Allocating page file space without allocating RAM

devblogs.microsoft.com/oldnewthing/20151221-00

December 21, 2015

Raymond Chen

The Windows Server 2003 Resource Kit comes with a tool called `consume.exe`, and one of the things you can ask it to do is to suck up space in your swap file. If you run the program in that mode, you see that it manages to do its job without consuming physical memory. (The amount of Free Pages in Task Manager remains constant.) How is that possible?

Recall that memory allocation in user-mode is virtual (unless you are using the special functions that specifically allocate physical memory). When you "allocate memory", you are really just allocating commit. Commit is the promise to produce memory upon access, but until you actually access it, there is no requirement that memory be produced.

The `consume.exe` does its work by simply calling `VirtualAlloc` to commit pages, but never accessing them. Since the program never accesses the pages, the system never needs to find physical RAM to back them. It can just leave the pages committed in the page file.

Actually, it's even more virtual than this: There is no specific page in the page file with your name on it. The operating system only needs to make sure that it can produce the memory on demand. Since you never wrote anything, the page starts out filled with zeros, and there is no need to find a spot in the page file and physically fill it with zero. The memory manager can just tag your commit with "If anybody asks for this memory, just give them a page filled with zero." Of course, if you actually write to the memory, then the memory manager needs to be sure that there's space in the page file to write your modifications, so that they can be read back on demand.

(In theory, the operating system could check if the memory about to be written to the page file is filled with zeroes, and if so, then don't actually write anything but merely edit the tag on the commit to say, "If anybody asks for this memory, just give them a page filled with zero." In practice, I don't think this happens because the cost of doing this extra work for every page doesn't exceed the benefit of saved I/O on the rare cases it actually detects a page that has been modified to be full of zeros.)

The catch for the `consume.exe` program is that it will eventually run out of virtual address space. At the time `consume.exe` was written, this was probably not an issue because computers didn't have that much memory, and page files were typically less than 2GB. But now, with 1GB of RAM being typical on entry-level machines, the prospect of exhausting the virtual address space on a 32-bit system becomes more likely.

Today's Little Program is a version of `consume.exe` that can soak up space in the page file in excess of 2GB, even on 32-bit systems.

The trick is to allocate commit without consuming address space. We actually saw how to do this a long time ago, when we discussed how a 32-bit application can allocate more than 4GB of memory: Use `CreateFileMapping` to create commit without mapping it.

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

#define GIGABYTE (1024 * 1024 * 1024)

int __cdecl main(int argc, char **argv)
{
 if (argc >= 2) {
  int gigabytes = atoi(argv[1]);
  for (int i = 0; i < gigabytes; i++) {
   printf("Allocating another gigabyte...\n");
   HANDLE h = CreateFileMapping(INVALID_HANDLE_VALUE, 0,
                                PAGE_READWRITE, 0, GIGABYTE, NULL);
   if (h != NULL) printf("Allocated\n");
   Sleep(1000);
  }
  printf("Done. Sleeping 30 seconds before exiting.\n");
  Sleep(30 * 1000);
 }
 return 0;
}
```

Raymond Chen

Follow