Demand Paged Virtual Memory — Today

- Up to now, we (mostly) assumed that the virtual address space of a process fit in memory.
- OS illusions:
  1. Memory is as big and cheap as disk
  2. Memory is as fast as DRAM (cache adds the illusion that memory is as fast as SRAM)
- Put another way
  1. treat disk (or other backing store) as a much larger, but much slower main memory
  2. analogous to the way in which main memory is a much larger, but much slower, cache or set of registers
- The illusion of an infinite virtual memory enables
  1. a process to be larger than physical memory, and
  2. a process to execute even if all of the process is not in memory (we assumed this already for general paging)
  3. Allows more processes than fit in memory to run concurrently.
- Demand paging even without large virtual memory is a good idea.

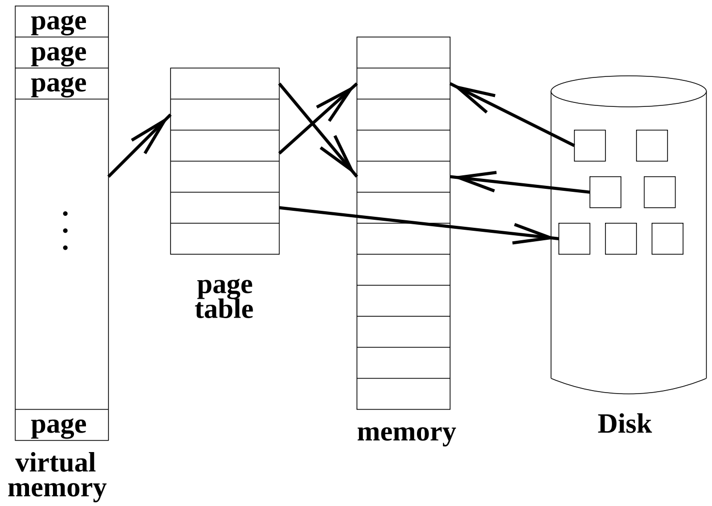
Demand Paged Virtual Memory

Idea – Demand Paging uses memory as a cache for disk.

The page table (memory map) indicates if the page is on disk, and thus is not in a physical page frame in memory right now using a valid bit. To access this page, the OS needs to put the page in a physical page frame. Once the page is in a frame, the OS updates the page table to reflect the placement.

To effectively use large virtual address spaces, we need an efficient implementation that finds the page in memory the vast majority of the time, otherwise the effective memory access time will approach that of the disk.

Key – Locality: the *working set* size of a process must fit in memory, and must stay there. (90/10 rule.)



When to load a page?

When the process starts: Load all the pages - the virtual address space must no larger than the physical memory.

Overlays: Application programmer indicates when to load and remove pages. Allows virtual address space to be larger than physical address space, but it is painful to do and is error-prone.

Demand paging: OS loads a page the first time it is referenced. May remove a page from memory to make room for the new page. A *page-fault* is the name for the interrupt that occurs when an instruction references a page that is not in memory. Process must give up the CPU while the page is being loaded.

Pre-paging: OS guesses in advance which pages the process will need and pre-loads them into memory. Allows more overlap of CPU and I/O if the OS guesses correctly. If the OS is wrong, the page is faulted in. Errors also may result in removing useful pages. Difficult to get right due to branches in code.

Implementation of Demand Paging

1. Swap space - a special part of the disk for running processes.
2. A copy of the entire program must be stored on disk (in swap space). Why?
3. Valid bit in page table indicates if page is in memory.

   1: in memory

   0: not in memory (either on disk or bogus address)
4. If the page is not in memory, the hardware traps to the OS on the reference to the invalid page.
5. The OS checks that the address is valid. Assuming it is, it...
   (a) selects a page to replace (page replacement algorithm)
   (b) invalidates the old page in the page table
   (c) starts loading new page into memory from disk
   (d) context switches to another process while I/O is being done
   (e) gets interrupt that page is loaded in memory
   (f) updates the page table entry
   (g) continues faulting process (why not continue current process?)

All of this is functionally transparent to the user (it may not be performance transparent!). The OS keeps the processor busy by running another job.

Swap Space

What happens when a page is removed from memory to make room for a new page?

- If the page contained code, we could simply remove it since it can be re-loaded from the disk.
- If the page contained data, we need to save the data so that it can be reloaded if the process it belongs to refers to it again.
- A portion of the disk is reserved as *swap space* where pages are moved when they are forced to give up their physical frame.

$\Longrightarrow$ At any given time, a page of virtual memory might exist in one or more of:

- The file system (never loaded pages)
- Physical memory
- Swap space

$\Longrightarrow$ Page table must be more sophisticated so that it knows where to find a page.

Performance of Demand Paging

- Theoretically, a process could access a new page (or more!) of memory with each instruction.
- Fortunately, processes typically exhibit *locality of reference* which yields reasonable performance for demand paging.
- Locality:

  Temporal locality: if a process accesses an item in memory, it will tend to reference the same item again soon.

  Spatial locality: if a process accesses an item in memory, it will tend to reference an adjacent item soon.

- Let p be the probability of a page fault $(0 \leq p \leq 1)$.
  Effective access time $= (1 - p) \times$ ma $+ p\times$ page fault time

  Assume memory access time is 200 nanoseconds while it takes 25 milliseconds to service a page fault. Then...
  Effective access time $= (1 - p) \times 200 + p\times 25{,}000{,}000$

  If we want the effective access time to be only 10% slower than memory access time, what value must p have?

Software-Loaded TLB with Virtual Memory

We have to keep the page table up-to-date too.

- In some implementations, the hardware loads the TLB on a TLB miss.
- If the TLB hit rate is very high, we can afford to have the software load the TLB (In Nachos the TLB is software loaded).
- An implementation,
  1. Valid bit in the TLB indicates if page is in memory.
  2. on a TLB hit, use the frame number to access memory
  3. trap on a TLB miss, the OS then
     (a) checks if the page is in memory
     (b) if page is in memory, OS picks a TLB entry to replace and then fills it in the new entry
     (c) if page is not in memory, OS picks a TLB entry to replace and fills it in as follows
        i. invalidates TLB entry
        ii. perform page fault operations as described earlier
        iii. updates TLB entry
        iv. restarts faulting process

All of this is still functionally transparent to the user.

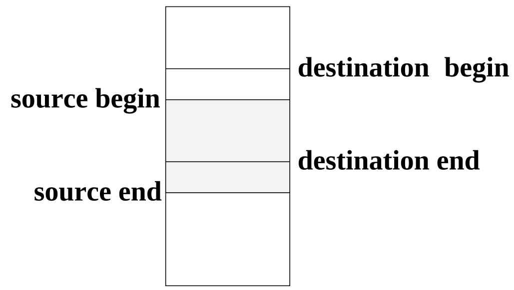
Transparent Page Faults

How does the OS transparently restart a faulting instruction?

- Hardware has to help out by saving (1) the faulting instruction, and (2) the CPU state.
- What about instructions with side-effects? (CISC)

  mov a, (r10)+

  Moves a into the address contained in register 10 and increments register 10.
- Solution: unwind side effects
- Block transfer instructions where the source and destination overlap can't be undone.



- Solution: check that all pages between the starting and ending addresses of the source and destination are in memory before starting the block transfer

Summary

Benefits of demand paging:

- Virtual address space can be larger than physical address space.
- Processes can run without being fully loaded into memory.
  - Processes start faster because they only need to load a few pages (for code and data) to start running.
  - Processes can share memory more effectively, reducing the costs when a context switch occurs.

Next Time

Page replacement algorithms: picking a page to replace