I/O System

The I/O system communicates with the hardware at the lowest level.

The I/O system consists of:

1. General device driver code
2. Device driver code for specific hardware units
3. A block buffer cache for the file systems

The I/O system also defines a common interface to the other parts of the operating system.

An important function for the I/O system is to hide the details in the different hardware units from the main part of the kernel.

UNIX I/O System

There are two main categories of I/O units in UNIX, **block devices** and **character devices**.

In addition there are **sockets** that are used for network communication.

Block devices

- Devices that addresses blocks of a fixed size, usually disk memories.
- Data blocks are buffered in the buffer cache.
- Block devices are usually called via the file system, but are also available as special files (for example /dev/hde1).

Character devices

- Terminals and printers, but also everything else (except sockets) that do not use the block buffer cache.
- There are for example /dev/mem that is an interface to physical memory.

UNIX I/O System

- Device drivers are called via a switch table.
- There is one switch table for block devices and one for character devices.
- A hardware device is identified by its type (block or character) and a **device number**.
- Device numbers consist of two parts: *major device number* and *minor device number*.
- *Major device number* is used as an index in the switch table to locate the correct device driver.
- *Minor device number* is forwarded to the device driver and used to select correct subunit. (For example correct file system partition if the disk is divided in several partitions).

Block Buffer Cache

- The goal for the block buffer cache is to reduce the number of read and write operations to the disk memory.
- If the same data are read several times within a short period of time, only one disk access is needed.
- It often happens that written data are erased within a short period of time. For example temporary files produced by a compiler. These data may never need to be written to the disk if the cache uses long enough delay.
- However using a long delay in writing imposes a risk. If the system crashes - all data that is not written to disk is lost.
- To reduce the data loss in case of a system crash, the command *sync* is run (usually with 30 second intervals) to write all modified cache blocks to the disk.

Raw I/O Interface

- Most block devices also have a character device file.
- Character devices for block oriented devices are called **raw device interfaces**.
- The difference between a block interface and a raw interface is that the raw interface do not use the buffer cache.
- With raw interfaces data is transfered directly from the virtual address space in the process to disk controller.
- Raw interfaces are more efficient when reading large amounts of data one time.

I/O Hardware

Many different types of I/O devices

Two different types of connections:

- point-to-point
- bus

In a typical PC (Fig. 13.1) external devices are connected to a PCI bus.

A controller is an electronic chip or circuit board used to operate a device.

Computers use two different methods to communicate with controllers:

- Special I/O instructions
- Memory mapped registers
- Some systems, for example the PC, uses both methods

I/O Hardware - Polling

Read device register to determine state of device.

If the device is not ready:

- Use a busy-wait loop to wait for the device.

Usually ok for writing to fast devices that give short wait times.

Not good for reading because the waiting may be for ever if no data arrives.

I/O Hardware - Interrupts

- A controller signals the CPU that it needs attention by generating an interrupt.
- An interrupt vector is usually used to direct the interrupt to the correct handler routine.
- There exist several interrupt request lines to request interrupts at different priority.
- The interrupt mechanism is also used to handle exceptions.

I/O Hardware - DMA

- To avoid the need for the processor to copy all data from the controller to the main memory - a DMA (Direct Memory Access) controller can be used.
- A DMA controller contains logic that enables it to control the data bus itself without help from the processor.
- When using a DMA controller - the processor allocates a kernel data buffer and informs the controller to do a data transport.
- The controller copies data from the device (disk memory) to the main memory and generates an interrupt when the transport is finished.

Blocking and Nonblocking I/O

- The UNIX system calls are normally **blocking** - that is they do not return until the requested service is completed.
- By using special options or special system calls also nonblocking I/O is available.
- Nonblocking operations always returns immediately and have a return parameter that reports the result of the operation.
- Nonblocking I/O is needed by processes that wait for data on more than one data channel at the same time.
- The life cycle of a typical blocking I/O operation is illustrated in fig. 13.13

Protection

Objects can be **hardware objects** such as CPU, memory segments and disks or **software objects** such as files and programs.

Protection domain A process executes within a protection domain, which specifies the resources the process may access. Each domain defines a set of objects and the type of operations that may be invoked at each object.

- All access rights in a system can be described by an **access matrix**.
- The rows of the access matrix represents domains and the columns represent objects.
- Each entry in the matrix consists of a set of access rights.
- The complete *access matrix* for a system is usually very big, but most of the entries are empty. For this reason only a part of the matrix is stored.

Access Lists

- Each column of the access matrix can be implemented as an access list for one object.
- The access list consists of ordered pairs <domain, access rights>, which defines all domains with a nonempty set of access rights for that object.
- This method is used by the Andrew file system and by NTFS.
- The UNIX standard file systems use a simplified variant of access list with only three domains (user, group, others) and three access rights (read, write, execute).

Capability Lists

- If the protection is based on the rows in the access matrix, a **capability list** is associated with each domain.
- A **capability list** for a domain is a list of objects together with the access rights for each object.
- A **capability** specifies an access right for a specific object.
- The **possession** of the capability gives access right.
- The **capability lists** are protected by the kernel and are not directly accessible by the processes.
- **Capabilities** are often named by their position in the capability list. This naming method is very similar to the file descriptors in UNIX.
- A capability can usually be transferred from one process to another and in this case the rights associated with the capability is transferred to the new process.

Revocation of Access Rights

With an access-list scheme, revocation is easy:

- Search the access list for the access right and remove or modify it.

Capabilities creates much more problems related to revocation.

Some possible solutions:

- **Reacquisition** - Periodically capabilities are deleted from each domain.
- **Back-pointers** - A list of pointers is maintained with each object, pointing to all capabilities for the object. General solution, but expensive to implement.
- **Indirection** - Each capability points to an entry in a global table that point to the object. When revoking access rights this table is searched and the entry is deleted.