Process Identity

- *Process ID (PID)*. The unique identifier for the process; used to specify processes to the operating system when an application makes a system call to signal, modify, or wait for another process.
- *Credentials*. Each process must have an associated user ID and one or more group IDs that determine the process's rights to access system resources and files.
  - User ID, Group ID, etc

Process Environment

- The process’s environment is inherited from its parent, and is composed of two null-terminated vectors:
  - The argument vector lists the command-line arguments used to invoke the running program; conventionally starts with the name of the program itself.
  - The environment vector is a list of “NAME=VALUE” pairs that associates named environment variables with arbitrary textual values.
- The environment-variable mechanism provides a customization of the operating system that can be set on a per-process basis, rather than being configured for the system as a whole.

Process Context (1)

- The (constantly changing) state of a running program at any one point in time.
- The *scheduling context* is the most important part of the process context; it is the information that the scheduler needs to suspend and restart the process.
- The kernel maintains *accounting* information about the resources currently being consumed by each process, and the total resources consumed by the process in its lifetime so far.
- The *file table* is an array of pointers to kernel file structures. When making file I/O system calls, processes refer to files by their index into this table.

Process Context (2)

- The *signal-handler table* defines the routine in the process's address space to be called when specific signals arrive.
- The *virtual-memory context* of a process describes the full contents of its private address space.

Scheduling

- The job of allocating CPU time to different tasks within an operating system.
- While scheduling is normally thought of as the running and interrupting of processes, in Linux, scheduling also includes the running of the various kernel tasks.
- Running kernel tasks encompasses both tasks that are requested by a running process and tasks that execute internally on behalf of a device driver.

Process Scheduling (1)

- Linux uses two process-scheduling algorithms:
  - A time-sharing algorithm for fair preemptive scheduling between multiple processes.
  - A real-time algorithm for tasks where absolute priorities are more important than fairness.
- A process's scheduling class defines which algorithm to apply.
- For time-sharing processes, Linux uses a prioritized, *credit-based* algorithm.
  - The crediting rule

    credits := credits/2 + priority

    factors in both the process's history and its priority.
  - This crediting system automatically prioritizes interactive or I/O-bound processes.

Process Scheduling (2)

- For real-time scheduling classes, Linux implements the FIFO and round-robin algorithms; in both cases, each process has a priority in addition to its scheduling class.
  - The scheduler runs the process with the highest priority; for equal-priority processes, it runs the longest-waiting one.
  - FIFO processes continue to run until they either exit or block.
  - A round-robin process will be preempted after a while and moved to the end of the scheduling queue, so that round-robin processes of equal priority automatically time-share between themselves.

Memory Management

- Linux’s physical memory-management system deals with allocating and freeing pages, groups of pages, and small blocks of memory.
- It has additional mechanisms for handling virtual memory, memory mapped into the address space of running processes.

Managing Physical Memory

- The page allocator allocates and frees all physical pages; it can allocate ranges of physically-contiguous pages on request.
- The allocator uses a *buddy-heap* algorithm to keep track of available physical pages.
  - Each allocatable memory region is paired with an adjacent partner.
  - Whenever two allocated partner regions are both freed up, they are combined to form a larger region.
  - If a small memory request cannot be satisfied by allocating an existing small free region, then a larger free region will be subdivided into two partners to satisfy the request.
- Memory allocations in the Linux kernel occur either statically (drivers reserve a contiguous area of memory during system boot time) or dynamically (via the page allocator).

Virtual Memory (1)

- The VM system maintains the address space visible to each process: it creates pages of virtual memory on demand, and manages the loading of those pages from disk or their swapping back out to disk as required.
- The kernel creates a new virtual address space.
  - When a process runs a new program with the exec( ) system call.
  - Upon creation of a new process by the fork( ) system call.

Virtual Memory (2)

- The VM paging system relocates pages of memory from physical memory out to disk when the memory is needed for something else.
- The VM paging system can be divided into two sections:
  - The pageout-policy algorithm decides which pages to write out to disk, and when.
  - The paging mechanism actually carries out the transfer, and pages data back into physical memory as needed.

Virtual Memory (3)

- The Linux kernel reserves a constant, architecture-dependent region of the virtual address space of every process for its own internal use.
- This kernel virtual-memory area contains two regions:
  - A static area that contains page table references to every available physical page of memory in the system, so that there is a simple translation from physical to virtual addresses when running kernel code.
  - The remainder of the reserved section is not reserved for any specific purpose; its page-table entries can be modified to point to any other areas of memory.

File Systems

- To the user, Linux's file system appears as a hierarchical directory tree obeying UNIX semantics.
- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the *virtual file system*(VFS).
- The Linux VFS is designed around object-oriented principles and is composed of two components:
  - A set of definitions that define what a file object is allowed to look like
    - the *inode-object* and the *file-object* structures represent individual files
    - the file system object represents an entire file system
  - A layer of software to manipulate those objects.
    - Vnode/vfs interface

The Linux Ext2fs File System

- Ext2fs uses a mechanism similar to that of BSD Fast File System (ffs) for locating data blocks belonging to a specific file.
- The main differences between ext2fs and ffs concern their disk allocation policies.
  - In ffs, the disk is allocated to files in blocks of 8Kb, with blocks being subdivided into fragments of 1Kb to store small files or partially filled blocks at the end of a file.
  - Ext2fs does not use fragments; it performs its allocations in smaller units. The default block size on ext2fs is 1Kb, although 2Kb and 4Kb blocks are also supported.
- Ext2fs uses 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.

The Linux Proc File System

- In SVR4 Unix, proc file system was introduced as an efficient interface to the kernel’s process debugging support
  - Each subdirectory of the file system corresponds to an active process on the current system and includes a statistic information of the process
- Linux extends the proc file system greatly by adding a number of extra directories and text files under the file system’s root directory
  - New entries correspond to various statistic information about the kernel and the associated loaded driver
  - /proc/1
  - /proc/cpuinfo
  - /proc/devices
  - /proc/kernel
  - /proc/fs

Input and Output

- The Linux device-oriented file system accesses disk storage through two caches:
  - Data is cached in the page cache, which is unified with the virtual memory system
  - Metadata is cached in the buffer cache, a separate cache indexed by the physical disk block.
- Linux splits all devices into three classes:
  - *block devices* allow random access to completely independent, fixed size blocks of data
  - *character devices* include most other devices; they don’t need to support the functionality of regular files.
  - *network devices* are interfaced via the kernel’s networking subsystem

Interprocess Communication

- Like UNIX, Linux informs processes that an event has occurred via *signals*.
- There is a limited number of signals, and they cannot carry information: Only the fact that a signal occurred is available to a process.
- The Linux kernel does not use signals to communicate with processes which are running in kernel mode, rather, communication within the kernel is accomplished via scheduling states and wait_queue structures.

Passing Data between Processes

- The pipe mechanism allows a child process to inherit a communication channel to its parent; data written to one end of the pipe can be read at the other.
- Shared memory offers an extremely fast way of communicating; any data written by one process to a shared memory region can be read immediately by any other process that has mapped that region into its address space.
- To obtain synchronization, however, shared memory must be used in conjunction with another interprocess-communication mechanism.

Network Structure (1)

- Networking is a key area of functionality for Linux.
  - It supports the standard Internet protocols for UNIX to UNIX communications
  - It also implements protocols native to nonUNIX operating systems, in particular, protocols used on PC networks, such as Appletalk and IPX
- Internally, networking in the Linux kernel is implemented by three layers of software:
  - The socket interface
  - Protocol drivers
  - Network device drivers

Network Structure (2)

- The most important set of protocols in the Linux networking system is the Internet protocol suite.
  - It implements routing between different hosts anywhere on the network.
  - On top of the routing protocol are built the UDP, TCP and other protocols.

Thank you