

```

/*****
/*          V I R A L L O C . C          */
/*-----*/
/* Task      : Demonstrates the effects of different COMMIT      */
/*              strategies in calling VirtualAlloc.              */
/*-----*/
/* Authors    : Michael Tischer and Bruno Jennrich              */
/* developed on : 09/03/1995                                      */
/* last update  : 09/03/1995                                      */
/*****/
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>

//==== Typedefs and Constants =====
typedef struct tagRECORD
{
    char Data[ 1 ];
} RECORD;
typedef RECORD *PRECORD;

#define ALLOC_COMPLETE 0
#define ALLOC_ONLYNEW 1
#define ALLOC_LOOKAHEAD 2

//==== Global Variables =====

long g_lLookAhead;          //Number of records that need to be pre-allocated
long g_lCommittedRecords;   // Number of committed records
long g_lMode;               // current allocation strategy
long g_lMaxRecords;         // Number of maximum records to be allocated
long g_lRecordSize;         // Size of a record

PRECORD g_pRecords = NULL; // Record array

/*****
/* PutRecord : Write new record to array      */
/*-----*/
/* Parameter:   iPos      - Array Index      */
/*              pRecord   - record to be transferred      */
/* Return value: none      */
/*****/
void PutRecord( int iPos, PRECORD pRecord )
{
    switch( g_lMode ) // Which strategy?
    {
        // Commit memory for records, from initial record to new record ----
        case ALLOC_COMPLETE:
            if( !VirtualAlloc( g_pRecords,
                              ( iPos + 1 ) * g_lRecordSize,
                              MEM_COMMIT,
                              PAGE_READWRITE ) )
            {
                printf("Commit failed!\n");
                exit( -1 );
            }
            break;

        // Commit only memory for the new record -----
        case ALLOC_ONLYNEW:
            if( !VirtualAlloc( g_pRecords + iPos * g_lRecordSize,
                              g_lRecordSize,
                              MEM_COMMIT,
                              PAGE_READWRITE ) )
            {
                printf("Commit failed!\n");
                exit( -1 );
            }
            break;

        // Always commit memory for some records in advance -----
        case ALLOC_LOOKAHEAD:
            if( g_lCommittedRecords <= iPos ) // Does the new record fit?
            {

```

```

LONG lActualCommit;

// Calculate number of records to be committed in advance -----
lActualCommit = g_lLookAhead;

// When we reach the end of the reserved memory, we have to
// make an adjustment to the records to be committed,
// because we aren't allowed to make commitments beyond the
// end of reserved memory
if( g_lCommittedRecords + g_lLookAhead > g_lMaxRecords )
    lActualCommit = g_lMaxRecords - g_lCommittedRecords;

// Commit in advance -----
if( !VirtualAlloc( g_pRecords + iPos * g_lRecordSize,
                  lActualCommit * g_lRecordSize,
                  MEM_COMMIT,
                  PAGE_READWRITE ) )
{
    printf("Commit failed!\n");
    exit( -1 );
}
// Update number of committed records -----
g_lCommittedRecords += lActualCommit;
}
break;
}

// Transfer record to array -----
memcpy( &g_pRecords[ iPos ], pRecord, g_lRecordSize );
}

/*****
/* main : Start function
/*-----
/* Parameter:      argc - Number of command line parameters
/*                argv - Parameter addresses
/* Return value: none
/*-----
/* Info: Since this application is a console application, we can
/*        call it from a DOS window.
*****/
void main( int argc, char *argv[] )
{
    PRECORD pRec;                // Pointer to dummy record
    long lTime;                  // Variable for time measurement
    int i;                       // counter

    if( argc < 4 )                // Less than 4 parameters?
    {
        printf("Call to program:\n");
        printf("%s Mode Number Size [Lookahead]\n", argv[ 0 ] );
        printf("Mode :    COMPLETE|ONLYNEW|LOOKAHEAD\n");
        printf("Number:   Number of records to be allocated\n");
        printf("Size :    Size of a record\n");
        printf("Lookahead: Number of records to be allocated in advance\n");
        _getch();
        exit( -1 );
    }
    else
    {
        if( !strcmp( argv[1], "COMPLETE" ) ) g_lMode = ALLOC_COMPLETE;
        else
        if( !strcmp( argv[1], "ONLYNEW" ) )   g_lMode = ALLOC_ONLYNEW;
        else
        if( !strcmp( argv[1], "LOOKAHEAD" ) )
        {
            g_lMode = ALLOC_LOOKAHEAD;

            // Up to now no record committed in array -----
            g_lCommittedRecords = 0;

            // If no lookahead parameters were passed, 100 records are always
            // allocated in advance
            g_lLookAhead = argc == 5 ? atol( argv[ 4 ] ) : 100;

            if( g_lLookAhead <= 0 )

```

```

    {
        printf("Lookahead size must be greater than 0!\n");
        _getch();
        exit( -1 );
    }
}
else
{
    printf("Invalid mode!\n");
    _getch();
    exit( -1 );
}

// Get number of records -----
g_lMaxRecords = atol( argv[ 2 ] );
if( g_lMaxRecords <= 0 )
{
    printf("Number must be greater than 0!\n");
    _getch();
    exit( -1 );
}

// Get size of a record -----
g_lRecordSize = atol( argv[ 3 ] );
if( g_lRecordSize <= 0 )
{
    printf("Record size must be greater than 0!\n");
    _getch();
    exit( -1 );
}
}

// Reserve memory for records -----
g_pRecords = VirtualAlloc( NULL,
                           g_lMaxRecords * g_lRecordSize,
                           MEM_RESERVE,
                           PAGE_READWRITE );

// Create dummy record -----
pRec = VirtualAlloc( NULL,
                    g_lRecordSize,
                    MEM_RESERVE | MEM_COMMIT,
                    PAGE_READWRITE );

if( g_pRecords && pRec )
{
    lTime = GetTickCount();                                // Timing begins
    for( i = 0; i < g_lMaxRecords; i++ )
    {
        PutRecord( i, pRec );                             // Store record away

        // So the user sees that something is happening -----
        if( ( i % 1000 ) == 0 )
            printf("%03d%\r", ( i * 100 ) / g_lMaxRecords );
    }
    lTime = GetTickCount() - lTime;                        // calculate elapsed time

    printf("Time[mm:ss:zht]: %02ld:%02ld:%03ld",
           ( lTime / (60 * 1000) ), // convert milliseconds to
           ( lTime / 1000 ) % 60,   // minutes, seconds and
           lTime % 1000 );         // thousandths
    _getch();
}
else
    printf("Could not allocate memory");

// Release record array and dummy record -----
if( g_pRecords )
    VirtualFree( g_pRecords,
                g_lMaxRecords * g_lRecordSize,
                MEM_DECOMMIT | MEM_RELEASE );

if( pRec )
    VirtualFree( pRec,
                g_lRecordSize,
                MEM_DECOMMIT | MEM_RELEASE );
}

```