Memory

Virtual Memory

Copy on Write

Different processes can easily share data as long as they don't try to update it

Copy on Write

Different processes can easily share data as long as they don't try to update it

Some data is read-only (e.g., a document containing an exam paper), so this could be stored in a page marked read-only, and this can be safely shared

Memory

Virtual Memory

Copy on Write

Different processes can easily share data as long as they don't try to update it

Some data is read-only (e.g., a document containing an exam paper), so this could be stored in a page marked read-only, and this can be safely shared

Other data you *do* want to update (e.g., a document containing an exam answer template)

Memory

Virtual Memory

Copy on Write

Different processes can easily share data as long as they don't try to update it

Some data is read-only (e.g., a document containing an exam paper), so this could be stored in a page marked read-only, and this can be safely shared

Other data you *do* want to update (e.g., a document containing an exam answer template)

Such pages can be marked with another flag: *copy on write* — again, as long as the hardware supports this

Memory

Virtual Memory

With copy on write, a page is shared up until a process tries to modify it

Memory

Virtual Memory

With copy on write, a page is shared up until a process tries to modify it

At this point a page fault occurs and the OS takes over

Memory

Virtual Memory

With copy on write, a page is shared up until a process tries to modify it

At this point a page fault occurs and the OS takes over

It makes a physical copy of the page and changes the page table and TLB for that process to point at the new copy

Memory

Virtual Memory

With copy on write, a page is shared up until a process tries to modify it

At this point a page fault occurs and the OS takes over

It makes a physical copy of the page and changes the page table and TLB for that process to point at the new copy

The write can then proceed on the private, unshared copy

Memory

Virtual Memory

With copy on write, a page is shared up until a process tries to modify it

At this point a page fault occurs and the OS takes over

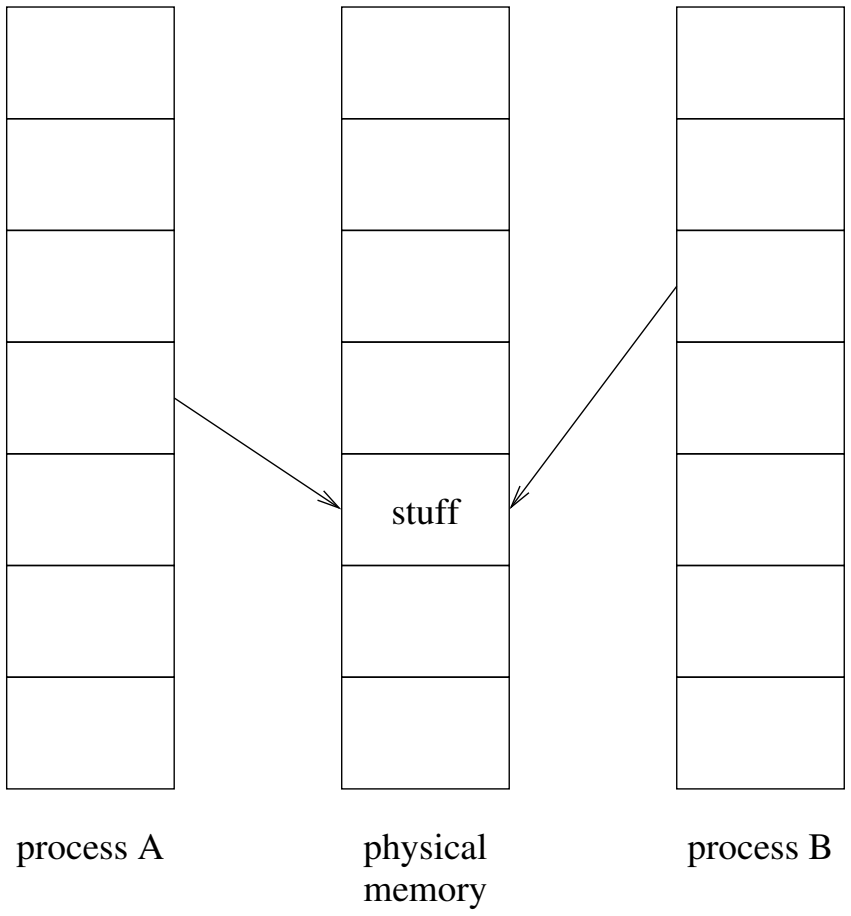
It makes a physical copy of the page and changes the page table and TLB for that process to point at the new copy

The write can then proceed on the private, unshared copy

Other processes still see the original, unmodified data

Memory

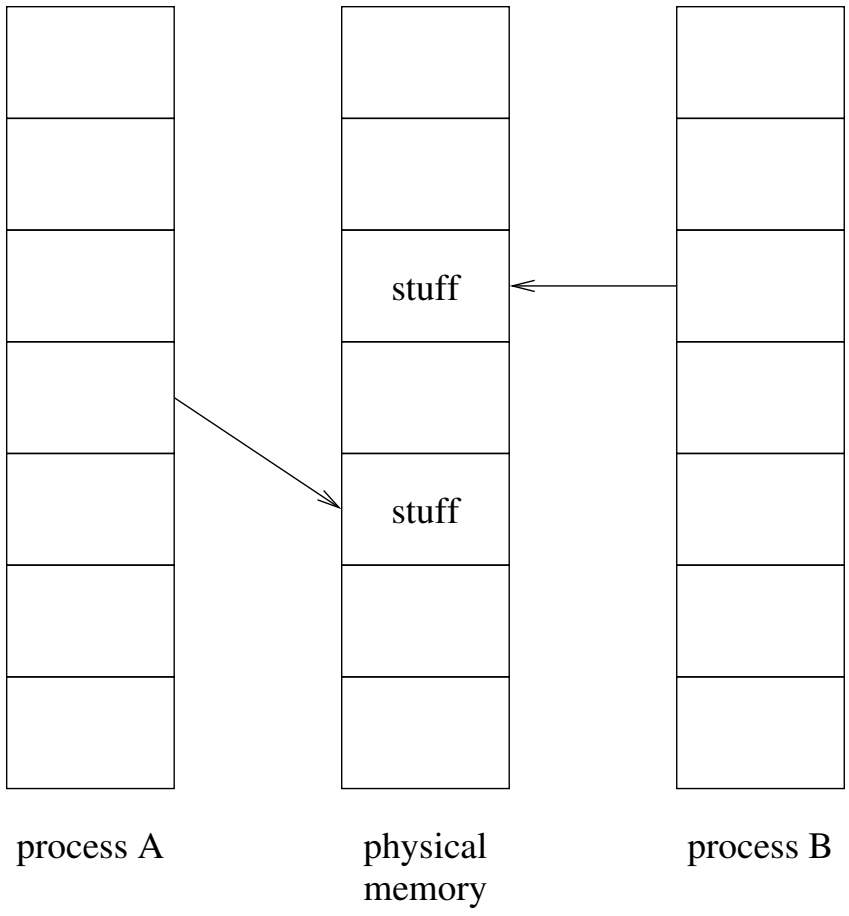
Virtual Memory



Some data is shared; process B tries to update the data

Memory

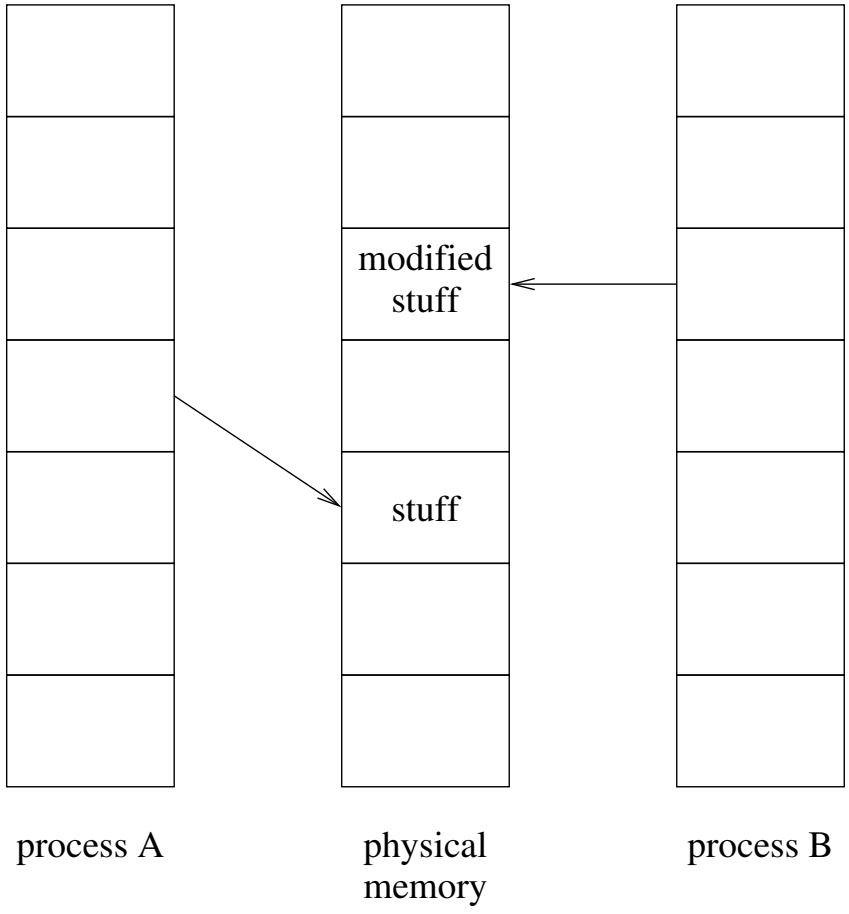
Virtual Memory



The OS takes over; copies the data to a new page; updates B's page mappings

Memory

Virtual Memory



The update continues and B modifies its own copy of the data

Memory

Virtual Memory

This works really well for when a majority of data is shared with only a few changes here and there

Memory

Virtual Memory

This works really well for when a majority of data is shared with only a few changes here and there

And it only uses an extra amount of memory proportional to the size of the changes

Memory

Virtual Memory

This works really well for when a majority of data is shared with only a few changes here and there

And it only uses an extra amount of memory proportional to the size of the changes

So another reduction in memory use, page faults and so on

Memory

Virtual Memory

This is excellent, but comes at the cost of extra complexity as the OS now has to track if a shared page has already been loaded and where it is physically

Memory

Virtual Memory

This is excellent, but comes at the cost of extra complexity as the OS now has to track if a shared page has already been loaded and where it is physically

And complexity in swapping as now it has to track which processes are using a page and it can't swap a page until no-one is using it

Memory

Virtual Memory

This is excellent, but comes at the cost of extra complexity as the OS now has to track if a shared page has already been loaded and where it is physically

And complexity in swapping as now it has to track which processes are using a page and it can't swap a page until no-one is using it

But this is offset by the fact we will need to swap less as we are using memory more efficiently

Other Tricks

OSs often keep a page full of zeros

Other Tricks

OSs often keep a page full of zeros

If a process asks for a big block of zeroed memory, the OS will supply the appropriate number of virtual pages, all pointing at the single zeroed physical page: much faster than allocating and clearing out a load of physical memory

Other Tricks

OSs often keep a page full of zeros

If a process asks for a big block of zeroed memory, the OS will supply the appropriate number of virtual pages, all pointing at the single zeroed physical page: much faster than allocating and clearing out a load of physical memory

If the process writes to that block, the OS does a copy-on-write shuffle behind the scenes, allocating and clearing a new writable page

Other Tricks

OSs often keep a page full of zeros

If a process asks for a big block of zeroed memory, the OS will supply the appropriate number of virtual pages, all pointing at the single zeroed physical page: much faster than allocating and clearing out a load of physical memory

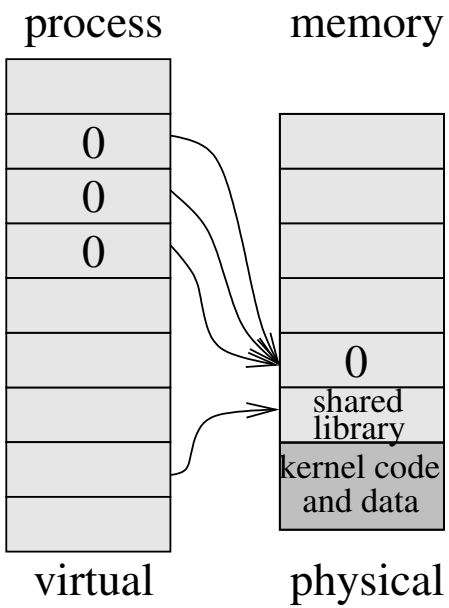
If the process writes to that block, the OS does a copy-on-write shuffle behind the scenes, allocating and clearing a new writable page

Thus only allocating and clearing pages that are actually used

Memory

Virtual Memory

Other Tricks

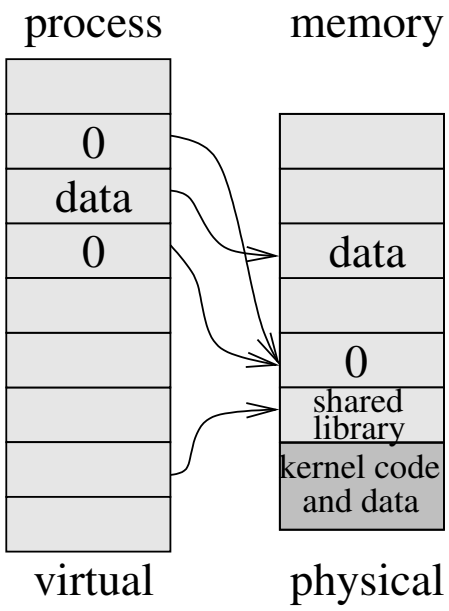


Zero Page

Memory

Virtual Memory

Other Tricks



Zero Page after a write

Other Tricks

Virtual memory has other useful features like *memory mapping* of devices

Other Tricks

Virtual memory has other useful features like *memory mapping* of devices

Parts of the virtual address space can be mapped on to things other than memory, e.g., files, the screen, sound card

Memory

Virtual Memory

Other Tricks

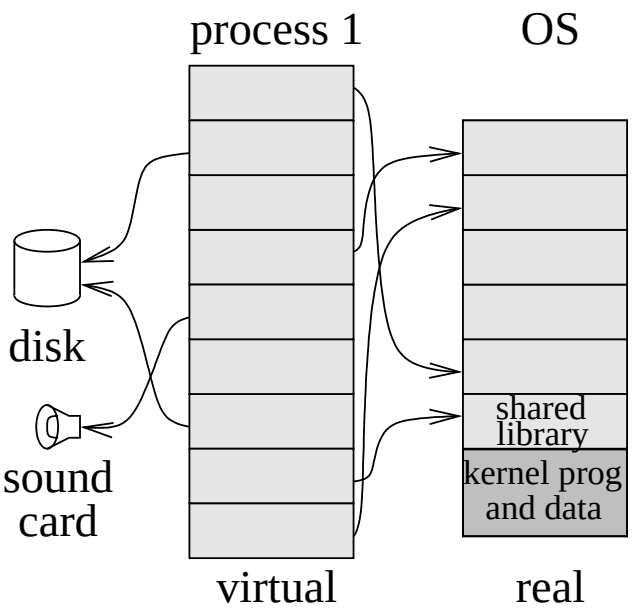
Virtual memory has other useful features like *memory mapping* of devices

Parts of the virtual address space can be mapped on to things other than memory, e.g., files, the screen, sound card

The OS can *mmap* (all or parts of) a file into memory: this means that reads and writes to "memory" are converted by the OS to reads and writes to that file (or screen, etc.)

Memory

Virtual Memory



Memory map

Memory

Virtual Memory

The hugely simplifies the problem for the programmer: rather than having to work out the fiddly details for a given piece of hardware, they can simply write to what looks like an area of memory and the OS sorts out all the details

Memory

Virtual Memory

Conclusion: TLBs solve many problems!

□