The Linux Operating System

- Krithika Swaminathan

Topics

- Design principles
- Process Management
- Scheduling
- Memory Management
- File Systems

Scheduling

- Job of allocating CPU time to different tasks within an operating system
- Linux supports preemptive multi-tasking
- Making decisions – balance fairness and performance

- Fairness: every process has a chance to get allocated time
- Performance: best processes or highest priority processes executed

Scheduling

- Process Scheduling
- Real-Time Scheduling
- Kernel Synchronization
- Symmetric Multiprocessing

Process Scheduling

- Two algorithms:
  - Fair and preemptive
  - Priority-based
- Completely Fair Scheduler (CFS)
- Linux Scheduler
- Terms to remember:
  - Nice value – smaller nice value, higher priority - (-20 to 19)
  - Time slice – length of time the processor is afforded
  - Target latency – interval of time during which every runnable task should run once
  - Minimum granularity – minimum length of time any process should run for

Completely Fair Scheduler (CFS)

- Instead of time slices, all processes allotted a proportion of the processor’s time
- Adjusts this allotment by weighting each process’s allotment by its nice value
- Function of the total number of runnable processes

- N runnable processes --> each afforded 1/N of the processor’s time
- Smaller nice value --> receive a higher weight (and vice-versa)
- (time slice) α (process’s weight) / (total weight of all runnable processes)

Real-Time Scheduling

- Linux implements FCFS and Round Robin
- Scheduler runs process with the highest priority
- If equal priority, runs process that has been waiting longest

- Soft vs Hard real-time scheduling:
  - Hard --> guarantees a minimum latency between when a process becomes runnable and when it really runs
  - Soft --> strict guarantees about relative priorities, but no minimum latency specified

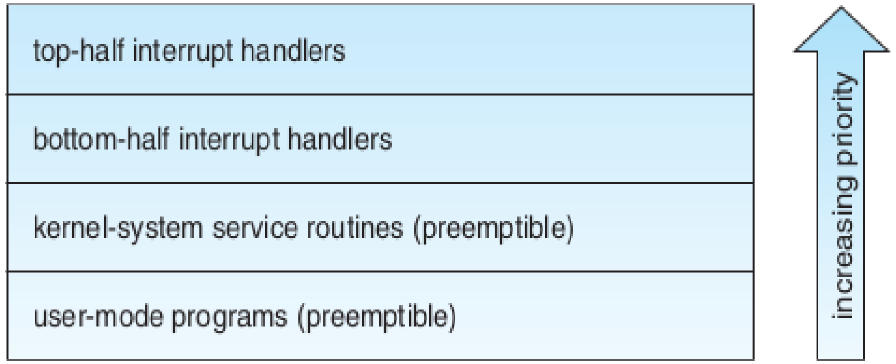
Kernel Synchronization

- Request for kernel-mode execution:
  - A running program may request OS service, explicitly or implicitly
  - A device controller may deliver a hardware interrupt
- Problem? - all tasks may try to access same internal DS --> inconsistency
- (critical section problem, shared data)
- Linux kernel provides spinlocks and semaphores for locking in the kernel
- On single-processor machines, spinlocks replaced by enabling and disabling kernel preemption --> preempt_enable() and preeempt_disable()

single processor	multiple processors
Disable kernel preemption.	Acquire spin lock.
Enable kernel preemption.	Release spin lock.

Kernel Synchronization

- Critical sections in interrupt service routines --> interrupt control H/W
- Disabling interrupts --> all I/O suspended --> performance degrades
- Solution - Synchronization architecture – Separating ISR into:
  - Top half:
    - Standard – runs with recursive interrupts disabled
    - Interrupts with same number disabled – others may ru
  - Bottom half:
    - Run with all interrupts enabled
    - Invoked automatically when an ISR exits
- kernel can complete any complex processing that has to be done in response to an interrupt without being interrupted itself



Symmetric Multiprocessing

- Linux 2.0 kernel – first stable SMP hardware
- Separate processes executed in parallel on separate processors
- Originally, only one processor at a time
- Version 2.2, single kernel lock (big kernel lock) allowed multiple processes to be active in the kernel concurrently

- Now, multiple locks – each protects a small subset of kernel's data structures

Memory Management

- Management of Physical Memory – pages, blocks of RAM
- Virtual Memory – memory-mapped into address space of running processes
  - Virtual Memory Regions
  - Lifetime of a Virtual Address Space
  - Swapping and Paging
  - Kernel Virtual Memory
- Execution and Loading of User Programs
  - Mapping of Programs into Memory
  - Static and Dynamic Linking

Management of Physical Memory

- Linux separates physical memory into:
  - ZONE DMA
  - ZONE DMA32
  - ZONE NORMAL
  - ZONE HIGHMEM
- Zones – architecture specifi

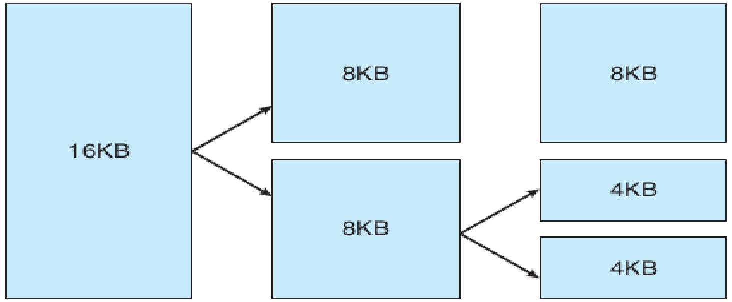
zone	physical memory
ZONE_DMA	< 16 MB
ZONE_NORMAL	16 .. 896 MB
ZONE_HIGHMEM	> 896 MB

Figure 18.3 Relationship of zones and physical addresses in Intel x86-32.

- Kernel maintains a list of free pages for each zone

Management of Physical Memory

- Page allocator
  - responsible for allocating and freeing all physical pages for the zone
  - capable of allocating ranges of physically contiguous pages on request
  - uses a buddy system to keep track of available physical pages
  - Buddy? - adjacent partner of allocatable memory region
  - two allocated partner regions freed up - combined to form larger region - buddy heap
  - converse true – subdivided into partners to satisfy re



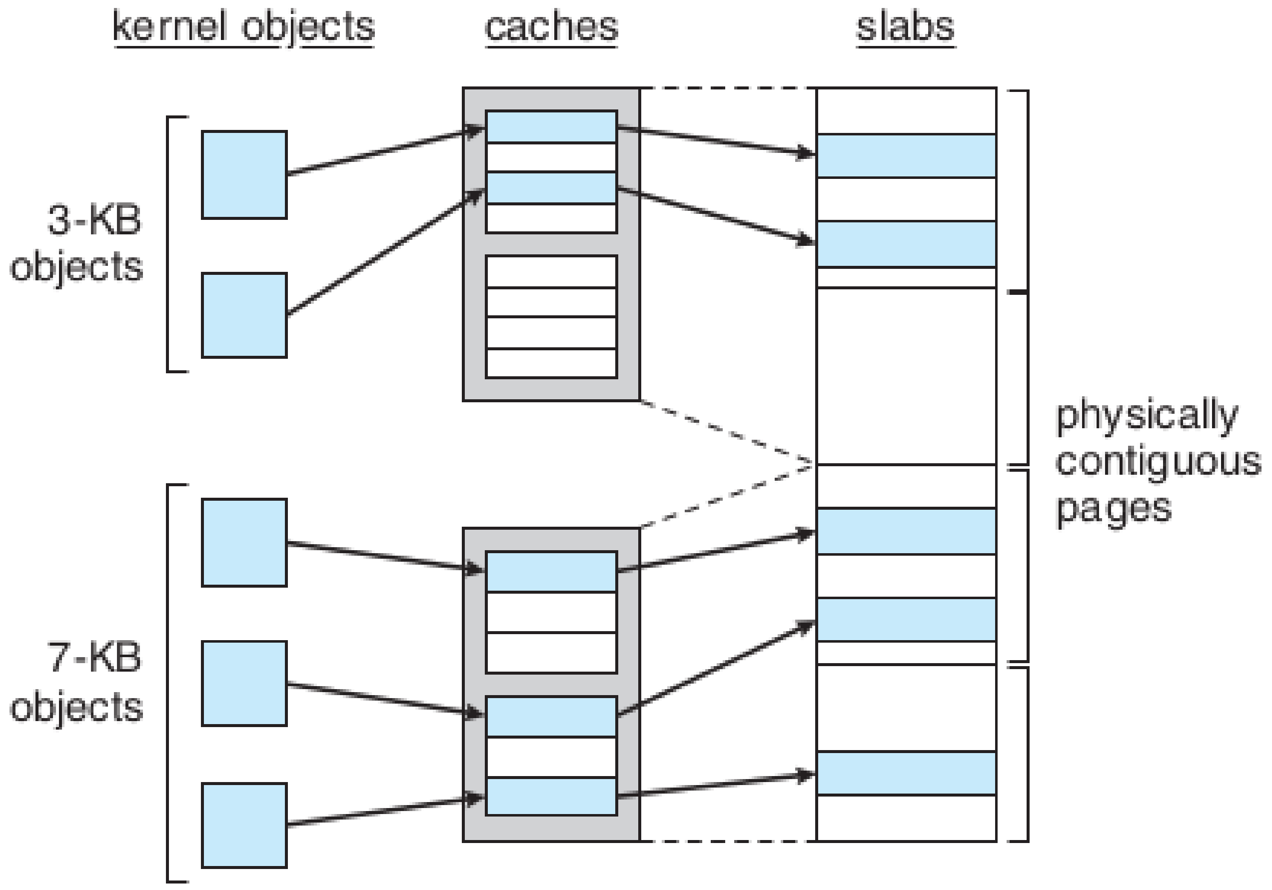
Management of Physical Memory

- Memory management sub-systems:
  - k-malloc() variable-length allocator
  - slab allocator, used for allocating memory for kernel data structures
  - page cache, used for caching pages belonging to files
- Slab
  - used for allocating memory for kernel data structures
  - made up of one or more physically contiguous pages
- Cache
  - consists of one or more slabs - populated with objects that are instantiations of the kernel DS
  - single cache for each unique kernel DS – ex: file objects, inodes, etc.

Management of Physical Memory

- Slab allocation algorithm

- Objects in cache are marked as *free* or *used*
- In Linux, a slab may be in one of three possible states:
  1. Full - All objects in the slab are marked as *used*
  2. Empty - All objects in the slab are marked as *free*
  3. Partial - The slab consists of both *used* and *free* objects



Management of Physical Memory

- Page cache
  - kernel’s main cache for files
  - main mechanism through which I/O to block devices performed
  - file systems of all types perform their I/O through the page cache
  - stores entire pages of file contents and is not limited to block devices

Virtual Memory

- maintains the address space accessible to each process
- creates pages of virtual memory on demand
- manages loading those pages from disk and swapping them back out to the disk as required
- Logical view
  - address space consists of a set of non-overlapping regions
  - linked into a balanced binary tree to allow fast lookup
- Physical view
  - hardware page tables
  - identify the location of each page of virtual memory, on disk or in physical memory
- vm_area_struct - structure that defines the properties of each region

Virtual Memory

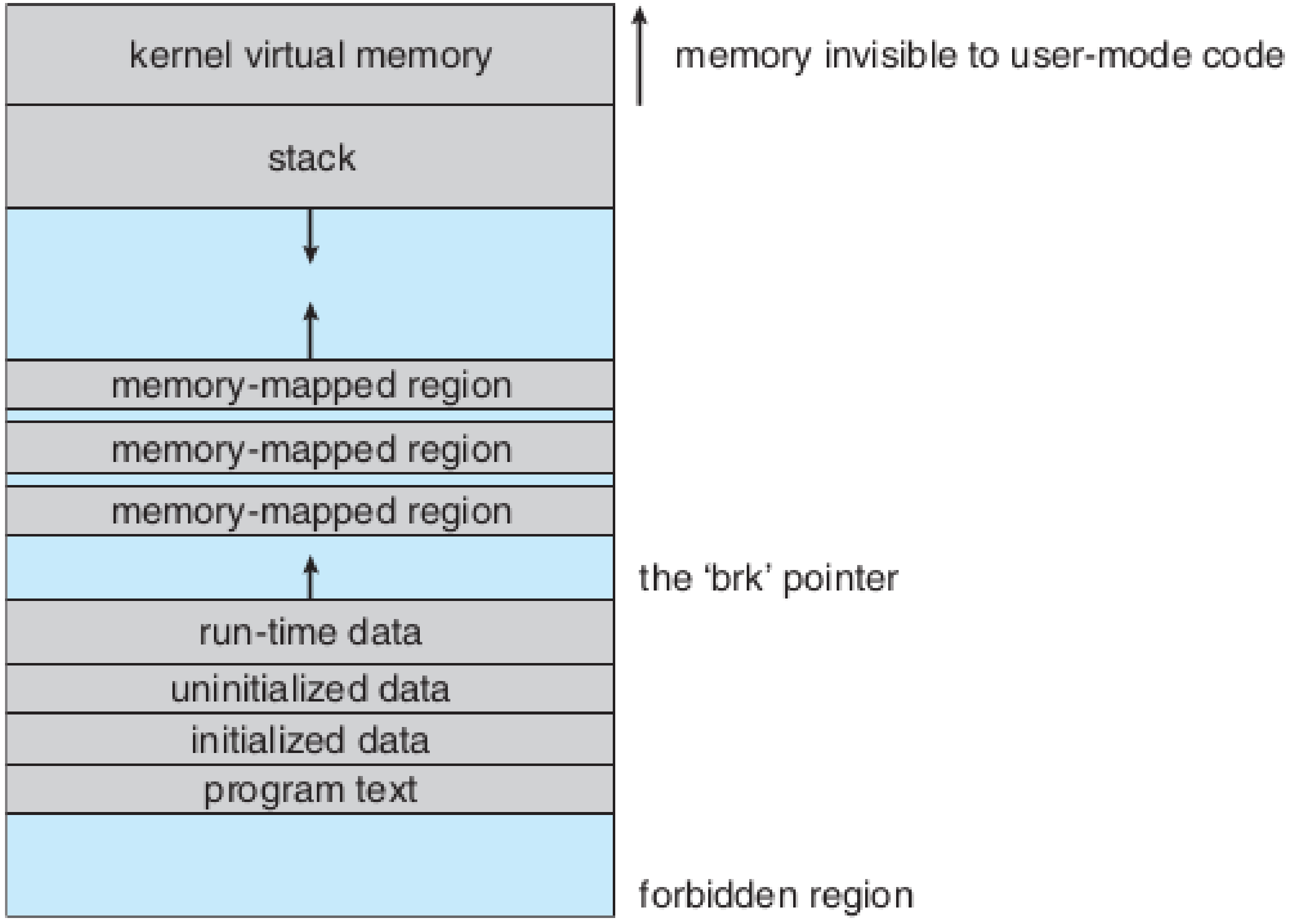
- Virtual Memory region:
  - Backing store; describes where pages come from
  - Demand-zero memory
  - Reaction to writes - private or shared mapping of a region
- Lifetime of a Virtual Address Space:
  - exec()
  - fork()
- Swapping and Paging – paging system:
  - Policy algorithm - decides which pages to write out to disk and when to write them
  - paging mechanism carries out the transfer and pages data back into physical memory
  - Linux's pageout policy – LFU policy (least frequently used)
- Kernel Virtual Memory – for internal use of Linux

Execution and Loading of User Programs

- Older Linux kernels understood a.out format
- Newer – ELF format
- Mapping of programs into memory:
  - ELF format binary file consists of a header followed by several page-aligned sections
  - ELF loader works by reading the header and mapping the sections of the file into separate regions of virtual memory
- Static and Dynamic Linking:
  - Static - necessary library functions embedded directly in the program’s executable binary file
  - Dynamic – stub code; contains small, statically linked function for every dynamically linked program

Mapping of Programs into Memory

- Kernel – privileged region – inaccessible to normal user-mode programs
- Initialise: stack, program's text and data regions
- Stack at top – grows downward
- Pointer (brk) – points to current extent of data region



File Systems

- The Virtual File System
- The Linux ext3 File System
- Journaling
- The Linux Process File System

Virtual File System (VFS)

- The VFS defines four main object types:
  1. An inode object represents an individual file
  2. A file object represents an open file
  3. A superblock object represents an entire file system
  4. A dentry object represents an individual directory entry

- `int open(. . .)` — Open a file.
- `ssize_t read(. . .)` — Read from a file.
- `ssize_t write(. . .)` — Write to a file.
- `int mmap(. . .)` — Memory-map a file.

Virtual File System (VFS)

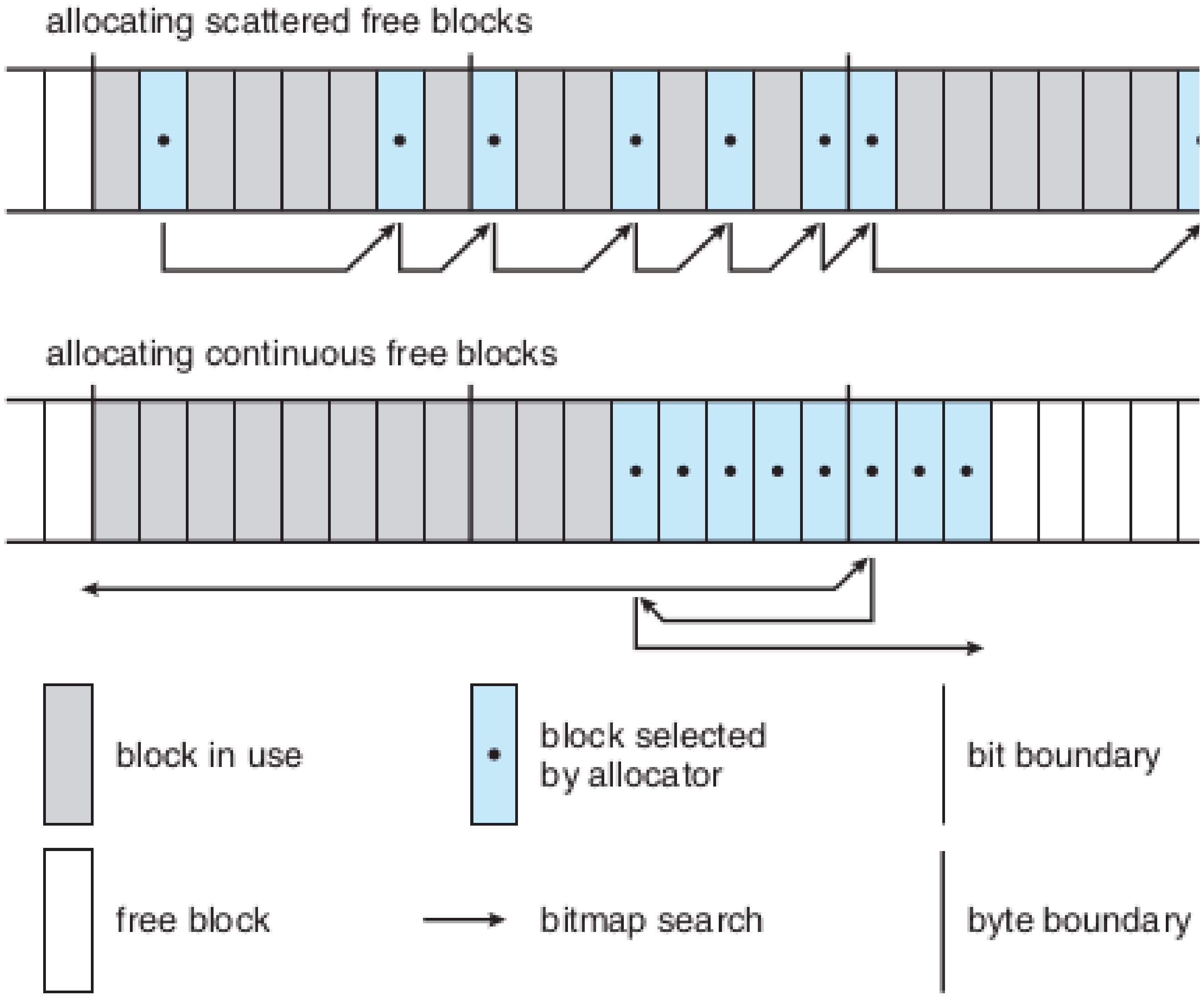
- inode and file objects are the mechanisms used to access files
  - inode object - a DS containing pointers to the disk blocks that contain the file contents
  - file object - represents a point of access to the data in an open file
- There is one file object for every instance of an open file, but always only a single inode object.
- Directory – defines directory or writing data, middle all the tests
- The superblock object represents a connected set of files that form a self-contained file system.
- A dentry object represents a directory entry, which may include the name of a directory in the path name of a file

Linux ext3

- Each block in a directory file consists of a linked list of entries. In turn, each entry contains the length of the entry, the name of a file, and the inode number of the inode to which that entry refers.
- The default block size on ext3 varies as a function of the total size of the file system. Supported block sizes are 1, 2, 4, and 8 KB.
- Allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation.
- Block groups, cylinder groups
- While allocating, try to reduce fragmentation

Linux ext3

- Try to keep allocations physically contiguous
- Search for entire free byte; then search for any free bit
- Once free block identified, search extend backward until allocated block encountered
- Reduces CPU cost of disk allocation by allocating multiple blocks simultaneously



Journaling

- Modifications to the file system written sequentially to a journal.
- When a committed transaction is completed, it is removed from the journal.
- The journal, a circular buffer, may be in a separate section of the file system, or it may even be on a separate disk spindle.

- Advantages:
  - Faster performance of operations
  - Updates proceed much faster when they are applied to the in-memory journal rather than directly to the on-disk data structures
  - Why? - performance advantage of sequential I/O over random I/O

Linux Process File System

- /proc file system
- contents are not actually stored anywhere
- computed on demand according to user file I/O requests
- Must implement two things:
  - A directory structure
  - File contents within
- Mapping from inode number to information type - split into two fields:
  - PID
  - type of information being requested about the process
- maintains a tree data structure of registered global /proc file-system entries

SUMMARY

Scheduling:

- Process Scheduling
  - Completely Fair Scheduler
- Real-Time Scheduling
- Kernel Synchronization
- Symmetric Multiprocessing

File Systems:

- The Virtual File System
- The Linux ext3 File System
- Journaling
- The Linux Process File System

Memory Management:

- Management of Physical Memory
- Virtual Memory
  - Virtual Memory Regions
  - Lifetime of a Virtual Address Space
  - Swapping and Paging
  - Kernel Virtual Memory
- Execution and Loading of User Programs
  - Mapping of Programs into Memory
  - Static and Dynamic Linking

THANK YOU!