CISC 3320
C20c A Few Other Considerations for Paging

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Acknowledgement

- These slides are a revision of the slides provided by the authors of the textbook via the publisher of the textbook

Outline

- Memory-Mapped Files
- Allocating Kernel Memory
- Other Considerations

- Operating-System Examples

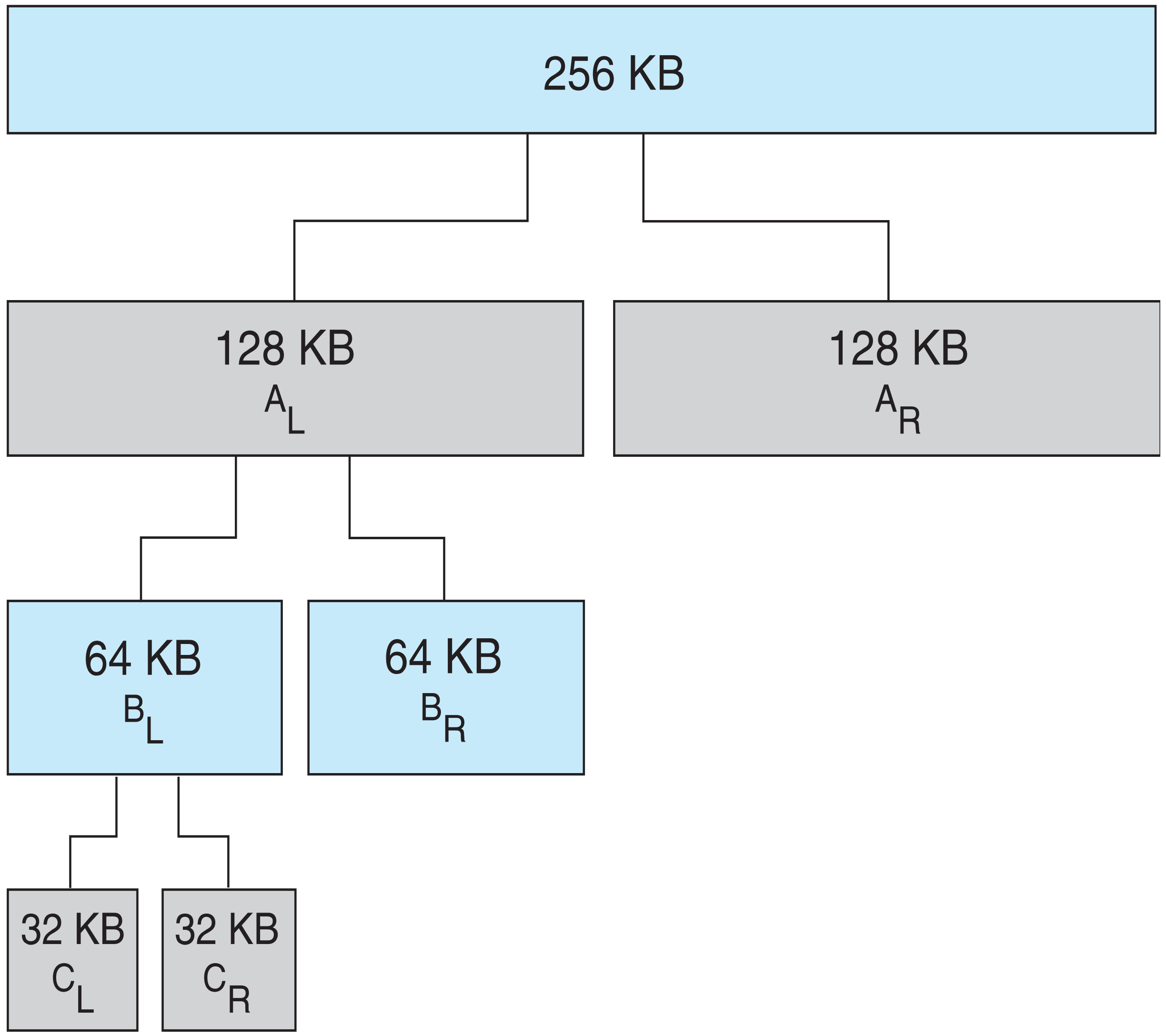
Allocating Kernel Memory

- Treated differently from user memory
- Often allocated from a free-memory pool
  - Kernel requests memory for structures of varying sizes
  - Some kernel memory needs to be contiguous
    - i.e. for device I/O

Buddy System

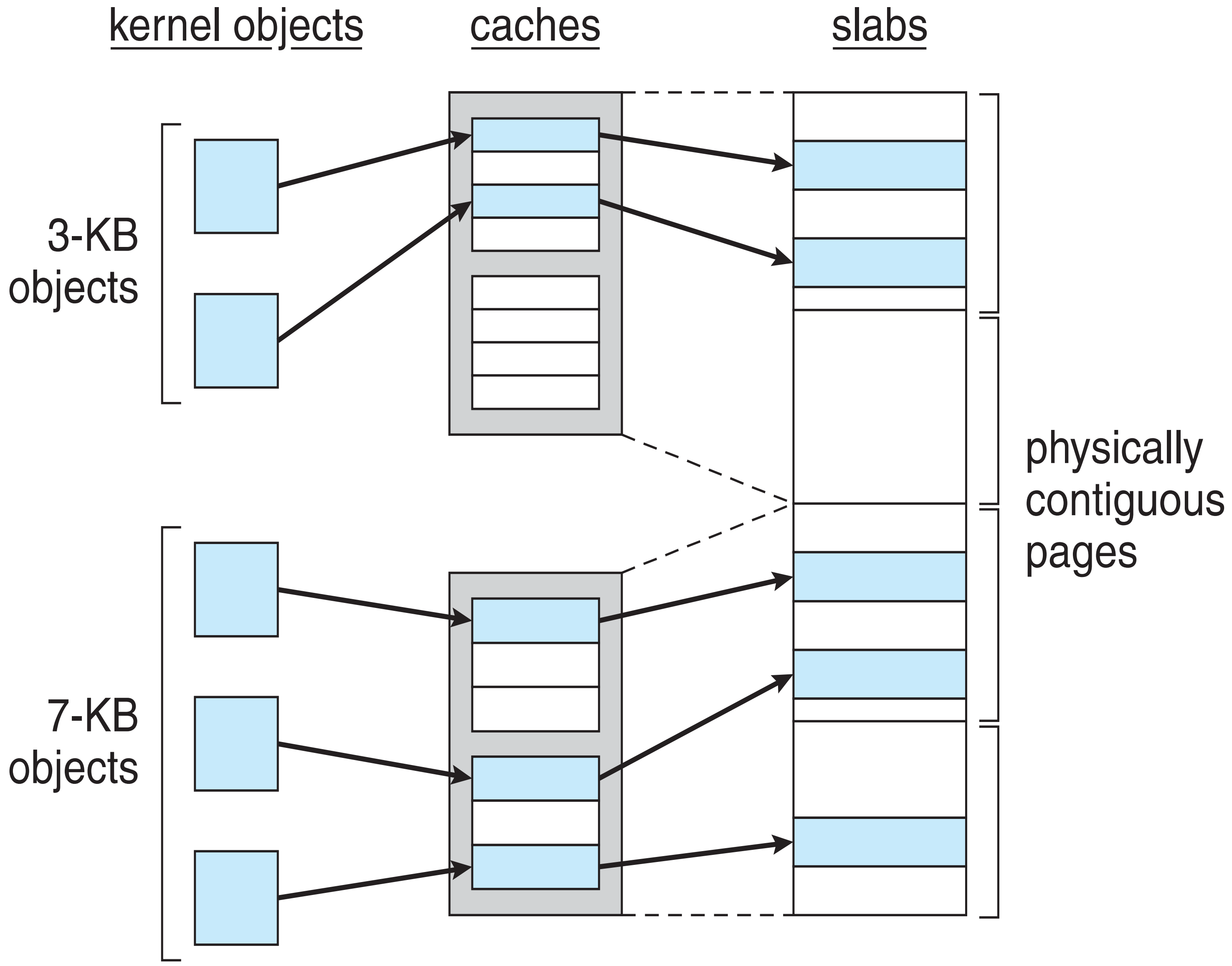
- Allocates memory from fixed-size segment consisting of physically-contiguous pages
- Memory allocated using **power-of-2 allocator**
  - Satisfies requests in units sized as power of 2
  - Request rounded up to next highest power of 2
  - When smaller allocation needed than is available, current chunk split into two buddies of next-lower power of 2
    - Continue until appropriate sized chunk available
- For example, assume 256KB chunk available, kernel requests 21KB
  - Split into A_L and A_R of 128KB each
    - One further divided into B_L and B_R of 64KB
      - One further into C_L and C_R of 32KB each – one used to satisfy request
- Advantage – quickly **coalesce** unused chunks into larger chunk
- Disadvantage - fragmentation

physically contiguous pages



Slab Allocator

- Alternate strategy
- **Slab** is one or more physically contiguous pages
- **Cache** consists of one or more slabs
- Single cache for each unique kernel data structure
  - Each cache filled with **objects** – instantiations of the data structure
- When cache created, filled with objects marked as `free`
- When structures stored, objects marked as `used`
- If slab is full of used objects, next object allocated from empty slab
  - If no empty slabs, new slab allocated
- Benefits include no fragmentation, fast memory request satisfaction


kernel objects
caches
slabs
3-KB objects
7-KB objects
physically contiguous pages

Slab Allocation in Linux

- Slab started in Solaris, now wide-spread for both kernel mode and user memory in various OSes
- Linux 2.2 had SLAB, now has both SLOB and SLUB allocators
  - SLOB for systems with limited memory
    - Simple List of Blocks – maintains 3 list objects for small, medium, large objects
  - SLUB is performance-optimized SLAB removes per-CPU queues, metadata stored in page structure

Slab Allocation in Linux

- For example process descriptor is of type `struct task_struct`
- Approx 1.7KB of memory
- New task -> allocate new struct from cache
  - Will use existing free `struct task_struct`
- Slab can be in three possible states
  1. Full – all used
  2. Empty – all free
  3. Partial – mix of free and used
- Upon request, slab allocator
  1. Uses free struct in partial slab
  2. If none, takes one from empty slab
  3. If no empty slab, create new empty

Other Considerations

- Prepaging
- Page size
- TLB reach
- Inverted page table
- Program structure
- I/O interlock and page locking

Prepaging

- To reduce the large number of page faults that occurs at process startup
- Prepage all or some of the pages a process will need, before they are referenced
- But if prepaged pages are unused, I/O and memory was wasted
- Assume *s* pages are prepaged and *α* of the pages is used
  - Is cost of ***s * α*** save pages faults > or < than the cost of prepaging
    s * (1 - α) unnecessary pages?
  - ***α*** near zero $\Rightarrow$ prepaging loses

Page Size

- Sometimes OS designers have a choice
  - Especially if running on custom-built CPU
- Page size selection must take into consideration:
  - Fragmentation
  - Page table size
  - **Resolution**
  - I/O overhead
  - Number of page faults
  - Locality
  - TLB size and effectiveness
- Always power of 2, usually in the range 2^{12} (4,096 bytes) to 2^{22} (4,194,304 bytes)
- On average, growing over time

TLB Reach

- TLB Reach
  - The amount of memory accessible from the TLB
- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB
  - Otherwise there is a high degree of page faults
- Increase the Page Size
  - This may lead to an increase in fragmentation as not all applications require a large page size
- Provide Multiple Page Sizes
  - This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation

Program Structure

- Program structure
  - `int[128,128] data;`
  - Each row is stored in one page
  - Assume 128 frames allocated to the entire program
  - Program 1

```
for (j = 0; j <128; j++)
   for (i = 0; i < 128; i++)
      data[i,j] = 0;
```

  - Program 2

```
for (i = 0; i < 128; i++)
   for (j = 0; j < 128; j++)
      data[i,j] = 0;
```

  - How many page faults are there?

Program Structure

- Program structure
  - `int[128,128] data;`
  - Each row is stored in one page
  - Assume 128 frames allocated to the entire program
  - Program 1

```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

128 x 128 = 16,384 page faults

- Program 2

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

128 page faults

I/O Interlock

- **I/O Interlock**
    - Pages must sometimes be locked into memory
- Consider I/O
    - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm
- **Pinning** of pages to lock into memory

Questions?