Memory

Virtual Memory

OSs often use *lazy* page allocation: don't allocate anything until the process actually accesses a page, so physical memory is only actually allocated on a page fault when we know we really need it

Memory

Virtual Memory

OSs often use *lazy* page allocation: don’t allocate anything until the process actually accesses a page, so physical memory is only actually allocated on a page fault when we know we really need it

If process requests 10GB and only uses 1GB, this is not a problem: only 1Gb will be mapped in the page table

Memory
Virtual Memory

OSs often use *lazy* page allocation: don't allocate anything until the process actually accesses a page, so physical memory is only actually allocated on a page fault when we know we really need it

If process requests 10GB and only uses 1GB, this is not a problem: only 1Gb will be mapped in the page table

And the process's virtual size can easily be bigger than the physical memory size, either through unmapped or swapped pages

Memory

Virtual Memory

The cost is kept low though the use of the TLB, but remember a page fault is relatively expensive

Memory

Virtual Memory

The cost is kept low though the use of the TLB, but remember a page fault is relatively expensive

And swapping is orders of magnitude slower still: we want to avoid swapping if at all possible

Memory

Virtual Memory

The cost is kept low though the use of the TLB, but remember a page fault is relatively expensive

And swapping is orders of magnitude slower still: we want to avoid swapping if at all possible

This is something in the hands of the programmer: don’t use memory stupidly!

Memory

Virtual Memory

Note that the terms “paging” and “swapping” are near-indistinguishable these days

Memory

Virtual Memory

Note that the terms “paging” and “swapping” are near-indistinguishable these days

Swapping used to mean entire processes

Memory

Virtual Memory

Note that the terms “paging” and “swapping” are near-indistinguishable these days

Swapping used to mean entire processes

Then *segments* (certain large areas) of memory

Memory

Virtual Memory

Note that the terms “paging” and “swapping” are near-indistinguishable these days

Swapping used to mean entire processes

Then *segments* (certain large areas) of memory

Now just pages are swapped

Memory

Virtual Memory

Note that the terms “paging” and “swapping” are near-indistinguishable these days

Swapping used to mean entire processes

Then *segments* (certain large areas) of memory

Now just pages are swapped

Note that when swapping a page back into memory, it doesn’t matter where in physical memory we put it : the page table/TLB ensures the process sees it in the same virtual place

Memory

Virtual Memory

TLBs are good but have limitations:

Memory

Virtual Memory

TLBs are good but have limitations:

- they are quite small capacity, but usually big enough to be highly effective

Memory

Virtual Memory

TLBs are good but have limitations:

- they are quite small capacity, but usually big enough to be highly effective
- they rely on temporal locality to be effective: again OK for all but weird programs

Memory

Virtual Memory

TLBs are good but have limitations:

- they are quite small capacity, but usually big enough to be highly effective
- they rely on temporal locality to be effective: again OK for all but weird programs

Perhaps the biggest problem is whenever the OS does a *context switch*, i.e., chooses to schedule to run a different process. The TLB must be *flushed* as the incoming process will have a different set of virtual-physical mappings

Memory

Virtual Memory

TLBs are good but have limitations:

- they are quite small capacity, but usually big enough to be highly effective
- they rely on temporal locality to be effective: again OK for all but weird programs

Perhaps the biggest problem is whenever the OS does a *context switch*, i.e., chooses to schedule to run a different process. The TLB must be *flushed* as the incoming process will have a different set of virtual-physical mappings

The TLB will then be re-populated by a bunch of TLB misses and page faults as the incoming process runs

Memory

Virtual Memory

This is a major reason why a context switch is so expensive: on top of the cost for the save/restore of process state there is a large overhead for the subsequent TLB misses

Memory

Virtual Memory

This is a major reason why a context switch is so expensive: on top of the cost for the save/restore of process state there is a large overhead for the subsequent TLB misses

Exercise. Read about the Spectre and Meltdown hardware bugs

Memory
Virtual Memory

This is a major reason why a context switch is so expensive: on top of the cost for the save/restore of process state there is a large overhead for the subsequent TLB misses

Exercise. Read about the Spectre and Meltdown hardware bugs

Exercise. Think about the difference between vectors and linked lists in terms of virtual memory and TLBs

Memory

Virtual Memory

This is a major reason why a context switch is so expensive: on top of the cost for the save/restore of process state there is a large overhead for the subsequent TLB misses

Exercise. Read about the Spectre and Meltdown hardware bugs

Exercise. Think about the difference between vectors and linked lists in terms of virtual memory and TLBs

Exercise to think about: the page tables in memory can grow so large they need to be swapped themselves…

Memory

Virtual Memory

Examples. A “Hello world” program in C, Java, Python and Perl

	C	Java	Python	Perl
Resident size KB	430	16500	4300	1850
Minor Fault	150	3800	1200	530
Major Fault	0	0	0	0
Context switch	2	150	8	4

In Linux 3.11.10; 8GB memory

Numbers are approximate and vary on runs due to scheduling

□