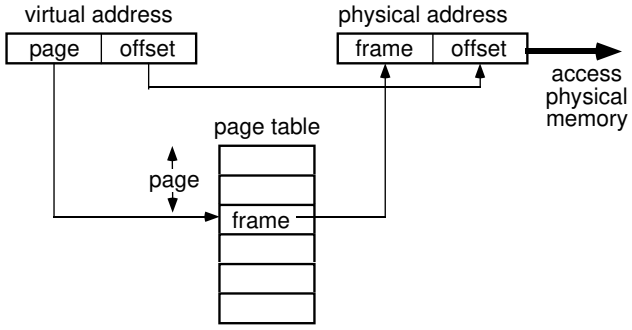
Paging

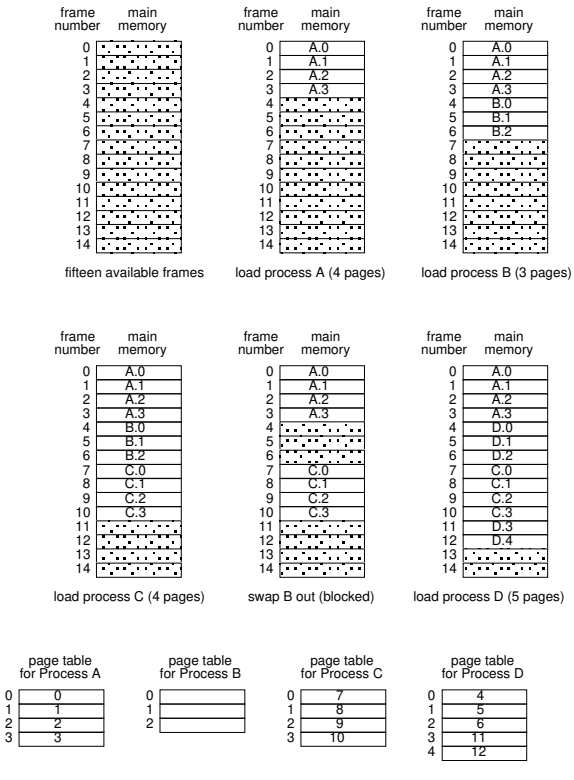
- Compared to segmentation, paging:
  - Makes allocation and swapping easier
  - No external fragmentation
- Each process is divided into a number of small, fixed-size partitions called *pages*
  - Physical memory is divided into a large number of small, fixed-size partitions called *frames*
  - Pages have nothing to do with segments
  - Page size = frame size
    - Usually 512 bytes to 16K bytes
  - The whole process is still loaded into memory, but the pages of a process do ***not*** have to be loaded into a contiguous set of frames
  - Virtual address consists of page number and offset from beginning of that page

Implementing Paging

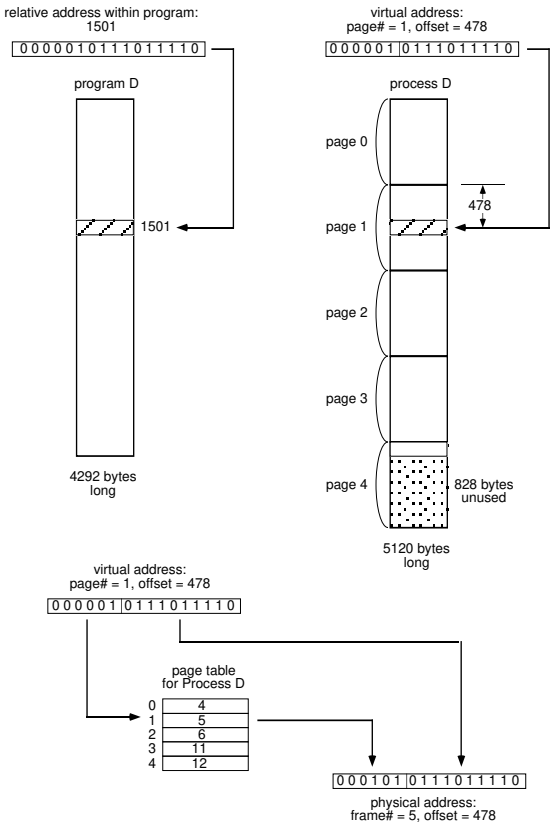


- A *page table* keeps track of every page in a particular process
  - Each entry contains the corresponding frame in main (physical) memory
  - Can add protection bits, but not as useful
- Additional hardware support required is slightly less than for segmentation
  - No need to keep track of, and compare to, limit. Why not?

Paging Example



Paging Example (cont.)



Managing Pages and Frames

- OS usually keeps track of free frames in memory using a bit map
  - A bit map is just an array of bits
    - 1 means the frame is free
    - 0 means the frame is allocated to a page
  - To find a free frame, look for the first 1 bit in the bit map
    - Most modern instruction sets have an instruction that returns the offset of the first 1 bit in a register
- Page table base pointer (special register) points to page table of active process
  - Saved/restored as part of context switch
  - Page table also contains:
    - Valid bit — indicates page is in the process's virtual address space
    - Other bits for demand paging (discussed next time)

Evaluation of Paging

- Advantages:
  - Easy to allocate memory — keep a list of available frames, and simple grab first one that's free
  - Easy to swap — pages, frames, and often disk blocks as well, all are same size
  - One frame is just as good as another!
- Disadvantages:
  - Page tables are fairly large
    - Most page tables are too big to fit in registers, so they must live in physical memory
    - This table lookup adds an extra memory reference for every address translation
  - Internal fragmentation
    - Always get a whole page, even for 1 byte
    - Larger pages makes the problem worse
    - Average = 1/2 page per process

Address Translation, Revisited

- A modern microprocessor has, say, a 32 bit virtual address space (2^{32} = 4 GB)
  - If page size is 1k (2^{10}), that means all the page tables combined could have over 2^{22} (approximately 4 million) page entries, each at least a couple of bytes long
  - Problem: if the main memory is only, say, 16 Mbytes, storing these page table there presents a problem!
    - Solution: store page tables in virtual memory, bring in pieces as necessary
  - New problem: memory access time may be doubled since the page tables are now subject to paging
    - (one access to get info from page table, plus one access to get data from memory)
    - New solution: use a special cache (called a Translation Lookaside Buffer (TLB)) to cache page table entries

Translation Lookaside Buffer (TLB)

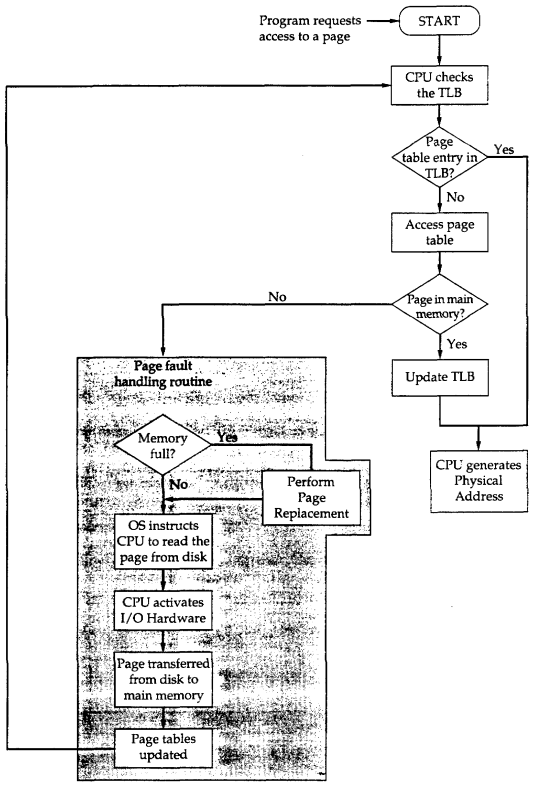


FIGURE 5.16 Operation of paging and translation lookaside buffer (TLB) [FURH87]

Paging and Segmentation

- Use two levels of mapping:
  - Process is divided into variable-size segments
    - Segments are logical divisions as before
  - Each segment is divided into many small fixed-size pages
    - Pages are easy for OS to manage
    - Eliminates external fragmentation
  - Virtual address = segment, page, offset
  - One segment table per process, one page table per segment
- Sharing at two levels: segment, page
  - Share frame by having same frame reference in two page tables
  - Share segment by having same base in two segment tables
  - Still need protection bits (sharing, r/o, r/w)