16.070 Introduction to Computers and Programming

26 April Recitation 10 Spring 2001

Topics

Multi-tasking (by example):
- Basic multi-tasking on the workstation
- Inter-thread communication
- Viewing and Changing Priority

Multi-tasking (by example)

Basic multi-tasking

```c
/*
*  Multithreaded example:
*
*  This program will let you spawn as many threads as
*  you like. Each thread, when spawned from main(), is
*  given a unique number. It will print its task number
*  to the screen, wait a random number of milliseconds,
*  then print, then wait, etc.
*  *
*  If you make the Sleep() a constant (say, 600), and
*  generate a small number of tasks, they will execute in
*  an orderly fashion.
*  *
*  If you generate a large number of tasks, or open netscape
*  and go to shockwave.com (or do something else that
*  distracts the CPU), you'll start to see some chaos.
*
*  This program requires the multithread library. go to the
*  Project -> Settings -> C/C++ tab and select Category "Code Generation"
*  and choose "multithreaded" under the "use runtime library:"
*  Max number of threads is 500 unless you make some changes.
*/

#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>

void MyThreadTask( void *MyID ); /* A task to be spawned */
/* multiple times */

int main() /* Task One */
{
    int i=0, Tasknum=0;
    char KeyInfo;
    /* Clear the screen.*/
    system("cls");
    printf("Generating thread... Press any key except Q to spawn a new one\n");
    /* A task to be spawned */
    /* multiple times */
```
/* Generate multiple tasks */
do {
    printf("main\n");
    fflush(stdin);
    KeyInfo = _getch();  /* Sleep until a key is pressed */
    if(KeyInfo != 'q')  /* If not "q", make a new task */
    {
        Tasknum++;
        _beginthread( MyThreadTask, 0, (void *)Tasknum );  /* spawn task */
    }
} while( KeyInfo != 'q' ); /* Loop until user presses 'q' */

void MyThreadTask( void *MyID )
{
    int num = (int)MyID;
    char space[500];
    memset(space,' ',500); /* create a string of spaces */
    space[num] = '\0'; /* "num" long, and null-terminated */

    printf(" Task %d Spawned \n", num);
    while(1) /* loop forever */
    {
        Sleep( rand()%2000 ); /* sleep a while */
        printf("\t%s<%d>\n",space,num); /* I'm alive! */
    }
}

Inter-thread communication

Mutex/Semaphore free Inter-thread communication (NOT DESIRABLE):
When two or more threads share the same resource (e.g. a global variable), multitasking might result in
undesirable resource modification. E.g. say threads A and B share resource R. A reads R and makes a
decision regarding modification of R based on its reading. Then A is swapped out and thread B is
swapped in. B modifies A. A now swaps back in and modifies R since it did not know of B’s modification
to R. Clearly this is undesirable.

#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>

/* Each task reads the following number and adds 5 to it if it is 0 and
subtracts 5 from it if it is 5 */
int Inc_Dec=0;  /* Increment Decrement */

void MyThreadTask( void *MyID );  /* A task to be spawned */
/* multiple times */

int main() /* Task One */
{
    int i=0, Tasknum=0;
    char KeyInfo;

    /* Create mutexes */
    hAvailable = CreateMutex(NULL, FALSE, NULL);  /* cleared */
}
Inter-task communication using a Mutex:
Before accessing and modifying a shared resource (like a global variable), determine if the resource is in use by another thread by checking if a mutex associated with the resource is locked. If locked, indicate to the operating system that you would like to queue for availability and wait for availability.
Once the mutex is unlocked, you have control over the resource and the operating system again locks the mutex on your behalf. If unlocked, lock mutex and modify/use resource. Unlock mutex when resource usage is complete.

/* Clear the screen.*/
system("cls");
printf("Generating thread... Press any key except Q to spawn a new one\n");

int main()
{
    int i=0, Tasknum=0;
    char KeyInfo;

    /* Create mutexes */
    hAvailable = CreateMutex(NULL, FALSE, NULL); /* cleared */

    /* Clear the screen.*/
system("cls");
printf("Generating thread... Press any key except Q to spawn a new one\n");

    /* Generate multiple tasks */
do
    { /* cleared */
        printf("main\n");
        fflush(stdin);
        KeyInfo = _getch(); /* Sleep until a key is pressed */
        if(KeyInfo != 'q') /* If not "q", make a new task */
        {
            Tasknum++;
            _beginthread( MyThreadTask, 0, (void *)Tasknum ); /* spawn task */
        }
    } while( KeyInfo != 'q' ); /* Loop until user presses 'q' */

    /* clean up handles */
    CloseHandle(hAvailable);
    return 0;
}
```c
printf("main\n");
fflush(stdin);          /* Sleep until a key is pressed */
KeyInfo = _getch();
if(KeyInfo != 'q')      /* If not "q", make a new task */
{
    Tasknum++;
    _beginthread( MyThreadTask, 0, (void *)Tasknum );   /* spawn task */
}
while( KeyInfo != 'q' );    /* Loop until user presses 'q' */

/* clean up handles */
CloseHandle(hAvailable);
return 0;
}

void MyThreadTask( void *MyID )
{
    int num = (int)MyID;
    char space[500];       /* create a string of spaces */
    int Inc_Dec_Local;
    memset(space,' ',500);   /* num" long, and null-terminated */
    space[num]='$0';

    printf(" Task %d Spawned \n", num);
    while(1)              /* loop forever */
    {
        Sleep(rand()%2000);/* sleep a while */
        WaitForSingleObject(hAvailable, INFINITE);
        Inc_Dec_Local=Inc_Dec;
        Sleep(100);
        if (Inc_Dec_Local==0)
            Inc_Dec+=5;
        else if (Inc_Dec_Local==5)
            Inc_Dec-=5;
        else Inc_Dec=10;
        printf(\"%s\<%d> Inc_Dec=%d\n\", space,num,Inc_Dec); /* I'm alive! */
        ReleaseMutex(hAvailable);  /* Release mutex */
    }
}
Viewing and Changing Priority

Pressing any key will cause the following program to spawn Task 1. Task 1 then spawns Task 3. Pressing another key results in Task 2 being spawned. Tasks 1 and 2 share a resource. Changing the relative priorities of the tasks provides and insightful gaze upon the intricacies of task priority and can highlight concepts such as priority inversion. Note the commands for viewing and changing tasks’ priorities.

```c
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>

HANDLE hAvailable;             /* Availability Mutex          */
HANDLE hThread1, hThread2, hThread3; /* Create thread handles */
/* Each task reads the following number and adds 5 to it if it is 0 and subtractions 5 from it if it is 5 */
int Inc_Dec=0;              /* Increment Decrement */

void MyThreadTask1( void ); /* A task to be spawned */
void MyThreadTask2( void ); /* A task to be spawned */
void MyThreadTask3( void ); /* A task to be spawned */
/* multiple times */

int main()               /* Task One */
{
    int i=0;
    char KeyInfo;

    /* Create mutexes */
    hAvailable = CreateMutex(NULL, FALSE, NULL); /* cleared */

    /* Clear the screen.*/
    system("cls");
    printf("Generating thread... Press any key except Q to spawn a new one\n");

    /* Generate multiple tasks */
    printf("Ready...\n");
    fflush(stdin);
    KeyInfo = _getch();        /* Sleep until a key is pressed */
    printf("\nSpawn Thread 1\n");
    hThread1=_beginthread( MyThreadTask1, 0, NULL);
    SetThreadPriority(hThread1, THREAD_PRIORITY_TIME_CRITICAL);
    fflush(stdin);
    KeyInfo = _getch();        /* Sleep until a key is pressed */
    printf("\nSpawn Thread 2\n");
    hThread2=_beginthread( MyThreadTask2, 0, NULL);
    SetThreadPriority(hThread2, THREAD_PRIORITY_BELOW_NORMAL);
    KeyInfo = _getch();        /* Sleep until a key is pressed */

    /* clean up handles */
    CloseHandle(hAvailable);
    return 0;
}

void MyThreadTask1(void)
{
    char space[500];
    int Inc_Dec_Local;
```
memset(space, ' ', 500);            /* create a string of spaces */
space[1]='\0';                    /* 1 long, and null-terminated */
/* Spawn Thread #3 from within Thread #1*/
printf("\n\nSpawn Thread 3 from within 1\n")
hThread3=_beginthread( MyThreadTask3, 0, NULL);
SetThreadPriority(hThread3, GetThreadPriority(hThread1));
//SetThreadPriority(hThread3, THREAD_PRIORITY_LOWEST);

printf("Task %d Spawned \n", 1);    /* loop forever */
{
    int i=0;
    /* check mutex before reading and altering Inc_Dec */
    WaitForSingleObject(hAvailable, INFINITE);
    //Sleep( 200 );                  /* sleep a while */
    for (i=0;i<1000000;i++);
    Inc_Dec_Local=Inc_Dec;
    Sleep(100);
    if (Inc_Dec_Local==0)
        Inc_Dec+=5;
    else if (Inc_Dec_Local==5)
        Inc_Dec-=5;
    else Inc_Dec=10;
    printf("%s<%d> Inc_Dec=%d\n",space,1,Inc_Dec);   /* I'm alive!    */
    ReleaseMutex(hAvailable);   /* Release mutex */
}
}
void MyThreadTask2(void)
{
    char space[500];
    int Inc_Dec_Local,i;
    memset(space, ' ', 500);        /* create a string of spaces */
    space[2]='\0';      /* 2 long, and null-terminated */
    printf("Task %d Spawned \n", 2);    /* loop forever */
    {
        //Sleep( 1000 );                 /* sleep a while */
        for (i=0;i<10000000000;i++);
        Inc_Dec_Local=Inc_Dec;
        Sleep(100);
        if (Inc_Dec_Local==0)
            Inc_Dec+=5;
        else if (Inc_Dec_Local==5)
            Inc_Dec-=5;
        else Inc_Dec=10;
        printf("%s<%d> Inc_Dec=%d\n",space,2,Inc_Dec);  /* I'm alive!    */
    }
}
void MyThreadTask3(void)
{
    char space[500];
int Inc_Dec_Local, i;

memset(space, ' ', 500); /* create a string of spaces */
space[3] = '\0'; /* 3 long, and null-terminated */

printf(" Task %d Spawned \n", 3);
while(1) /* loop forever */
{
    /* check mutex before reading and altering Inc_Dec */
    WaitForSingleObject(hAvailable, INFINITE);

    //Sleep( 1000 ); /* sleep a while */
    for (i=0; i<100000000; i++);
    Inc_Dec_Local = Inc_Dec;
    Sleep(100);

    if (Inc_Dec_Local == 0)
        Inc_Dec += 5;
    else if (Inc_Dec_Local == 5)
        Inc_Dec -= 5;
    else
        Inc_Dec = 10;

    printf(" \t%s<%d> Inc_Dec=%d\n", space, 3, Inc_Dec); /* I'm alive! */
    ReleaseMutex(hAvailable); /* Release mutex */
}
}