Writing Assignment 3 - Memory Management

Isaac Stallcup

CS 444

Spring 2017

1 WINDOWS

Please note: all Windows material is from [1] or a lecture given by Professor McGrath on June 01. This statement is designed to replace a citation [1] after every single sentence for reading pleasure, but feel free to mentally insert one before each period in the following sections on Windows. They have been inserted for you in some cases.

In Windows, 32-bit processes have a 2GB virtual memory space; this is increased to 314GB on ”hybrid” 32/64-bit windows. For pure 64-bit windows processes have 8192GB of virtual memory space. This number is chosen arbitrarily and could possibly be increased. Processes are also more a container for threads than a singular process.

Windows supports several other generic services in regards to memory management:

- Address mapping
- Paging
- Memory mapped files
- Copy-on-write memory
- Direct physical memory allocation and use

Windows also maintains a ”working set”, the set of pages physically present in memory at any one time. Other than this working set there are five essential components to Windows' memory management structure: the working set manger, the process and stack swapper, the modified page writer, the mapped page writer, and the zero page thread. The windows system is fully reentrant and is capable of finely grained locking, the result of Windows' broad architecture support.

1.1 Pages

There are two page sizes in windows to choose from; large page sand small pages. Large pages are 2MB in size, and small pages are 4KB in size. Pages can be of three types, each type controlling access to the page. Reserved pages have been requested by one page or another, but not used yet. They are set to be private. Committed pages are being used, and are therefore private. They also by default have a valid mapping to them, which is fortunate as they are (as mentioned) being used. Sharable pages can be mapped to and are resident, but not private.

2 FREEBSD

As with the Windows section, this is based on information from [2] and Kevin's lecture on the first of June, 2017. FreeBSD implements memory management in much the same way that Linux does, except for the way it handles virtual memory. FreeBSD has a stack section instead of a BSS section like Linux does. The stack section keeps track of a process' run time stack.

3 Comparisons to Linux

3.1 Windows

Linux and Windows both have copy-on-write memory, and maintain a working set (or equivalent). Processes in Windows are essentially containers for threads, whereas in Linux they retain some of their process-oriented properties.

3.2 FreeBSD

Linux supports copy-on-write memory by default, while FreeBSD does not, and only performs copy-on-write if a process is allowed to keep a copy of the page. Additionally, as mentioned, FreeBSD does not have the block started by section, instead having a stack section.

References

[1] A. I. Mark Russinovich, David A. Solomon, *Windows Internals, Sixth Edition, Part 2*.

[2] R. N. W. Marshall Kirk McKusick, George V. Neville-Neil, *The Design and Implementation of the FreeBSD Operating System*.