Memory Management

CS 241: Systems Programming
Discussion, Week 9

slides adapted by Sameer Sundresh

Outline

- LMP1 QuizB: passwd
- Memory Management
  - *Fragmentation*
  - *Storage Placement Algorithms*
  - *malloc revisited*
  - *paging*
- Virtual Memory
  - *Why Virtual Memory*
  - *Virtual Memory Addressing*
  - *TLB (Translation Lookaside Buffer)*

Memory Management

Virtual Memory

Fragmentation

- External Fragmentation

  Free space becomes divided into many small pieces

  Caused over time by allocating and freeing the storage of different sizes

- Internal Fragmentation

  Result of reserving space without ever using its part

  Caused by allocating fixed size of storage

Storage Placement Algorithms

- Best Fit
- First Fit
- Next Fit
- Worst Fit

Exercise

- Consider a swapping system in which memory consists of the following hole sizes in memory order: 10KB, 4KB, 20KB, 18KB, 7KB, 9KB, 12KB, and 15KB. Which hole is taken for successive segment requests of (a) 12KB, (b) 10KB, (c) 9KB for

    First Fit?

    Best Fit?

    Worst Fit?

    Next Fit?

malloc Revisited

- Free storage is kept as a list of free blocks

  Each block contains a size, a pointer to the next block, and the space itself

- When a request for space is made, the free list is scanned until a big-enough block can be found

  Which storage placement algorithm is used?

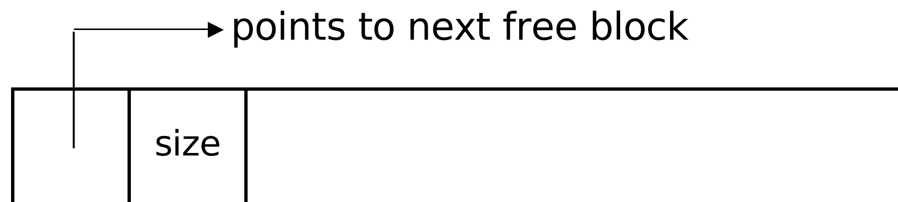
- If the block is found, return it and adjust the free list. Otherwise, another large chunk is obtained from the OS and linked into the free list

malloc Revisited (continued)

```
typedef long Align;      /* for alignment to long */

union header {           /* block header */
  struct {
    union header *ptr;   /* next block if on free list */
    unsigned size;       /* size of this block */
  } s;
  Align x;               /* force alignment of blocks */
};

typedef union header Header;co
```



Paging

- Divide memory into pages, all of equal size
  - *We don't need to assign contiguous chunks*
  - *Internal fragmentation can only occur on the last page assigned to a process*
  - *External fragmentation cannot occur at all*

Memory Management

Virtual Memory

Why Virtual Memory?

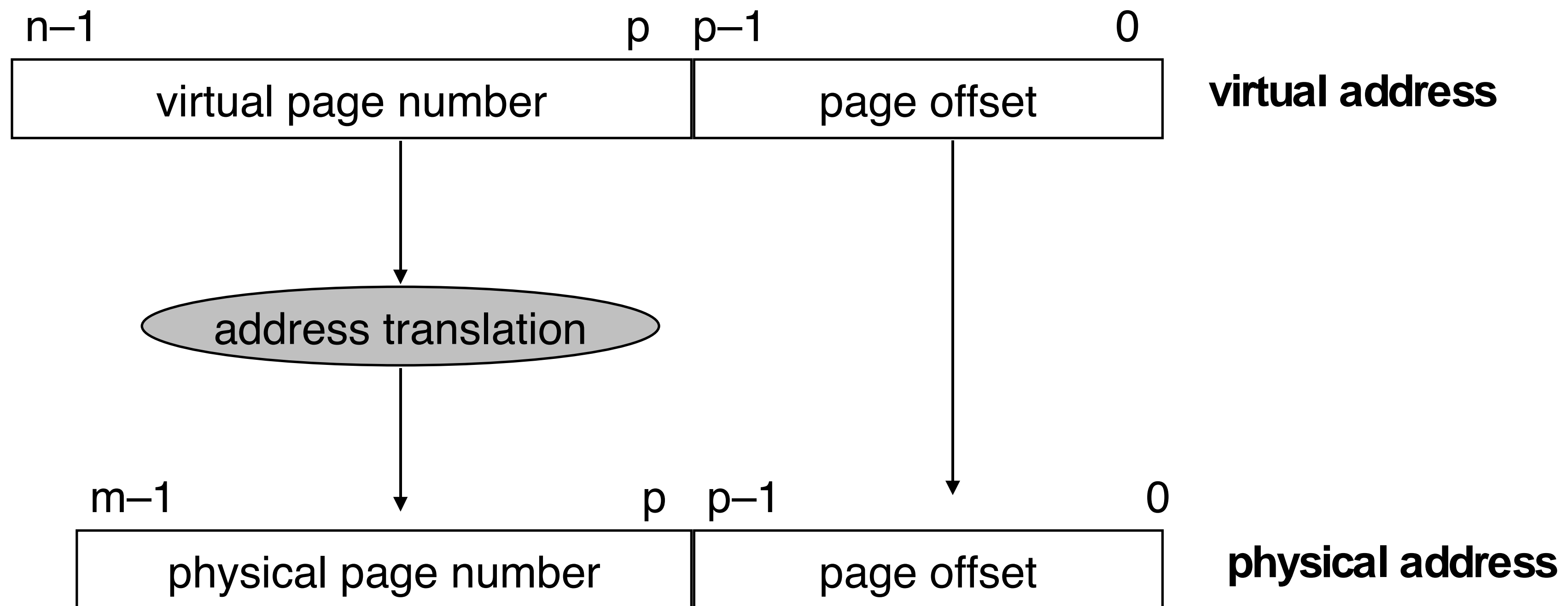
- Use main memory as a Cache for the Disk
  - *Address space of a process can exceed physical memory size*
  - *Sum of address spaces of multiple processes can exceed physical memory*
- Simplify Memory Management
  - *Multiple processes resident in main memory.*
    - *Each process with its own address space*
  - *Only "active" code and data is actually in memory*
- Provide Protection
  - *One process can't interfere with another.*
    - *because they operate in different address spaces.*
  - *User process cannot access privileged information*
    - *different sections of address spaces have different permissions.*

Principle of Locality

- Program and data references within a process tend to cluster
- Only a few pieces of a process will be needed over a short period of time (active data or code)
- Possible to make intelligent guesses about which pieces will be needed in the future
- This suggests that virtual memory may work efficiently

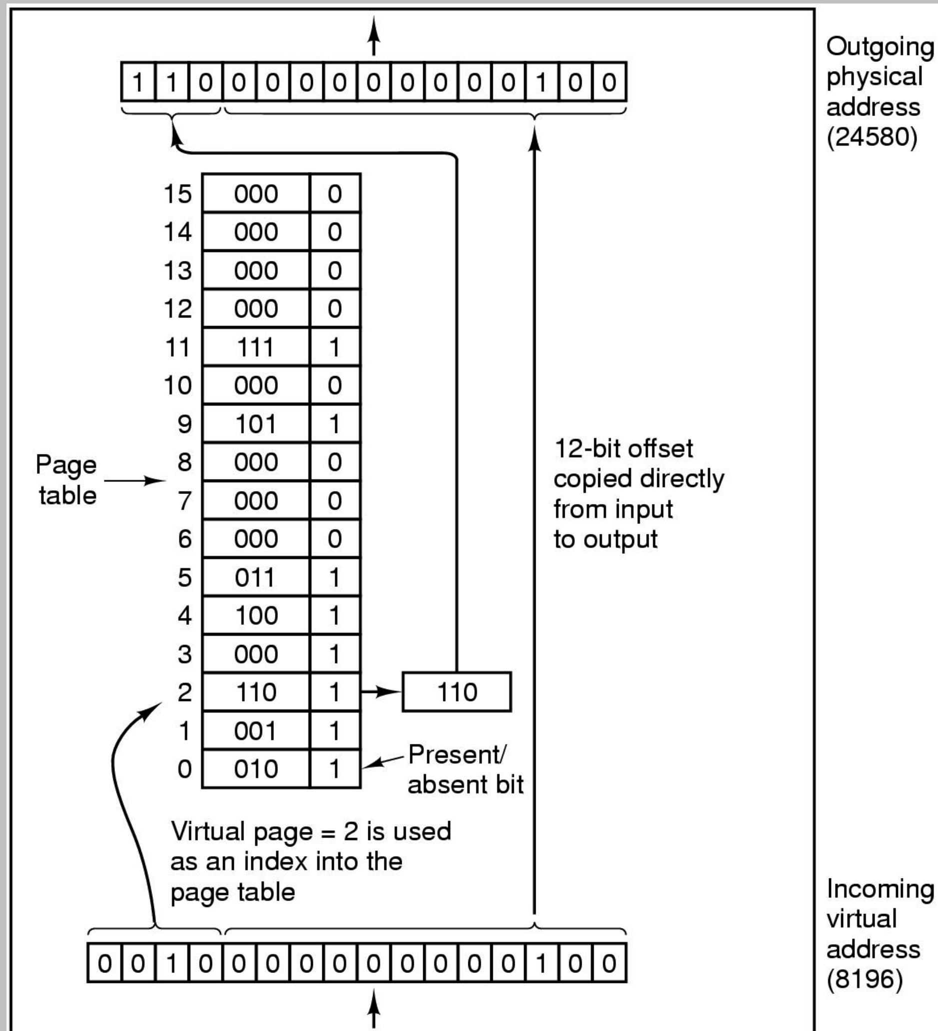
VM Address Translation

- Parameters
  - $P = 2^p$ = *page size (bytes).*
  - $N = 2^n$ = *Virtual address limit*
  - $M = 2^m$ = *Physical address limit*



Page offset bits don't change as a result of translation

Page Table



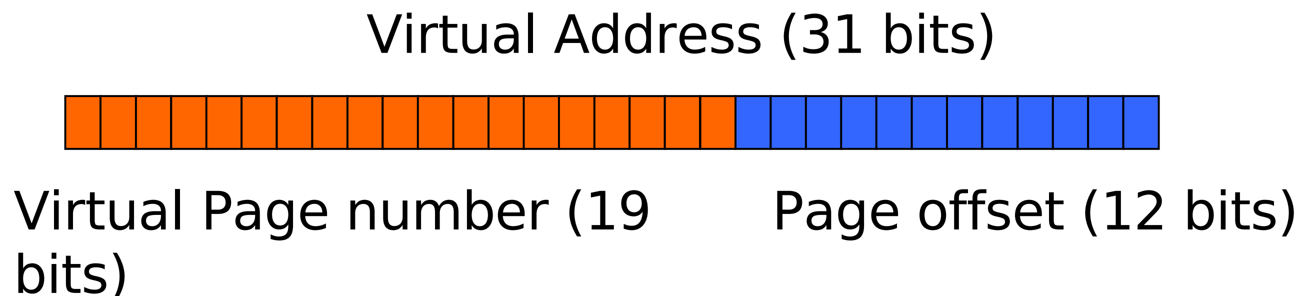
- Keeps track of what pages are in memory
- Provides a mapping from virtual address to physical address

Handling a Page Fault

- Page fault
  - *Look for an empty page in RAM*
    - *May need to write a page to disk and free it*
  - *Load the faulted page into that empty page*
  - *Modify the page table*

Addressing

- 64MB RAM (2^26)
- 2^31 (2GB) total memory
- 4KB page size (2^12)
- So we need 2^12 for the offset, we can use the remainder bits for the page
  - *19 bits, we have 2^19 pages (524288 pages)*



Translation Lookaside Buffer (TLB)

- Each virtual memory reference can cause two physical memory accesses
  - *One to fetch the page table*
  - *One to fetch the data*
- To overcome this problem a high-speed cache is set up for page table entries
- Contains page table entries that have been most recently used (a cache for page table)

Translation Lookaside Buffer (TLB)

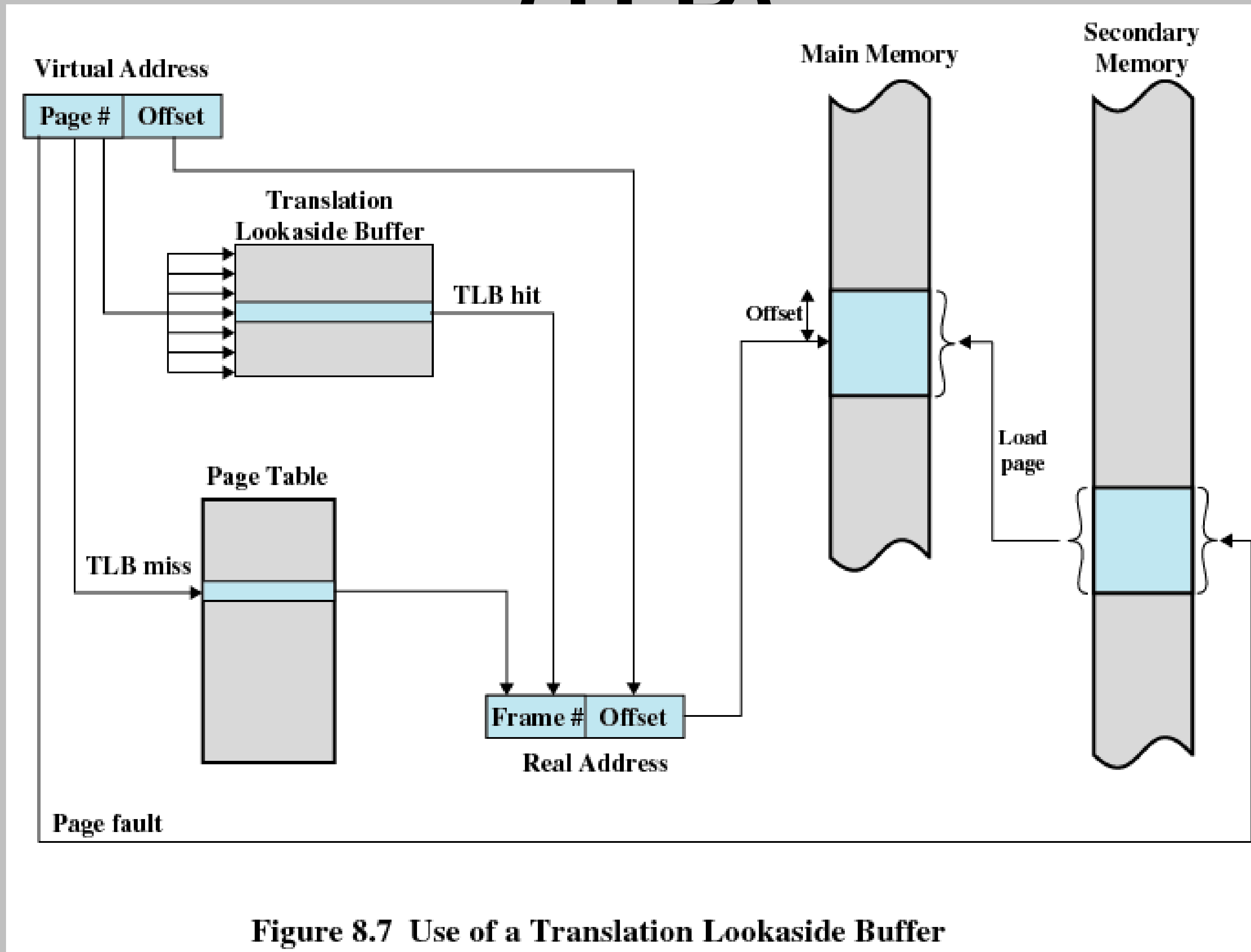


Figure 8.7 Use of a Translation Lookaside Buffer

Address Conversion

- That 19bit page address can be optimized in a variety of ways
  - *Translation Look-aside Buffer*
    - *m – memory cycle, - hit ratio, - TLB lookup time*
    - *Effective access time (Eat)*
      - *Eat =* $(m + \ ) \quad (2m + \ )(1 - \ ) = 2m + \ - m$
  - *Multilevel Page Table*
    - *Similar to indirect pointers in I-nodes*
    - *Split the 19bits into multiple sections*
  - *Inverted Page Table*
    - *Much smaller, but is slower and more difficult to lookup*

Multilevel Page Tables

- Given:
  - *4KB (2^{12}) page size*
  - *32-bit address space*
  - *4-byte PTE*
- Problem:
  - *Would need a 4 MB page table!*
    - *2^{20} *4 bytes*
- Common solution
  - *multi-level page tables*
  - *e.g., 2-level table (P6)*
    - *Level 1 table: 1024 entries, each of which points to a Level 2 page table.*
    - *Level 2 table: 1024 entries, each of which points to a page*