Module 6 –Virtual Memory

This module discusses the steps required to implement virtual memory.

Virtual Memory allows us to use more memory than is physically available. The way it does this is by swapping pages to and from disk as necessary.

Example:

Given a virtual memory size V_m, a physical memory size R_m where $V_m > R_m$ and a fixed process size P_m where $\sum P_m > V_m$ how do we manage memory so that the right pages are in memory when we need to access them.

We will change our Memory Management unit to use virtual memory rather than physical. This involves additional steps:

Add two new flags to your page object:

Flag	Meaning
IsValid	Is the page in physical memory now?
IsDirty	Has the physical page been

1. Add a flag to designate each page as valid (in physical memory now) or invalid (not currently in physical memory)
2. Add a flag to designate each page as dirty or clean.

When the operating system starts:

1. Create a virtual page table, in addition to your physical one.
2. In your virtual page table, mark each page of physical memory as valid.
3. Mark all pages as clean

When bringing programs in memory, insert them in virtual memory pages.

1. If there are free, valid pages, insert them.
2. Always record the process id.
3. Your process page table should now refer to virtual memory pages, not physical ones.
4. If there are no free, valid pages, swap out the least recently used page to disk.
5. Swap in the new page

When a memory address is accessed in a process

1. If the page is in physical memory (IsValid = true), use it as we normally would.
    a. If it is a write operation, mark the page as dirty.
2. If the page is not in physical memory (IsValid = false):
    a. Find the least recently used page.
    b. Write it to the hard drive.
    c. Read in the page file from disk.

3. Update appropriate metrics.

Things to consider:

1. How do you avoid thrashing? (constantly swapping pages).
2. If a page isn't marked dirty, how can you improve performance?