Creating a shared memory block that can grow in size

devblogs.microsoft.com/oldnewthing/20150130-00

January 30, 2015

Raymond Chen

A little-known feature of shared memory blocks in Win32 is that it is possible to resize them, sort of.

When you create a shared memory block, you can pass the `SEC_RESERVE` flag to `CreateFileMapping`, then the size you pass to the function is treated as a maximum rather than an exact size. (Don't forget that `CreateFileMapping` is used for creating both memory-mapped files and for creating plain old shared memory. The name of the function is misleading unless you're wearing kernel-colored glasses.)

When you map this shared memory block, you are reserving address space, but no memory is committed yet. You call `VirtualAlloc` to commit memory into the shared memory block.

This means that you can create a growable shared memory block by creating an initially empty block, and then committing a small amount of memory into it. When you want to grow the block, you commit more. However, you cannot shrink the shared memory block. Once the memory is committed, it cannot be decommitted.

Here's a demonstration. Note that most error checking has been elided for expository purposes. Note also that since the memory isn't actually being shared with anybody, this program working too hard; it could have just used plain old `VirtualAlloc`. So pretend that the memory is being shared with somebody else.

```
#include <windows.h>
#include <stdio.h>

#define ONE_GIGABYTE (1024 * 1024 * 1024)
#define VIEW_SIZE (ONE_GIGABYTE / 2) // We will map half of it

void ReportMemoryPresence(void *p)
{
 MEMORY_BASIC_INFORMATION mbi;
 VirtualQuery(p, &mbi, sizeof(mbi));
 printf("Memory at %p is %s\n", p,
        (mbi.State & MEM_COMMIT) ? "committed" : "not committed");
}

void WaitForEnter()
{
 char dummy[64];
 fgets(dummy, 64, stdin);
}

int __cdecl wmain(int, wchar_t **)
{
 BYTE *pView;
 HANDLE h = CreateFileMapping(INVALID_HANDLE_VALUE, NULL,
                              PAGE_READWRITE,
                              0, VIEW_SIZE,
                              NULL);
 printf("Created the file mapping\n");
 WaitForEnter();

 pView = (BYTE*)MapViewOfFile(h, FILE_MAP_WRITE, 0, 0, VIEW_SIZE);
 printf("Mapped half of it at %p\n", pView);

 ReportMemoryPresence(pView);
 ReportMemoryPresence(pView + VIEW_SIZE - 1);
 WaitForEnter();

 return 0;
}
```

In this version, we create a one-gigabyte shared memory block with no special flags, which means that all the memory gets committed up front. When you run this program, it reports that the memory at the start and end of the mapping is present. That's because the normal mode for shared memory is to commit it all at creation.

You can watch the effect of commit by running Task Manager, going to the Performance tab, and looking at the value under *Committed*. It should jump by a gigabyte when "Created the file mapping" is printed. (For some reason, the *Commit size* in the Details pane counts the view as commitment, even though the view consists almost entirely of reserved rather than committed pages.)

Now let's add the `SEC_RESERVE` flag:

```
 HANDLE h = CreateFileMapping(INVALID_HANDLE_VALUE, NULL,
                              PAGE_READWRITE | SEC_RESERVE,
                              0, VIEW_SIZE,
                              NULL);
```

Now when you run the program, Task Manager's Committed memory does not increase. That's because we created an empty shared memory block with the *potential* to grow up to one gigabyte, but right now it is size zero. This is confirmed by the memory presence check, which reports that the memory at the start and end of the mapped view is not committed.

Okay, well, a zero-length shared memory block isn't very useful, so let's make it, say, 100 megabytes in size.

```
#define BLOCK_SIZE (100 * 1024 * 1024)

int __cdecl wmain(int, wchar_t **)
{
 BYTE *pView;
 HANDLE h = CreateFileMapping(INVALID_HANDLE_VALUE, NULL,
                              PAGE_READWRITE | SEC_RESERVE,
                              0, VIEW_SIZE,
                              NULL);
 printf("Created the file mapping\n");
 WaitForEnter();

 pView = (BYTE*)MapViewOfFile(h, FILE_MAP_WRITE, 0, 0, VIEW_SIZE);
 printf("Mapped half of it at %p\n", pView);

 ReportMemoryPresence(pView);
 ReportMemoryPresence(pView + VIEW_SIZE - 1);
 WaitForEnter();

 VirtualAlloc(pView, BLOCK_SIZE, MEM_COMMIT, PAGE_READWRITE);
 printf("Committed some of it at %p\n", pView);

 ReportMemoryPresence(pView);
 ReportMemoryPresence(pView + BLOCK_SIZE - 1);
 ReportMemoryPresence(pView + BLOCK_SIZE);
 ReportMemoryPresence(pView + VIEW_SIZE - 1);
 WaitForEnter();

 return 0;
}
```

Watch the Committed memory in Task Manager go up by 0.1 gigabytes when we commit some of it. Also observe that the memory presence checks show that we have exactly 100 megabytes of memory available; the byte at 100 megabytes + 1 is not present.

Okay, so we were able to grow the shared memory block from zero to 100 megabytes. Let's grow it again up to 200 megabytes.

```
int __cdecl wmain(int, wchar_t **)
{
 ...

 VirtualAlloc(pView + BLOCK_SIZE, BLOCK_SIZE, MEM_COMMIT, PAGE_READWRITE);
 printf("Committed some of it at %p\n", pView + BLOCK_SIZE);

 ReportMemoryPresence(pView);
 ReportMemoryPresence(pView + 2 * BLOCK_SIZE - 1);
 ReportMemoryPresence(pView + 2 * BLOCK_SIZE);
 ReportMemoryPresence(pView + VIEW_SIZE - 1);
 WaitForEnter();

 return 0;
}
```

Okay, well there you go, a growable shared memory block. If you wanted to conserve address space, you could use `MapViewOfFile` to map only the number of bytes you intend to commit, and each time you want to grow the memory block, you create a new larger view. I didn't bother with that because I'm lazy.

Bonus chatter: Another way to get the effect of growable and shrinkable shared memory blocks is to cheat and create multiple shared memory blocks, but map them right next to each other.

Bonus chatter 2: You can get sort of the effect of decommitting memory from the block by resetting it (`MEM_RESET`). The memory is still committed, but you told the memory manager that if the memory needs to be paged out, just discard it rather than writing it to disk.

Bonus chatter 3: Be aware that creating very large `SEC_RESERVE` sections can incur high commit charges for the page tables themselves. This is significantly improved in Windows 8.1, which defers committing the page tables until you actually use them.

Raymond Chen

Follow