

Very Brief Introduction to Virtual Memory

ICS312 - Spring 2009
Machine-Level and Systems Programming

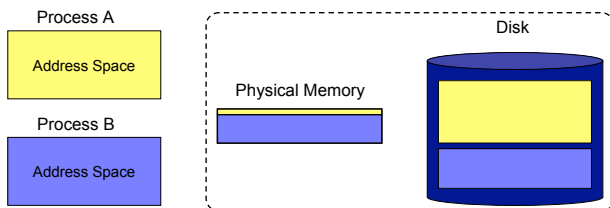
Henri Casanova (henric@hawaii.edu)

Virtual Memory

- The topic of virtual memory is at the frontier between **computer architecture** and **operating systems**
- **Virtual memory**: the technique by which the disk is used to provide the illusion of a much larger main memory
- Multiple processes run on a computer at the same time
 - This wasn't always the case
- Each process has an **address space**
 - The set of all logical addresses that the process may reference
- The sum of the address space sizes is typically much larger than main memory
- But only smaller parts of these address spaces are active at a time
 - The same old locality principle
- Therefore, virtual memory is feasible
 - **The memory acts like a cache for the disk**

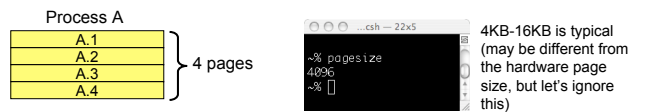
Sharing

- Consider a computer with 1GB RAM
- Consider two processes, A and B, with 2GB address spaces
- Both processes can have part of their address space actually in RAM
 - e.g., 2GB for P1 and .8GB for P2
- Whenever a process references a part of its address space that is not in RAM, that part must be brought into RAM, and some other part of some (perhaps other) address space must be evicted
 - Just like a cache, this needs a replacement policy



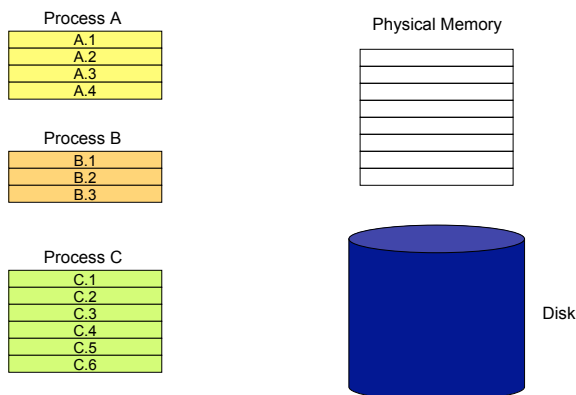
Memory Pages

- Each process' address space is split into **memory pages**
 - When we looked at caching, the memory was split into memory blocks
 - Now for this level of caching we have a higher "granularity", with memory pages

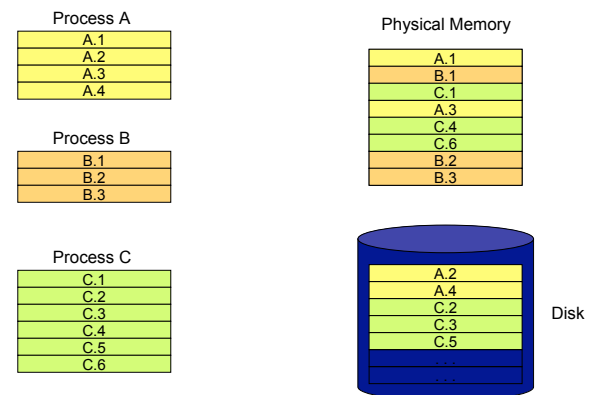


- Each process' address space size is a multiple of the page size
 - On most Linux system, a multiple of 4KB
- The page size has an impact on performance
 - Just like caches with different block sizes have different performance

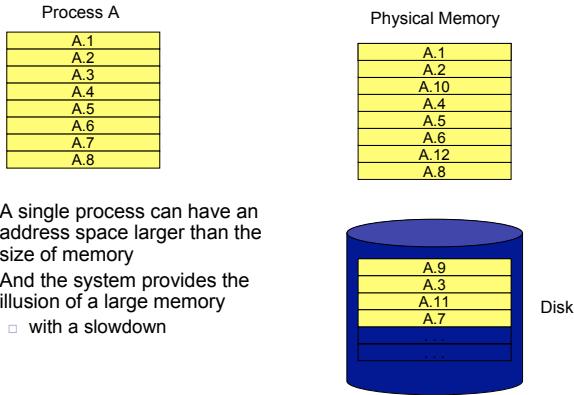
Example



Example: Memory Snapshot

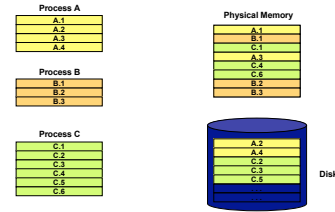


Example: Memory Snapshot



- A single process can have an address space larger than the size of memory
- And the system provides the illusion of a large memory
 - with a slowdown

Memory Protection

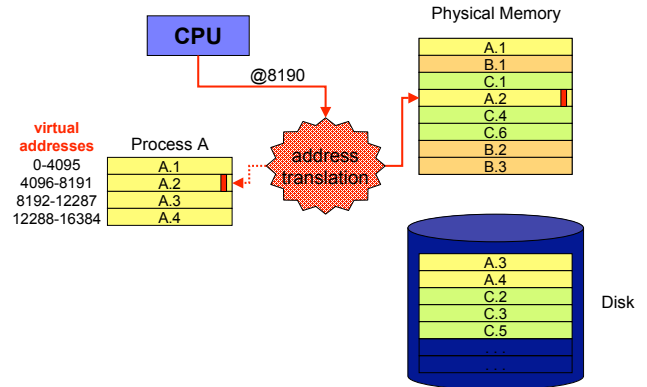


- Virtual memory ensures **protection**
- Process A cannot read/write into the memory of process B
- As we're going to see, this is easily enforced by the virtual memory address translation scheme

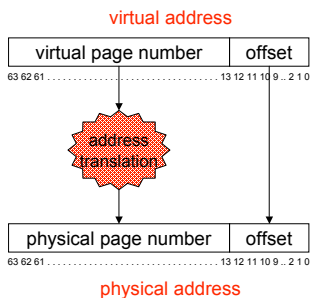
Address Translation

- **Question:** since we do not know ahead of time which processes will be in memory at the same time, how can we share/protect memory?
 - In fact, before virtual memory, programmers used to deal with this by hand and figure out in which part of physical memory each part of the address space would have to fit!
- **Answer:** provide another level of addressing
- A process can run in any physical memory location
- Each process has its own address space, which is independent from physical locations
- Addresses in this process' address space are called virtual addresses
- Virtual addresses are **translated** into physical addresses
- Advantages
 - The program doesn't care about physical memory
 - The CPU works with "nice" virtual addresses
 - A program can be loaded anywhere in memory transparently

Address Translation

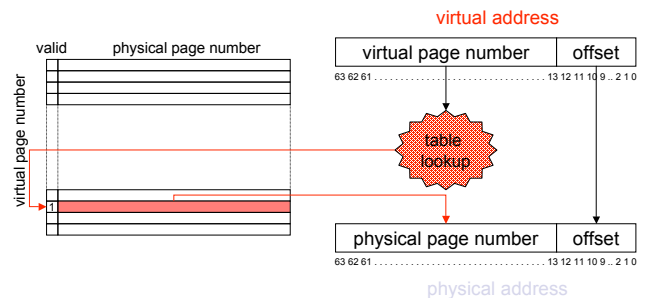


Address Translation



- **Question:** how do we do the translation?
- **Answer:** with a **page table**

Page Table



Page Table

- Each process has its own page table
 - Which provides protection by the O/S
- The page table is kept in RAM, and pointed to by a special register
- The O/S is responsible for updating page tables, and to make sure that the correct page table is used for each process
- The page table has one entry for each page
- Page table size is easy to compute
 - Example:
 - 32-bit addresses
 - 4KB (2^{12}) page size
 - Therefore, 2^{20} possible virtual pages and 20-bit page numbers
 - Each page table entry is $1 + 20 = 21$ bits, say stored in $32 = 2^5$ bits
 - Total size = 2^{20+5} bits = 4 MB
 - There are clever schemes to reduce the size of the page table

Page (re)placement

- When the memory is full and we need to load a new page from disk, one page must be evicted from the memory and saved to disk
 - We don't want to lose any page!
- A popular option for this is LRU (Least Recently Used)
 - Evict the page that was referenced the least recently
 - Likely won't be needed soon due to locality of reference

What is a process

- We typically mean that a **process** is a running program
- A better definition is that a process is the *state* of a running program
- Now we have a good definition of the **state of a running program**
 - A program counter
 - A page table
 - Register values
- This is the state that the O/S manipulates
 - saving
 - restoring

Page Faults

- A miss in the page table is called a **page fault**
- When the "valid" bit is not set to 1, then the page must be brought in from disk, possibly replacing another page in memory
- The part of the disk on which memory pages are kept is called the **swap space**
- The O/S has data about the location of each page on disk
 - May be part of the page table
- When a process spends a lot of time paging in and out from/to disk, on says "it's swapping"
 - Which is never a good thing

Conclusion

- Virtual memory is something that we now take for granted
 - There are more things we could talk about regarding virtual memory, but they belong in an O/S course or a computer architecture course
 - e.g., the TLB
- Caching is a general idea that can be applied over and over
- The virtual memory provides two things
 - protection
 - the illusion of a larger memory
 - Which comes at a high performance cost if used constantly