Linux

CS 256/456

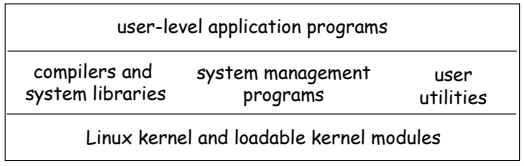
Dept. of Computer Science, University of Rochester

History

- Linux is a modern, open-source operating system that is mostly POSIX-compliant.
- First developed as a small but self-contained kernel in 1991 by Linus Torvalds, with the major design goal of UNIX compatibility.
- Collaboration by many users all around the world, corresponding almost exclusively over the Internet.

Linux Kernel/System/Distribution

- Kernel
  - the OS code that runs on privileged mode
- System
  - essential system components, but runs in user mode
  - compilers, system libraries
- Linux distribution
  - extra system-installation and management utilities
  - precompiled and ready-to-install tools & packages
  - popular distributions: Redhat/Fedora, Debian, SuSE, Caldera, …



Processes and Threads

- Linux uses the same internal representation for processes and threads; a thread is simply a new process that happens to share the same address space as its parent.
  - getpid() semantics?
- A distinction is only made when a new thread is created by the **clone** system call.
  - a process is a task with its own entirely new context (including address space)
  - a thread is a task with its own identity, but not an dedicated address space
- The **clone** system call allows fine-grained control over exactly what is shared between two threads.
  - open files, memory space, page tables

Linux Task Scheduling

- Linux uses two task-scheduling classes:
  - time-sharing and real-time
- A prioritized, epoch-based algorithm for time-sharing
  - Each task has a static credit (default=20) and a dynamic quantum
  - Scheduling is prioritized based on quantum at the beginning of each epoch; each tasks runs its quantum length of time
  - The initial process quantum at its first epoch is credit.
  - An epoch ends when no runnable tasks have any quantum; new quantum is calculated for new epoch

$$\text{initial quantum in new epoch} = \frac{\text{remaining quantum}}{2} + \text{priority}$$

  - This quantum crediting system automatically prioritizes interactive or I/O-bound tasks.

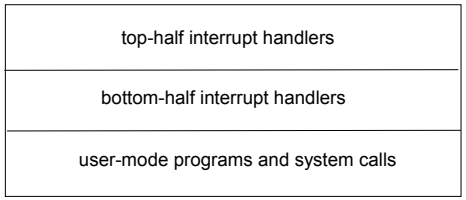
Linux Task Scheduling: O(1) Scheduler

- Linux O(1) scheduler
  - the scheduling overhead is constant, which is independent of the number of processes in the system
- Main operations in Linux scheduler
  - schedule(), epoch transition
- Using two priority arrays
  - one for active array, one for those whole quantum has been used up (called "expired")
  - array index indicates the priority (multiple tasks with the same priority chained in a link list pointed from the array index
- O(1) scheduling
  - fixed number of priorities (bit search instruction like BSFL to speed up).
- O(1) epoch transition
  - swap active and expired arrays.

Interrupt Handling

- Interrupt handling is usually atomic
  - new interrupts are disabled during the handling of an old interrupt, why?
- Linux's kernel allows long interrupt service routines to run without having interrupts disabled for too long.
- Interrupt service routines are separated into a *top half* (urgent) and a *bottom half* (not so urgent).
  - The top half runs with interrupts disabled.
  - The bottom half is run later, with interrupts enabled.
  - Bottom halves run one by one (they do not interrupt each other).

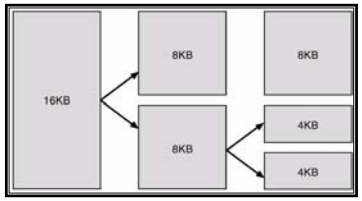
Interrupt Protection Levels



- Each level may be interrupted by code running at a higher level, but will never be interrupted by code running at the same or a lower level.

Managing Physical Memory

- Keep track of free memory?
- Linux page allocator can allocate ranges of physically-contiguous pages on request.
- The allocator uses a *buddy-heap* algorithm to keep track of available physically-contiguous memory regions.
  - A free region list is maintained for each region size: 4KB, 8KB, 16KB, 32KB, … …
  - A large region can be split into multiple smaller regions if necessary



Memory Page Replacement

- All memory pages are managed together
  - stack/heap/code, …
  - file system buffer cache
- Memory pages are managed in two LRU lists: active list and inactive list
  - each LRU list is managed using a CLOCK (second-chance) LRU approximation
  - pages evicted from the active list go to the inactive list; pages evicted from the inactive list are out of the system
  - pages in the inactive list may be promoted to the active list under certain circumstance

Ext2fs File System

- Disks are divided into contiguous block groups
  - the hope is that there is not much seeking within each block group
  - there is a section for inodes in each block group
  - the FS tries to keep inodes and corresponding file blocks in the same block group
- Ext2fs tries to place logically adjacent blocks of a file into physically adjacent blocks on disk
  - with the help of the free block bitmap
- Ext3fs supporting file system journaling

The Linux /proc File System

- The **proc** file system does not store data, rather, its contents are computed on demand according to user file I/O requests.
  - When data is read from one of these files, **proc** collects the appropriate information, formats it into text form and places it into the requesting process's read buffer.
- /proc is not a unique feature on Linux

Prefetching and I/O Scheduling

- File prefetching/read-ahead
  - prefetching sequentially when the I/O access is considered as sequential
  - how to detect sequential pattern?
- Disk I/O scheduling
  - an elevator-style seek-reduction scheduling
  - non-work conserving scheduling: anticipatory scheduling
  - deadline to prevent starvation

Robustness and Dependability

- Modern operating systems are complex and potentially contain bugs
  - Linux is not an exception – including memory errors, synchronization errors (races, deadlocks, …), etc.
- A study [Chou et al. sosp2001] finds that:
  - device drives are 3-7 times more error-prone
  - average bugs live for 1.8 years
  - errors cluster significantly

Disclaimer

- Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).