Paging

This is all augmented by the idea of *paging*

Paging

This is all augmented by the idea of *paging*

Paging is similar to swapping, but simpler in concept

Paging

This is all augmented by the idea of *paging*

Paging is similar to swapping, but simpler in concept

And much harder in the hardware required

Paging

This is all augmented by the idea of *paging*

Paging is similar to swapping, but simpler in concept

And much harder in the hardware required

To describe paging we must first go back to pages

Memory

Virtual Memory

A big problem is memory fragmentation due to the irregular sizes of processes/partitions

Memory

Virtual Memory

A big problem is memory fragmentation due to the irregular sizes of processes/partitions

So to fix this we chop everything up into equally sized chunks

Memory

Virtual Memory

A big problem is memory fragmentation due to the irregular sizes of processes/partitions

So to fix this we chop everything up into equally sized chunks

Recall (from memory protection) a *page* is just a contiguous area of memory: e.g., 4096 bytes

Memory

Virtual Memory

A big problem is memory fragmentation due to the irregular sizes of processes/partitions

So to fix this we chop everything up into equally sized chunks

Recall (from memory protection) a *page* is just a contiguous area of memory: e.g., 4096 bytes

Hardware is designed so copying pages in and out of memory from disk is as efficient as possible

Memory

Virtual Memory

Next, we introduce *virtual* vs. *physical* addresses

Memory

Virtual Memory

Next, we introduce *virtual* vs. *physical* addresses

A physical address is what we are used to, just a numbering of the actual bytes in the system from 0 to n

Memory

Virtual Memory

Next, we introduce *virtual* vs. *physical* addresses

A physical address is what we are used to, just a numbering of the actual bytes in the system from 0 to n

A virtual address is a per-process fictional address

Memory

Virtual Memory

Next, we introduce *virtual* vs. *physical* addresses

A physical address is what we are used to, just a numbering of the actual bytes in the system from 0 to n

A virtual address is a per-process fictional address

The user process sees only the virtual addresses: the system will translate them on the fly into physical addresses

Memory

Virtual Memory

The OS has tables, one per process, called *page tables*, that contains the virtual-physical address mappings for each page in each process

Memory

Virtual Memory

The OS has tables, one per process, called *page tables*, that contains the virtual-physical address mappings for each page in each process

For example, with a page size of 4096 bytes, address 12298 is 10 bytes from the start of page 3: $12298 = 3 \times 4096 + 10$

Memory

Virtual Memory

The OS has tables, one per process, called *page tables*, that contains the virtual-physical address mappings for each page in each process

For example, with a page size of 4096 bytes, address 12298 is 10 bytes from the start of page 3: $12298 = 3 \times 4096 + 10$

Under the entry for page 3 in the page table for this process we might find the number 7, meaning physical page 7

Memory

Virtual Memory

The OS has tables, one per process, called *page tables*, that contains the virtual-physical address mappings for each page in each process

For example, with a page size of 4096 bytes, address 12298 is 10 bytes from the start of page 3: $12298 = 3 \times 4096 + 10$

Under the entry for page 3 in the page table for this process we might find the number 7, meaning physical page 7

So virtual address 12298 **in this process** refers to physical byte $7 \times 4096 + 10 = 28682$

Memory

Virtual Memory

In another process, virtual page 3 could be mapped to physical page 42

Memory

Virtual Memory

In another process, virtual page 3 could be mapped to physical page 42

And then the same virtual address 12298 **in this process** refers to physical byte $42 \times 4096 + 10 = 172042$

Memory

Virtual Memory

In another process, virtual page 3 could be mapped to physical page 42

And then the same virtual address 12298 **in this process** refers to physical byte $42 \times 4096 + 10 = 172042$

The *same* virtual address in different processes is mapped to *different* physical addresses

Memory

Virtual Memory

In another process, virtual page 3 could be mapped to physical page 42

And then the same virtual address 12298 **in this process** refers to physical byte $42 \times 4096 + 10 = 172042$

The *same* virtual address in different processes is mapped to *different* physical addresses

We use pages, of course, to make this translation manageable

Memory

Virtual Memory

The table only contains entries for pages that are actually in use by that process: this keeps the tables to a reasonable size

V page	P page
3	7
4	9123
5	121
10	1232
etc.	

Memory

Virtual Memory

The table only contains entries for pages that are actually in use by that process: this keeps the tables to a reasonable size

V page	P page
3	7
4	9123
5	121
10	1232
etc.	

Note: page tables contain page *mappings*, not pages

Memory

Virtual Memory

The table only contains entries for pages that are actually in use by that process: this keeps the tables to a reasonable size

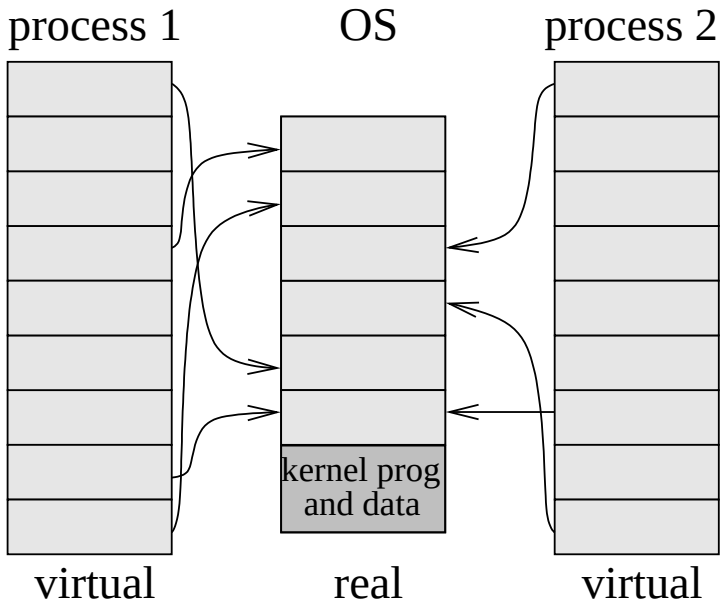
V page	P page
3	7
4	9123
5	121
10	1232
etc.	

Note: page tables contain page *mappings*, not pages

Note: though still called “tables”, in modern OSs they are likely to be more sophisticated datastructures, such as trees

Memory

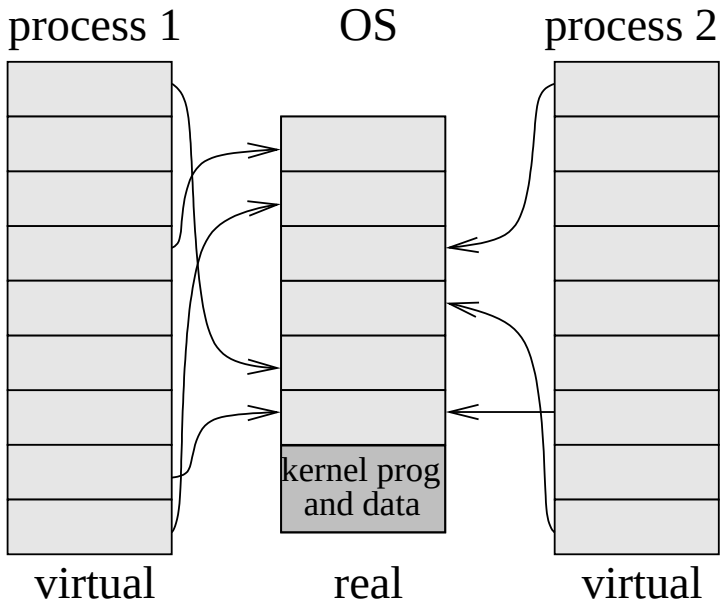
Virtual Memory



Every process gets its own complete and separate address space, mapped into the physical address space

Memory

Virtual Memory



Every process gets its own complete and separate address space, mapped into the physical address space

Even for the same userid: this is usually what you want, protection of one process from another