University of Arizona

CSc 340

Foundations of Computer Systems

Christian Collberg

April 19, 2001

Paging

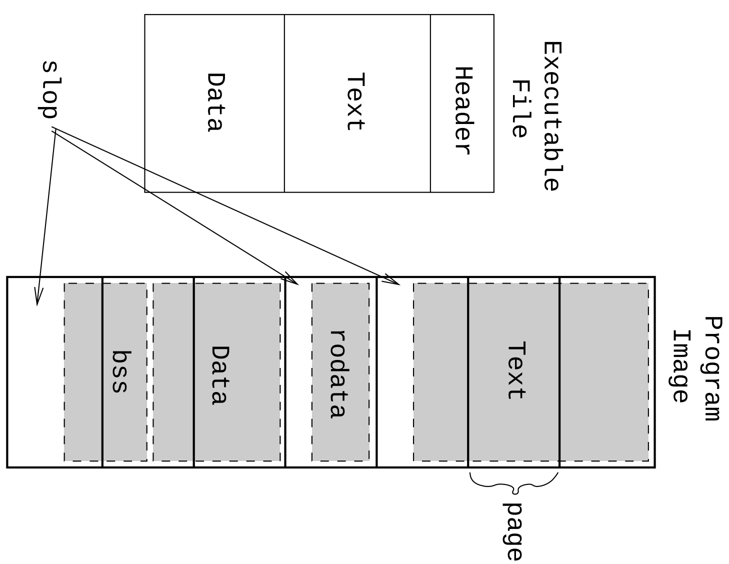
Paging

- Segments are different sizes and each must be stored in contiguous physical memory; this leads to fragmentation problems:
  1. Makes it difficult to move segments.
  2. Makes it difficult to swap segments to disk.

Paging...

- Solution: allocate memory in fixed-size chunks called pages.
  - Each page is the same size (a power of two).
  - Typical sizes range from 512 to 8K bytes.
  - A page of physical memory is often called a page frame to distinguish it from a page of virtual memory.
- For each process, a page table defines the base address of each page along with access (read/write/execute) and existence bits.

Paging...



Slide 22–4

Page Table

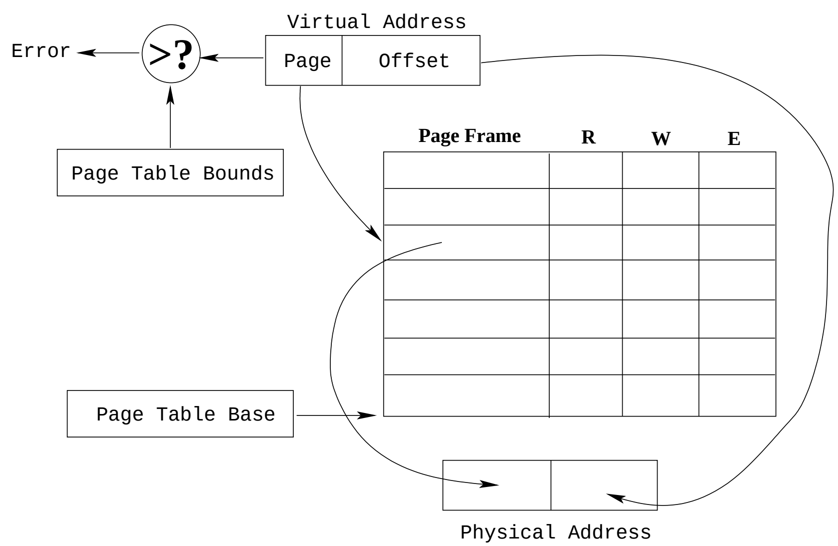
- Easy to allocate: keep a free list of available pages and grab the first one. Easy to swap since everything is the same size.
- Can share memory by sharing page table entries (page frames). Note that the virtual addresses do not need to be the same.

Slide 22–5

Page Table

- The existence bit is used to determine if the page is in memory or not (if it is in a page frame).
  1. If the bit is 1, translation proceeds as shown above.
  2. If it's 0, a page fault exception occurs. The operating system then finds the page on the disk, brings it into memory, updates the page table, and re-executes the offending instruction. This trick is called virtual memory.

Slide 22–6



Slide 22–7

Paging Loader

```
struct execFileHeader {
   unsigned int startAddr,textSize,dataSize,bssSize;}

char* pagingLoader (FILE *execFile) {
   struct execHeader header = readHeader(execFile);
   char *pageMap[MAX_PAGES];

   unsigned int pgmSize = header.textSize +
         header.dataSize + header.bssSize;
   int npages = (pgmSize+PAGE_SIZE-1)/PAGE_SIZE;

   for (p=0; p < npages; p++)
      pageMap[p] = getPageFrame();
```

Slide 22–8

Paging Loader...

```
   for (p=0; p < npages; p++) {
     char *byteAddr = pageMap[p];
     for the next PAGE_SIZE bytes b in execFile do {
         *byteAddr = b;
         byteAddr++;
     }
   }

   setPageMapping(pageMap,npages);
   return (char*) header.startAddr;
}
```

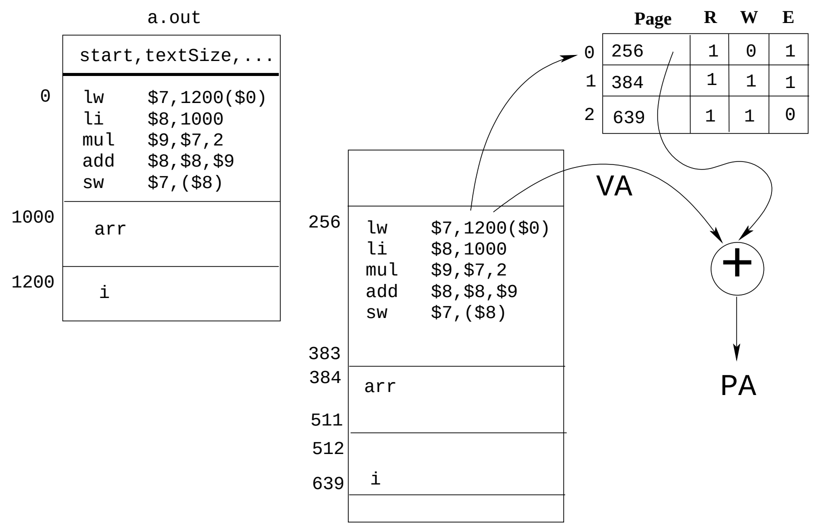
Slide 22–9

Example

- The next slide shows our running example executing in a paged environment.

```
        .data
arr     .space 200
i:      .word
        .text
lw      $7, 1200($0)
li      $8, 1000
mul     $9, $7, 2
add     $8, $8, $9
sw      $7, ($8)
```

Slide 22–10



Slide 22–11

Problems with paging

- Efficiency of access: even small page tables are generally too large to be stored in the MMU. Instead, page tables are kept in main memory and the MMU has only the page table's base address. It thus takes one overhead reference for every real memory reference.
- Table space: page tables can be large. Consider a 32-bit address space with 4k pages. How much memory does the table require? Partial solution: keep base and bounds for page table, so only large processes have to have large tables.

Problems with paging...

- Internal fragmentation: page size doesn't match up with information size. The larger the page is, the worse this is.

Readings and References

- MacCabe, pp. 388–392.