*readAddress;
    while(1)
    {
        sendByte('0');
        pendMailOS(number, TIMEOUT_INFINITE);
        readAddress=readMailOS();
        for(i=0; i<3; i++)          //0xaaaaaaaa 0xbbbbbbbb 0xcccccccc
        {
            print32bits((unsigned int)*(readAddress+i));
        }
        postMailOS(box1, mail1);
    }
}

void task1(void)
{
    int i;
    int number[]={box1, -1};
    int*readAddress;
    while(1)
    {
```

```
            sendByte('1');
            pendMailOS(number, TIMEOUT_INFINITE);
            readAddress=readMailOS();
            for(i=0; i<3; i++)              //1 2 3
            {
                print32bits((unsigned int)*(readAddress+i));
            }
            postMailOS(box2, mail2);
        }
    }

    void task2(void)
    {
        int i;
        int number[]={box2, -1};
        char*readAddress;
        while(1)
        {
            sendByte('2');
            pendMailOS(number, TIMEOUT_INFINITE);
            readAddress = readMailOS();
            for(i=0; i<3; i++)              //‘4’=0x34, ‘5’=0x35, ‘6’=0x36
            {
                print32bits((unsigned int)*(readAddress+i));
            }
            postMailOS(box3, mail3);
        }
    }

    void task3(void)
    {
        int i;
        int number[]={box3, -1};
        double*readAddress;
        double d;
        while(1)
        {
            sendByte('3');
```

```
            pendMailOS(number, TIMEOUT_INFINITE);
            readAddress=readMailOS();
            for(i=0; i<3; i++)              //7 8 9
            {
                d=*(readAddress+i)-1.0;
                d*=10.0;
                print32bits((unsigned int) (int)d );
            }
        }
    }

    void task4(void)
    {
        while(1)
        {
            sendByte('#');
            mail4[0]=0xaaaaaaaa;
            mail4[1]=0xbbbbbbbb;
            mail4[2]=0xcccccccc;
            postMailOS(box0, mail4);
            delayTimeOS(0,0,0,800);
        }
    }

    void(*taskName[])(void)={task0,task1,task2,task3, task4};
```

(二)執行結果

圖 3 是訊息郵箱功能之實驗結果，其中

task0 顯示：0xaaaaaaaa 0xbbbbbbbb 0xcccccccc

task1 顯示：1 2 3

task2 顯示：“4” “5” “6” 其 ASCII 分別為 0x34 0x35 0x36

task3 顯示：7 8 9

task4 顯示：僅代號#

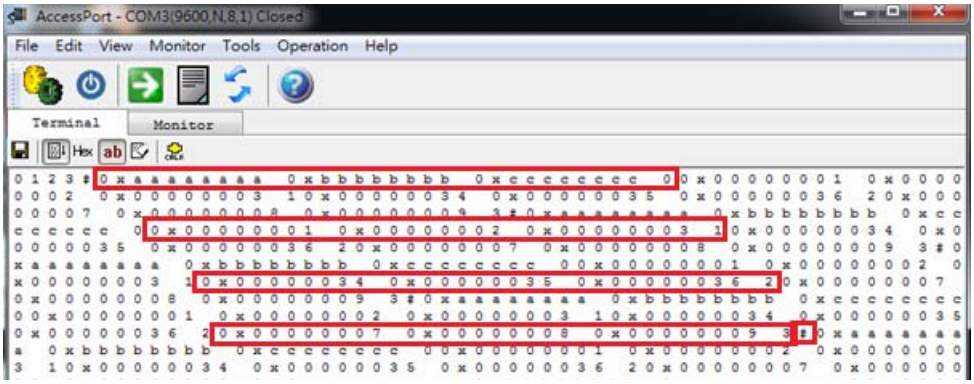

圖 3　訊息郵箱功能之實驗結果

四、互斥信號量

所有任務皆等待同一個互斥信號量(號碼 0)，以信號量來控制任務之間使用互斥信號量的順序流程，獲得信號量的任務才能夠等待互斥信號量。

(一)SIZE.h

#define	TASKSIZE	(int) 4
#define	PADDING	(char) 18
#define	PENDSIZE	(int) 7
#define	MEMORYPOOLBYTES	(int) 0
#define	QSIZE	(int) 0
#define	QBYTES	(int) 0
#define	FLAGSIZE	(int) 0
#define	MUTEXSIZE	(int) 1

(二)任務程式碼

```
void task0(void)
{
    int number[]={0, -1};
    int sem[]={0, -1};
    while(1)
    {
        pendSemOS(sem, TIMEOUT_INFINITE);
        pendMutexOS(number, TIMEOUT_INFINITE);
        sendByte('A');
        postMutexOS();
```

```
            postSemOS(1);
    }
}

void task1(void)
{
    int number[]={0, -1};
    int sem[]={1, -1};
    while(1)
    {
            pendSemOS(sem, TIMEOUT_INFINITE);
            pendMutexOS(number, TIMEOUT_INFINITE);
            sendByte('B');
            postMutexOS();
            postSemOS(2);
    }
}

void task2(void)
{
    int number[]={0, -1};
    int sem[]={2, -1};
    while(1)
    {
            pendSemOS(sem, TIMEOUT_INFINITE);
            pendMutexOS(number, TIMEOUT_INFINITE);
            sendByte('C');
            postMutexOS();
    }
}

void task3(void)
{
    int number[]={0,1, -1};
    while(1)
    {
            pendMutexOS(number, TIMEOUT_INFINITE);
            sendByte('D');
```

```
            postMutexOS();
            postSemOS(0);
            delayTickOS(6);
        }
    }

    void(*taskName[])(void)={task0, task1, task2, task3};
```

(三)執行結果

圖 4 是互斥信號量功能之實驗結果，task0 至 task2(代碼分別是 A、B、C)設計成串連式信號量啟動，而 task3 是自發性啟動(代碼 D)，task3 會去啟動 task0，task0 會去啟動 task1，task1 會去啟動 task2，任務啟動之後才可以等待互斥信號量。

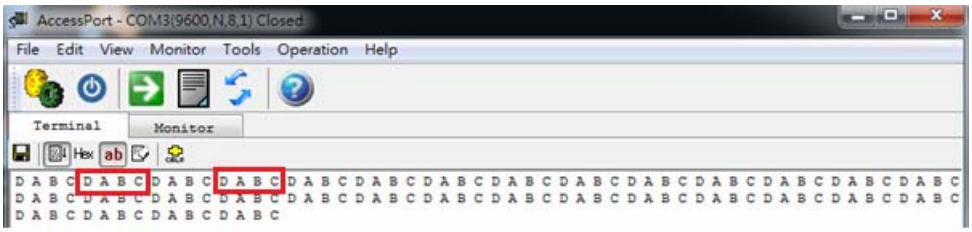

圖 4　互斥信號量功能之實驗結果

五、旗標

以旗標來控制任務的流程，以訊息顯示來證明任務流程正確無誤。

(一)SIZE.h

```
//PEND MAIL         計 task0, task6, task7 三個
//PEND FLAG         計 task1, task2, task3, task4, task5 五個

#define    TASKSIZE              (int) 9
#define    PADDING               (char) 20
#define    PENDSIZE              (int) 8
#define    MEMORYPOOLBYTES       (int) 0
#define    QSIZE                 (int) 0
#define    QBYTES                (int) 0
#define    FLAGSIZE              (int) 2
#define    MUTEXSIZE             (int) 0
```

(二)任務程式碼

```
    int box0=0;
    int box1=1;
    int box2=2;

    int flag0=0;
    int flag =1;
    unsigned int postFlag0=0x000000ff;          //set flag0
    unsigned int postFlag1=0x0000ff00;          //set flag1
    unsigned int postFlag2=0x000000ee;          //flag0 is cleared to 0x11
    unsigned int postFlag3=0x00008c00;          //flag1 is clear to 0x7300
    unsigned int postFlag4=0x00000001;          //flag0 is cleared from 0x11 to 0x10

    unsigned int pendFlag1=0x00000008;          //ANY flag0
    unsigned int pendFlag2=0x00000400;          //ANY flag1
    unsigned int pendFlag3=0x00000011;          //ALL flag0
    unsigned int pendFlag4=0x00007000;          //ALL flag1
    unsigned int pendFlag5=0x00000010;          //ALL flag0

    unsigned int writeEvent[3];
    unsigned int write1;

void task0(void)
{
   int i;
   int number[] = { box0 , -1};
   unsigned int *readAddress;
   while(1)
   {
        sendByte('0');
        pendMailOS(number, TIMEOUT_INFINITE);
        readAddress=readMailOS();
        for(i=0; i<3;i++)
        {
            print32bits((unsigned int)*(readAddress+i));
        }
        postFlagOS(flag0,postFlag0 , FLAG_SET);
   }
}
```

```
void task1(void)
{
    int number[]={flag0, -1};
    while(1)
    {
          sendByte('1');
         pendFlagOS(number, &pendFlag1, FLAG_MATCH_ANY, TIMEOUT_INFINITE);
          postFlagOS(flag1,postFlag1, FLAG_SET);
    }
}

void task2(void)
{
    int number[]={flag1, -1};
    while(1)
    {
    sendByte('2');
    pendFlagOS(number, &pendFlag2, FLAG_MATCH_ANY, TIMEOUT_INFINITE);
    postFlagOS(flag0,postFlag2 , FLAG_CLEAR);
    }
}

void task3(void)
{
    int number[]={flag0, -1};
    while(1)
    {
          sendByte('3');
          pendFlagOS(number, &pendFlag3, FLAG_MATCH_ALL, TIMEOUT_INFINITE);
          postFlagOS(flag1,postFlag3 , FLAG_CLEAR);
    }
}

void task4(void)
{
    int number[]={flag1, -1};
    while(1)
    {
```

```
        sendByte('4');
        pendFlagOS(number, &pendFlag4, FLAG_MATCH_ALL, TIMEOUT_INFINITE);
        postFlagOS(flag0,postFlag4, FLAG_CLEAR);
    }
}

void task5(void)
{
    int number[]={flag0, -1};
    while(1)
    {
        sendByte('5');
        pendFlagOS(number, &pendFlag5, FLAG_MATCH_ALL, TIMEOUT_INFINITE);
        write1=0xffffffff;
        postMailOS(box1, &write1);
    }
}

void task6(void)
{
    int number[]={box1, -1};
    unsigned int *readAddress;
    while(1)
    {
        sendByte('6');
        endMailOS(number, TIMEOUT_INFINITE);
        readAddress = readMailOS();
        print32bits(*readAddress);
        postMailOS(box2, 0x0);      //don't send message
    }
}

void task7(void)
{
    int number[]={box2, -1};
    while(1)
    {
        sendByte('7');
```

```
            pendMailOS(number, TIMEOUT_INFINITE);
        }
    }

    void task8(void)
    {
        while(1)
        {
            sendByte('8');
            writeEvent[0]=0xaaaaaaaa;
            writeEvent[1]=0xbbbbbbbb;
            writeEvent[2]=0xcccccccc;
            postMailOS(box0, writeEvent);
            delayTimeOS(0, 0, 0, 400);
        }
    }

    void(*taskName[])(void)={task0,task1,task2,task3,task4,task5,task6,task7,task8};
```

(三)執行結果

圖 5 是旗標功能之實驗結果，main()函數執行後電腦螢幕上將首先會輸出 9 個 task 的代碼 0 1 2 3 4 5 6 7 8，隨後 task0 被 task8 就緒而執行顯示 3 單位的接收訊息 0xaaaaaaaa、0xbbbbbbbb 與 0xcccccccc(由 task8 所傳送)，該 3 單位訊息顯示完畢後會依序顯示六個 task 執行時所傳送給電腦的代碼 012345，當 task6 被 task5 所就緒而執行，task6 將會顯示 1 單位的接收訊息 0xffffffff，之後將再依序顯示剩餘各 task 的代碼 678，task 之間持續依此順序同步。

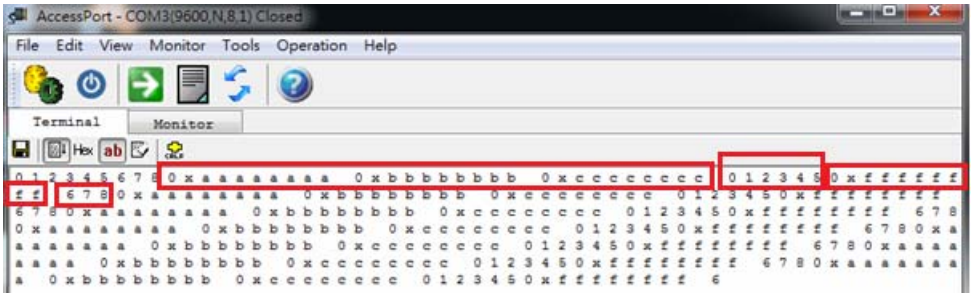

圖 5 旗標功能之實驗結果

六、記憶體管理

展示兩種 tinyOS 所提供記憶體申請的方法：getMemoryOS()與 getMemory3OS()，getMemoryOS()僅提供所申請的安全位址，getMemory3OS()可提供安全位址、邊界位址與不安全位址。若申請與歸還記憶體是在不同任務所執行，就必須注意申請與歸還的同步問題。

(一)SIZE.h

```
#define     TASKSIZE            (int) 5
#define     PADDING             (char) 12
#define     PENDSIZE            (int) 0
#define     MEMORYPOOLBYTES     (int) 1000
#define     QSIZE               (int) 0
#define     QBYTES              (int) 0
#define     FLAGSIZE            (int) 0
#define     MUTEXSIZE           (int) 0
```

(二)任務程式碼

```
void task0(void)
{
    while(1)
    {
        sendByte('A');
        pint2=getMemoryOS(100);
        if(pint2==0x0)
        {
            sendByte('@');          //error
        }
            //以 pint2 作資料存取，略程式碼
        delayTickOS(12);
    }
}

void task1(void)
{
    while(1)
    {
        sendByte('B');
```

```
            pchar2=getMemoryOS(100);
            if(pchar2==0x0)
            {
                sendByte('@');          //error
            }
            error=putMemoryOS(pchar2);
            if(error)
            {
                sendByte('#');
            }
            delayTickOS(10);
        }
    }

    void task2(void)
    {
        void*       pv[3];      //必須是三個元素的陣列空間
        int         bytes;
        while(1)
        {
            sendByte('C');
            bytes=getMemory3OS(100, pv, &dangerBytes);
            if(bytes>0)             //success
            {
                pdouble=pv[0];  //可靠安全位址
                pint=pv[1];     //可靠邊界位址，共 8 bytes 緩衝可用空間
                pchar=pv[2];    //不安全位址，任務未被強制中斷前仍可使用
                *pdouble=0.3;
                *pdouble*=10.0;     //3
                *pint=5;
                *pint+=1;           //6
                *pchar='1';         //0x31
                *(pchar+150)='2';   //0x32
                print32bits((unsigned int) (int)*pdouble);      //3
                print32bits((unsigned int)*pint);               //6
                print32bits((unsigned int)*pchar);              //0x31
                print32bits((unsigned int)*(pchar+150));        //0x32
```

```
            sendByte('$');                  //以$作顯示區隔
            print32bits((unsigned int) bytes);      //核心提供的安全空間，以 byte 為單位
            print32bits((unsigned int) dangerBytes );  //不安全空間，以 byte 為單位
        }
        delayTickOS(9);
    }
}

void task3(void)
{
    while(1)
    {
    sendByte('D');
    pfloat=getMemoryOS(100);
    if(pfloat==0x0)
    {
        sendByte('@');          //error
    }
    delayTickOS(6);
    }
}
            //所有申請的記憶體都在 task4 歸還

void task4(void)
{
    char error;
    while(1)
    {
        sendByte('Q');
        error=putMemoryOS(pdouble);
        if(error)
        {
            sendByte('#');
        }
        error=putMemoryOS(pfloat);
        if(error)
        {
            sendByte('#');
```

```
            }
            error=putMemoryOS(pint);
            if(error)
            {
                  sendByte('#');
            }
            error=putMemoryOS(pint2);
            if(error)
            {
                  sendByte('#');
            }
            error=putMemoryOS(pchar);
            if(error)
            {
                  sendByte('#');
            }
            delayTickOS(4);
      }//while
   }

   void (*taskName[])(void)={task0, task1, task2, task3, task4};
```

(三)執行結果

圖 6 是記憶體管理功能之實驗結果，task2(代碼是 C)的可靠安全位址運算 double 資料為 3，可靠邊界位址運算 int 資料為 6，不安全位址運算 char 資料為 0x31 及 0x32(遠至 150 bytes 外的不安全空間作存取)，所有任務所申請的記憶體都在 task4 歸還。任務申請 100 bytes，而核心提供安全空間是 0x70 bytes 即 112 bytes(額外多給 8 bytes 緩衝空間)，且核心尚提供在記憶體池中尚有 0x300 bytes 的空間未被使用(因任務的同步問題會動態更新)之額外資訊，任務使用無主的未被使用空間須自負風險。

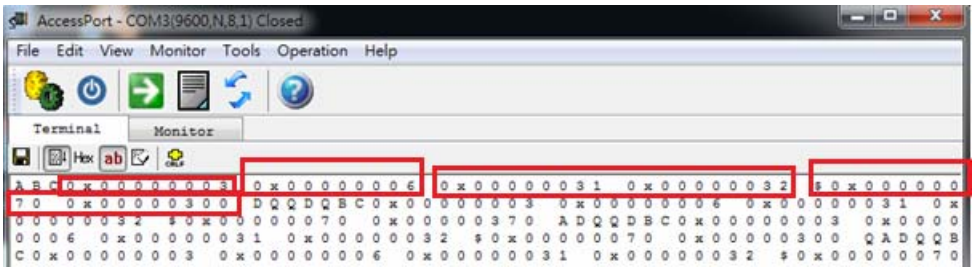

圖 6　記憶體管理功能之實驗結果

七、佇列

tinyOS 的佇列僅儲存指標位址，本例是以申請的記憶體位址來作為佇列存取的依據。

(一)SIZE.h

```
#define    TASKSIZE              (int) 3
#define    PADDING               (char) 34
#define    PENDSIZE              (int1
#define    MEMORYPOOLBYTES       (int) 1000
#define    QSIZE                 (int) 2
#define    QBYTES                (int) 96
#define    FLAGSIZE              (int) 0
#define    MUTEXSIZE             (int) 0
```

(二)任務程式碼

```
void task0(void)
{
   while(1)
   {
        sendByte('A');
        pint=getMemoryOS(4);
        print32bits((unsigned int) pint);
        *pint=x++;
        postQOS(n0, pint);
        pchar=getMemoryOS(1);
        print32bits((unsigned int) pchar);
        *pchar=(char)x;         //*pchar 會比*pint 多 1 且資料是動態更新
        postQOS(n0, pchar);
        x++;
        pdouble=getMemoryOS(8);
        print32bits((unsigned int) pdouble );
        pdouble=0.16;
        *(pdouble+1)=0.32;
        postQOS(n0, pdouble);              //pdouble 有兩筆固定資料
        delayTickOS(8);
   }
}
```

```
void task1(void)
{
    while(1)
    {
        sendByte('B');
        pfloat=getMemoryOS(4);
        print32bits((unsigned int) pfloat);
        *pfloat=y;
        y+=1.0;
        postQOS(n1, pfloat);          //*pfloat 的資料也是動態增加 1
        delayTickOS(8);
    }
}

void task2(void)
{
    int i;
    int number[]={n0, n1, -1};
    int items;
    void *retrieve[10];
    double *pdouble;
    double w1, w2;
    while(1)
    {
        sendByte('Q');
        pendQOS(number, TIMEOUT_INFINITE);
        items=readQOS(n0, retrieve);
        for(i=0; i<items; i++)
        {
            print32bits((unsigned int) retrieve[i]);   //顯示 task0 所申請的三個位址
        }
        print32bits(*(int*) retrieve[0]);
        print32bits(*(char*) retrieve[1]);
        pdouble = retrieve[2];              //pdouble 攜帶兩筆固定資料
        w1=*pdouble*100.0;                  //16
        w2=*(pdouble+1)*100.0;              //32
        print32bits( (int)w1);
```

```
            print32bits( (int)w2);
            putMemoryOS(pint);
            putMemoryOS(pchar);
            putMemoryOS(pdouble);
                //-------------------------------------------------------------
            items=readQOS(n1, retrieve);
            print32bits((unsigned int) retrieve[0]);
            print32bits((int)*(float*)retrieve[0]);
            putMemoryOS(pfloat);
    }//while
}

void (*taskName[])(void)={task0, task1, task2};
```

(三)執行結果

圖 7 是佇列功能之實驗結果，task0(代碼是 A)顯示從核心申請到的三個安全位址 0x100004a0、0x100004b0、0x100004c0，task1(代碼是 B)顯示從核心申請到的一個位址 0x100004d0，task2(代碼是 Q)會歸還該四個位址並分別取出該四位址內之資料來顯示，task2 會顯示從佇列所取出的位址確實是 task0 與 task1 存入佇列的位址，且位址內的資料確實是 task0 與 task1 先前動態計算所存入的資料，證明佇列功能正確無誤。

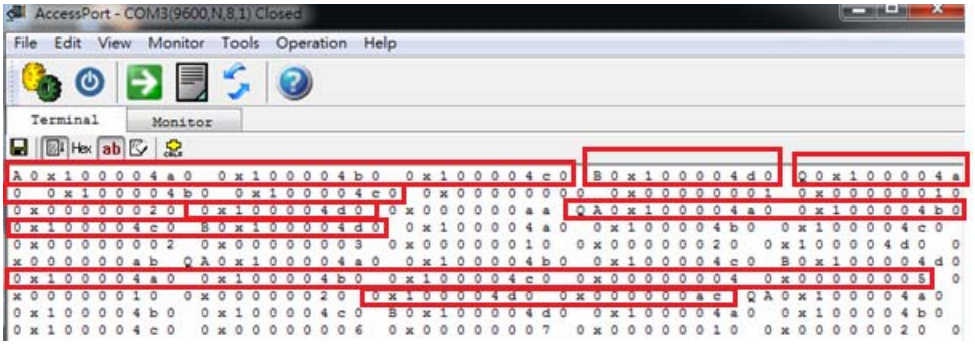

圖 7　佇列功能之實驗結果

柒、結論

tinyOS 提供一個即時作業系統所具備的完整服務功能，且繼承已發表的 tinyOS1 之精簡架構，本文所有範例都是在 4KB RAM 之 ARM Cortex-M0 微控制器順暢執行，基本上所有的 M0 微控制器教學開發板都可以順利執行，本文所展示的核心與應用程式之原始程式碼電子檔將提供給《高雄師大學報》，以程式的精簡程度與記憶體的消耗量而言，tinyOS 絕對會比當下最流行的 FreeRTOS 更適合教學工作的進行。

參考文獻

蔡長達(2019)。即時作業系統 tinyOS1：以 ARM Cortex-M0 微控制器實測。**高雄師大學報，自然科學與科技類，47**，53-72。

JEAN J. LABROSSE 著，黃文增譯(2005)。MicroC/OS-II：即時作業系統核心。新北：全華。

Joseph Yiu 著，阮聖彰、莊文維譯(2014)。ARM Cortex-M0 嵌入式系統設計入門。臺北：旗標。

Adam Heinrich (2016). *A simple context switcher for Cortex-M0 processors.* Retrieved from https://github.com/adamheinrich/os.h/tree/blog_2016_07