System 101

Maël Auzias

ENSIBS - UBS

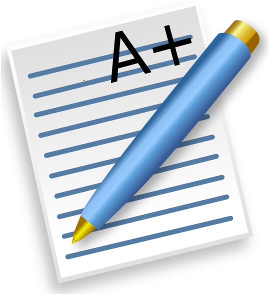
March 2016

Course details

Objectives

- How do *computers* work?
- What are they made of?
- What is an OS?

Course details



Evaluation

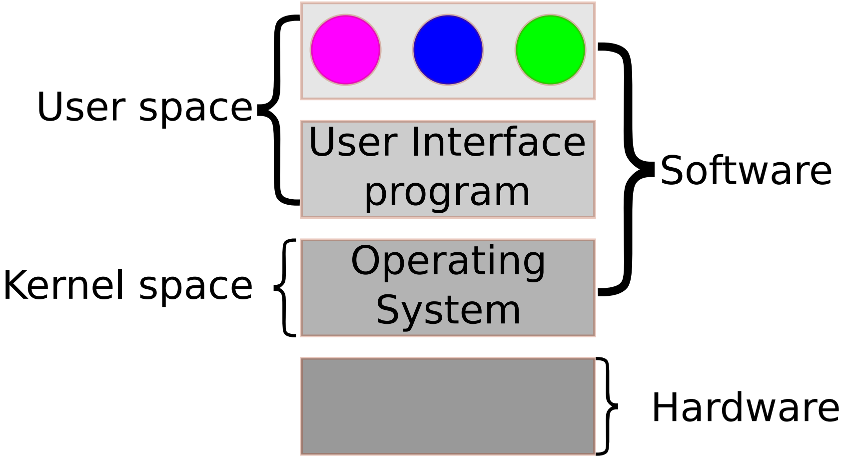
- Short test at the end of each practice
- Final exam (1 hour)
- All equal weighting

Material

- Slides available at sys-101.auzias.net (github too)
- **To read**: Modern Operating System - Andrew Tanenbaum. ISBN-13: 978-0133591620

Presentation Outline

Operating System



Operating System

Two basic unrelated functions

- provide application programmers a clean abstract set of resources,
- manage hardware resources.

Customers

OS real customers are **programs developpers**, not end users of theses developed programs.

The OS as an API provider

Abstraction challenge

- Hardware design, as well made as it can be, only offer awkward and ugly interface to communicate.
- Instruction set, memory organization, I/O, bus structure are not user friendly.
- Programmers need not to worry about all of that thanks to the **abstraction level** provided by OS.

The OS as the resource manager

Resource challenge

- Orderly allocation of the processors, memory, I/O devices for all the programs competing for them.
- Software resources (files, DB, network access) are also managed by the OS.
- Multiplexing:
  - Time multiplexing (CPU, printer),
  - Space multiplexing (RAM, disk).

The OS history

1945-55	First generation: vacuum tubes
1955-65	Second generation: Transistors and Batch Systems
1965-80	Third generation: ICs[1] and Multiprogramming[2]
1980-now	Fourth generation: Personal Computers
future	Fifth generation: any suggestion?

[1] Integrated Circuit
[2] several programs running at once

The OS Zoo

Mainframe Thousands of disks and millions gigabytes of data (high end web servers, servers for business-to-business transactions). Theses OS are focused on executing many jobs at once.

Server Multiple users served at once through a network, they provide print/file/web services.

Multiprocessor Multiples CPUs are hosted into one system (also called, according to what and how they share it: parallel computers, multicomputers, or multiprocessors).

Personal Computers Usually used for game, spreadsheet, word processing and web browsing (laptop, desktop).

The OS Zoo

Handheld	Small computers offering telephony, address book, web apps. They are becoming more and more sophisticated and blurring the difference between personal computers and handheld computers.
Embedded	Microwave ovens, (non-smart) TV and swatches, (not connected) cars, Bluray readers... They usual do not allow user-installed softwares.
Sensor Node	Usually small and simple to run on constraint devices with little RAM/ROM and battery life (TinyOS).
RTS[3]	Industrial process control, avionics, military...
Smart Card	Credit card.

[3] Real-Time System

Presentation Outline

OS Concepts

Overview

- Processes,
- Address spaces,
- Files,
- Input/Output,
- Permissions.

Processes

A process is a program being executed.

Each process:

- has an address space (**core image**),
- has a register (program counter and stack pointer),
- has a list of open files,
- has a list of related processes,
- and all the details needed to run a program.

Process

OS management of Processes

- Execute,
- Save execution state (file pointers list, number of bytes to be read next) in a **process table**[4],
- Stop.

A process corresponds to its **core image** and its **process table entry**.

[4] Array or linked list

Child Process

Process life

- A **system call** starts a process.
- Binary code is executed.
  - The process can create other processes, called **child processes** (and so on – tree).
  - Processes can communicate together using **IPC** means[5].
- The OS may send **alarm** signal (interruption) to the process.
- The process executes a **system call** to terminates itself.

[5] Inter Process Communication

Users Process

Process life

- **UID**[6] is a unique number assigned to each system user.
- Every process started has the UID of the user who started it.
- Every child process has the UID of its parent.
- One UID is called the super-user. The super-user has all permission.
- Users may also be members of groups. Each group has a **GID**.

[6] User Identification

Address Space

An address space is a memory location (from 0 to some maximum) and contains:

- executable program,
- program's data,
- program's stack.

Address Space

- Physical memory,
- Virtual memory (swap).

File

OS hide all peculiar disk operations to offer abstracted model of device-independent file management.

- System calls are required to:
  - Create, remove, read and write files; create and remove directories.
- File system also match a tree structure.

File

- Path
- Several files may have the same name.
- Each file has a unique absolute path (and an infinity of relative ones).
- Mounted file system, merging trees.

Special Files

I/O devices are abstracted to be used through same system calls as files do.

- Devices:
    - Block files,
    - Character files.
    - Special files are kept in /dev.
- Pipe:
    - IPC mean,
    - a special system call needs to be performed to known it's not a real file.

Tree

- Both process and file are structured as tree.
- Process tree are usually not very deep, unlike file trees.
- Process hierarchy are usually short-lived (minutes or less) while directories may exist for years.
- Ownership and protection differs too.

Permissions

u g o : user group other

- Three 3-bit fields
  - read
  - write
  - execute
- rwx rwx rwx do-what-ever-you-want-file
- rwx rwx r-x web-file
- rw- rwx rwx virus
- r-x — — personal-backup.tgz

Right	File	Directory
r	can read	can list files
w	can write	can add/delete files
x	can execute	can go through

Figure: Permissions meaning

Boot procedure

The BIOS[7] contains the procedures to read the keyboard, write to the screen, perform I/O on disk. Held in RAM, OS can modify it when bugs are found.

1. The BIOS is started,
2. The BIOS checks how much RAM is available,
3. The BIOS verifies if keyboard, mouse (and other basic devices) are correctly installed and responding,
4. The BIOS scans PCI[8] and ISA[9] buses to detect attached devices,
   4.1 If new devices are found since last boot, these new devices are configured.

[7] Basic Input Output System: low level I/O software system program present on the parentboard

[8] Peripheral Component Interconnect

[9] Industry Standard Architecture

Boot procedure

5. The BIOS determines the boot device (floppy? CD-ROM? DVD? USB? ...?) by using a list of devices stored in CMOS memory,
6. The program contained at the first sector of the boot device is executed,
   6.1 It usually examines the partition table at the end of the boot sector to determine which partition is active.
   6.2 A secondary boot loader is read from that partition.
   6.3 This loader reads the operation system from the active partition and starts it.
7. The OS queries the BIOS to get configuration details,
8. The OS checks the driver of each device,
9. The OS loads all these modules into the kernel,
10. The OS initializes its table, creates background processes and starts up a login program.

Presentation Outline

Processes and Thread

Overview

- Why processes are so important?
- What differences between processes and threads?

Processes

- Most important abstraction,
- Turn single-CPU into multiple virtual CPU
- Enable pseudo concurrent operations (pseudoparallelism),
- Without, modern computing could not exit.

Processes

Processes

- are instance of executing program
- include currents values of
  - program counter,
  - registers,
  - variables.
- have their own virtual CPU (multiprogramming) – considerations about time management, RTS.

The student partying (a fictional analogy)

- The student, at a home party, makes a cocktail.
  - Student: CPU, recipe: program, drinks: data, glasses: resource, action: process.
- While pouring the last ingredient her/his phone rang and s/he answers.
  - Student: CPU, phone-skill: program, phone: resource, phone call details: data, action: process.

A process is an activity having a program, input, output and a state. OS uses scheduling algorithm to determines when to stop/start which process.

Processes

Creation

- System initialization,
- Process creation done by a running process,
- User request,
- Initiation of a batch jobs.

Termination

- Voluntary
  - Normal exit,
  - Error exit.
- Involuntary
  - Fatal error,
  - Killed by another process.

Process states

State

a. Running,
b. Ready,
c. Blocked.

Transition

1. Scheduler pick another process.
2. Scheduler pick this process.
3. Input available.
4. Input required.

Threads

Threads

1. Processes within a process.
2. Enable to decompose big task into multiple sequential smaller tasks ..
3. .. while **sharing** a memory space.
4. Easier, faster, to create and destroy than processes as they are lighter.

Web browser example with multiple threads.

Threads

Concurrent programming: race condition (example)

- Thread S, and N, respectively correspond to the South, and North, gate in a Park.
- They count the current number of visitors:
  - When a visitor enters, the counter is incremented by one.
  - When a vistor leaves, the counter is decremented by one.
- Both threads share a common variable, the counter.
- They run on the same computer and, thus, share a sole CPU.
- To increment a variable at least three operations are required:
  1. Read the variable.
  2. Compute the incrementation.
  3. Write the new value of the variable.

What could go wrong?

Presentation Outline

System calls

System operations

- Read file, create directory or a process, modify permissions ..
- .. results in a system call.

System calls

System calls procedure

1. Switch from the program, within the user space, into the kernel space by executing a TRAP instruction.
2. The program starts the execution at a **fixed** address in the kernel space.
3. The kernel code then examine the system call number and execute the matching system call handler.
4. The system call handler execute its code and then returnes the control to the program at the instruction following the TRAP instruction.
    - unless the system call blocks (i.e., waiting for an input on the keyboard).
5. The program is put back into user space and continues.

Main system calls: Process management

Call	Description
pid = fork()	Create a child process (identical to the parent)
pid = waitpid(pid, &stat, opt)	Waif for a child to terminate
s = execve(name, argv, envp)	Execute a program
exit(status)	Terminate a process and return status

Figure: Process management

Main system calls: File management

Call	Description
fd = open(file, flags)	Open (or create) a file
close(fd)	Close a file
n = read(fd, buff, nbytes)	Read data from a file into a buffer
n = write(fd, buff, nbytes)	Write data from a buffer into a file
p = lseek(fd, offset, whence)	Reposition the pointer within the file
s = stat(name, buff)	Get file status

Figure: File management

Main system calls: Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a directory
s = rmdir(name)	Delete a directory
s = link(oldpath, newpath)	Make a new name for a file
s = unlink(path)	Delete a name and possibly the file it refers to
s = mount(s, t, fst, f, d)	Mount a filesystem
s = umount(target)	Unmount a filesystem

Figure: Directory and file system management

Main system calls: Miscellaneous management

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since 01/01/70

Figure: Miscellaneous

Presentation Outline

Memory Management

Historically, no abstraction at all:

- RAM: OS (from 0 to 0xN) and user program (to 0xN to 0xMAX)[10]
- RAM: user program (from 0 to 0xMAX), ROM: OS
- RAM: OS (from 0 to 0xN) and user program (to 0xN to 0xMAX), ROM: device drivers (see fn 10)

Usually, only one sole program could run at a time. Unless...

[10] These are dangerous: if a bug is present in the program, it can wipe the OS

No abstraction, several user programs

- Swapping: the constraint is that only one program can be in the RAM at once. The OS can save into a file the whole memory and load up another back-up program state.
- Divide and conquer: IBM 360 split memory with a protection key to identify (and match) memory and programs.
  - Static relocation: modify on the fly the program when it's loaded (i.e., JUMP 28 into JUMP 28+constant).

Memory abstraction: address spaces

Problems to solve

- Protection
- Relocation

Address space?

- IP address
- Phone numbers
- Internet domain (.com, .net)

Memory abstraction: address spaces

Each process address space is mapped onto a different part of the physical memory.

Base and limit CPU registers

- Base register: loaded with the physical address where the process begins.
- Limit register: loaded with the length of the process.

Every time a memory reference is done, the CPU add the value contained in the base register, checks if it is equal or greater than the value in the limit register and cases a fault if needed.

Managing Free Memory

- bitmaps: memory is divided as allocation units (from a few words to several kB). Each unit is map to a bit (1 if used, 0 if free).[11]
- linked lists: memory is a whole block. When a process starts the process' address space is removed from the list till the process ends. The list is sorted by address. Several algorithm to allocate memory are: first fit, next fit, best fit[12], worst fit. This can be speed up with two lists: one for processes, one for holes – and holes-list can then be sorted by size making best fit/first fit faster. Holes list can also be stored... in holes!

[11] smaller units allow a more precise allocation but the map consumes more memory

[12] A bit wasteful as it tends to leave unusable tiny free spaces

Virtual Memory

Each program has its own address space, broken up into chunks called pages. These pages are mapped to physical memory, but not only.

Paging

Virtual addresses[13] form the virtual address space. When used, they go to an MMU[14], instead of the memory bus, that maps the virtual addresses onto the physical memory addresses. It aims to facilitate memory overlay.
A page fault occurs when a process references an address that is not stored in the memory.

[13] program-generated addresses
[14] Memory Management Unit

Page Replacement Algorithms

- optimal[15]: remove the last referenced page. The OS has no way of knowing when each of the pages will be referenced next.
- not-recently-used: randomly removed one page of the lowest-numbered non empty class (classed thanks to R (rreferenced-flag) and M (modified-flag)):
  - Class 0: not referenced, not modified
  - Class 1: not referenced, modified
  - Class 2: referenced, not modified
  - Class 3: referenced, modified

[15] easy to explain, impossible to implement on *first* run

Page Replacement Algorithms

- FIFO: First-In, First-Out (can throw out a heavily used page).
- second-chance: to not throw out a heavily used page, R-flag is inspected on the oldest pages. The algorithm seek for old pages that has not been referenced in the most recent clock interval.
- clock: circular-list contains the pages, the oldest is always pointed. If its R-flag is 1, it is cleared and the next is pointed, when a page with R-flag cleared is found it is replaced.

Page Replacement Algorithms

- least-recently-used: recently heavily used page will probably be reused and pages not used for ages probably won't be. Longest unused page is replaced. Heavy algorithm: the list of pages must be updated at every memory reference. Special hardware does exist to do so.
- working-set: pages are loaded only on demand, not in advance. A process having a working-set won't cause page fault until it moves into another execution phase.
- WSClock: based on the clock algorithm and the working-set.

Page Fault Handling

1. The hardware traps to the kernel saving the program counter.
2. ASM code is started to save volatile information[16].
3. The OS discovers a page fault and seek for the needed page.
4. Check if the address is valid and the protection. If not, process is killed. Is a page frame free? No: page replacement algorithm is run to select a victim.

[16] such as registers

Page Fault Handling

5. The found page is dirty, it is scheduled for transfer. Another process is run, in any event the page is marked as busy.
6. The page is clean, the OS schedules a disk operation to bring the new page in.
7. The disk interrupt indicates the new page arrival. Page tables are updated, the page is marked as being in normal state.
8. Fault instruction is backed up the state it had when it began, the program counter is reset to point to that instruction.
9. Faulting process is scheduled, the OS return to the routine that called it.
10. The routine reloads the registers and other state information and returns to user space to continue execution.

Segmentation

Segmentation

Division into different segments (or sections) of different types (code, data, array, stack).

- Each segments is contiguous.
- Memory is then address by two-part address: segment number and address within the segment.
- Segments simplify growing and shrinking data structures handling and sharing.
- Each segment can have a different protection.

Presentation Outline

File System

Challenges

- How to store very large amount of data?
- How to store it after *t*he process has been killed?
- How to allow multiple processes to access it concurrently?

File System

Two operations

- read_block(block_number)
- write_block(block_number)

File System

Two operations

- read_block(block_number)
- write_block(block_number)

How to, with these two, ..

- Find information?
- Keep a user from reading another user's data?
- Know which blocks are free?

File Abstraction

OSes abstract processor into process, physical memory into virtual address space and file system into files.
Processes (Threads), address space and files are the three most important abstractions that an Operation System offers.

File Naming

Naming rules

- Different rules for different OS.
- Names length varies from 1 to 255 characters.
- Case-sensitive (UNIX), or not (MS-DOS).
- Extension consideration.

Files Structure

File Structure

1. Unstructured sequence of byte (any meaning is imposed by the user procces). Offers the maximum flexibility.[17]
2. Fixed length-records each with internal structure (80 characters records, 132 char for line printer).[18]
3. Tree records, each files has a key value, tree sorted according to these values.[19]

[17] UNIX, MS-DOS, Windows
[18] Not used anymore
[19] Still used in some commercial data processing

Files Types

File Types

- Regular files (ASCII or binary files).
- Names length varies from 1 to 255 characters.
- Case-sensitive (UNIX), or not (MS-DOS).
- Extension consideration.

Binary File Example

Executable binary example:

- header,
  - Magic-number (identifying the file as executable),
  - Size of the various pieces of the file,
  - Address at which execution starts
  - Various flags.
- text,
- data,
- relocation bits,
- symbol table (for debugging purposes).

Binary File Example

Archive example (collection of library procedures compiled but not linked):

- header (telling its name),
- Creation date,
- Owner,
- Protection code,
- Size.

Files Extension and OS

- An OS must at least recognize its own executable file type (they usually recognized more than this one type),
- TOPS-20 checked the creation date of executable file, sought for its source file, and recompile it if the source file was updated.
- *make* program reproduce this behavior, file extensions are mandatory.
- Strong mandatory file extensions make an OS unusable (e.g., all extension file output produced are ".dat" – impossible to copy a file!)

File Access

- Sequential access: files were read from the beginning to the end (magnetic tape).
- Random access files: files can be read in any order. To choose the position in the file and read you can:
  - Give the position as a parameter at teach read operation,
  - Seek at a specific position and then sequentially read (UNIX and Windows).

File Attributes (Metadata)

- Right: who and how can the file be accessed.
- Password: required to access the file.
- Creator: ID of the user who created the file.
- Owner: ID of the user who owns the file.
- Read-only flag.
- Hidden flag.
- System flag.
- Archive flag (set by the system when changed, cleared by the back-up program).

File Attributes (Metadata)

- ASCII/binary flag: 0 for ASCII file, 1 for binary file.
- Random access flag: 0 for sequential access only, 1 for random access.
- Temporary flag: 0 for normal, 1 for delete file on process exit
- Lock flag: 0 for unlocked, nonzero for locked.
- Creation time.
- Time of last access.
- Time of last change.
- Current size.

File Operations

- Create: while no data are put into, the file is announced and some of its attributes are set.
- Delete: frees up some disk space (sys-call).
- Open: the system fetchs attributes and list of disk addresses into main memory for rapid access later on.
- Close: frees up internal table space. Some OS impose a maximum number of open files.
- Read: the user must specify how many data are expected and a buffer to put them in.
- Write: data are written. If position is the end of the file, the file size increases, if it is in the middle, existing data are overwritten.
- Append: write-like but only at the end of the file.

File Operations

- Seek: for random access files, set the file pointer at the requested position.
- Get attributes: many programs need it, i.e.*make*.
- Set attributes: allows the user to modify some of the file attributes.
- Rename: simple as that.

GH.com/p-e-w/maybe

Directories

- Single-level directory system: only **one** folder (digital cameras, music players, first PCs).
- Hierarchical directory system.

Path Names

- Absolute (from **root** directory): */etc/apt/source.list*, */var/log/messages*.
- Relative (from **current** directory):
  - Implicit: *Download/Gramatik.zip*, *../Movies/*.
  - Explicit: *./Download/Gramatik.zip*, *./../Movies/*.

What are dot and dotdot?

Directory Operation

- Create: creates a directory with dot and dotdot.
- Delete: deletes a directory (only empty directories can be deleted).
- Opendir: opens a directory so it can read.
- Readdir: reading a directory allows to list all files within it.
- Closedir: frees up internal table space.
- Rename: simple as that.
- Link: (hard link) increments the file's i-node counter and allows to make the file appear in different directories.
- Unlink: decrements the file's i-node counter. If it is zero the file is delete from the system, otherwise only the path name specified is removed (UNIX system call to delete a file is unlink).

File System Layout

- Sector 0 contains the MBR[20], used to boot the computer, at the end of the MBR is the partition table.
- Partition table contains starting and ending disk addresses of the different partitions. One is marked as active.
- At boot time, the BIOS reads and executes the MBR, the MBR program locates the active partition, reads its first block (boot block) and execute it.
- The boot block loads the OS contained in that partition.
  - All partition contains a boot block, even if it does not contain a bootable OS.

[20] Master Boot Record

Partition System Layout

- Boot block: cf previous slide.
- Superblock: details include a magic number to identify the file system type and the number of blocks in the file system.
- Free space management: can be in a form of a list of pointers.
- I-nodes: array of data structure, one per file.
- Root directory: contains the top of the system tree.
- Files and directories: remainder of the disk.

Implementing Files

How to keep track of which disk blocks match with which file?

- Contiguous allocation,
- Linked list allocation,
- Linked list allocation using a table in memory,
- I-nodes[21].

[21] Index-nodes

Implementing Files: contiguous allocation

Each file occupies consecutive disk blocks.

Advantages

- Simple to implement:
  - Keeping track of a file's block requires only two numbers: starting disk address, number of blocks.
- Read performance is excellent (the entire file can be read from the disk in a single operation).

Drawbacks

- Fragmentation.
- Disk space.
- What about a file becoming bigger?

Implementing Files: linked list allocation

Each file is matched with a linked list of disk blocks.

Advantages

- Every block of the disk can be used (no waste, no fragmentation).
- Directories entry only need to store the disk address of the first block (no matter how big are the files).

Drawbacks

- Performance
  - Many head rotation may be required.
  - Amount of data storage in a block is not a power of two.
- Each block starts with the address of the next block

Implementing Files: linked list allocation using a table in memory

Let's put the pointers and put them in a table in memory (FAT).

Advantages

- Every block of the disk can be used (no waste, no fragmentation).
- Random access is much easier as no disk references are needed (as these references are in the memory).

Drawbacks

- Memory usage. The whole table must be in memory all the time.
  - 200-GB disk, 1-kB block size, entries of 3 bytes: 600 MB.

Implementing Files: i-nodes

The index lists the attributes and disk addresses of the file's blocks.

Advantages

- The memory is occupied only when file is open.
- The footprint is smaller, the RAM usage does not depends on the disk size (but the number of opened files).

Indexes have the same size, thus, for big files, the last disk address point for an address of a block containing more disk block addresses.

Implementing Directories

Main function of directories: map the ASCII name of the file onto the information needed to locate the data.

- Directories entry provide the needed details to find file's disk blocks
  - Disk address of the entire file,
  - Number of the first block,
  - Number of the i-node.

Implementing Directories: File Names' Length

How to deal with file names' length?

- Fixed-length
  - MS-DOS: 1-8 character base name and optional extension of 1-3 characters.
  - UNIX v7: 1-16 characters, including any extensions.
- Variable-length
  - Set a maximum limit and use it for the design.
  - In-line.
  - In a heap.

Shared Files

- Disk blocks are listed in a data structure[22] associated with the shared file, not in directories.
- Symbolic linking.

[22] the i-node in Linux

Other File Systems

- Log-Structured File System (LFS).
  - Computer speed bottleneck is the hard drive speed. reads are OK, writes are buffered and done by burst.
- Journaling File System (NTFS, ext3).
  - The OS keep a log of what the FS is going to do before it does it, so if a crash occurs, on the reboot it can repair it.

Virtual File Systems

A VFS integrates multiple file systems into an orderly structure.

- /: ext4 on an SSD.
- /media/cd-rom: ISO 9660.
- /home: NFS.
- /usr: NTFS on a HDD.

While there are 4 different FS, users access any file with the standard POSIX calls.

Defragmentation

Free blocks become scattered all over the disk among the used blocks. Big files are then stored in many places that slows read-calls.

Questions

1. Why a magic number is giver for executable files whereas random ones are given to other files (UNIX - old versions)?
2. Is *open* required? Is there a way to avoid this and used *read* or *write* without calling *open*?

Presentation Outline

I/O

OS objectives

- Issue commands to the devices
- Catch interrupts
- Handle errors
- Provide an easy-to-use interface
- Provide common interface for all devices

I/O Devices

- block devices
  - stores information in fixed-size blocks.
  - each one with its own address.
  - Transfers are in units of entire consecutive blocks[23].
  - Each block can be read or write independently.
  - Examples: USB sticks, CD-ROM, hard drives.
- character devices
  - delivers/accepts a stream of characters.
  - is not addressable.
  - cannot seek.
  - Examples: printers, mice, network interfaces.

All devices do not fit in this two-part categories. Clocks (that cause interrupts at intervals) or memory-mapped screens do not fit in.

[23] ranging from 512 B to 32kB

I/O data rates

Device	Data rate
Keyboard	10 bytes/s
52x CD-ROM	7.8 MB/s
USB 2.0	60 MB/s
Gigabit Eth	125 MB/s
SATA disk drive	300 MB/s
PCI bus	528 MB/s

Table: Devices and their data rate

Memory-Mapped I/O

Each device has an electronic component called the device controller.

Each controller has a few registers, used to communicate with the OS through the CPU.

Memory-Mapped I/O

Two means of communication

- Each control register is assigned an I/O port number.
  - overhead to read/write control registers.
  - device driver must contain ASM.
  - read operations require more instructions, slowing the responsiveness.
- Each control register is assigned a unique memory – this is called memory mapped I/O.
  - overhead to read/write control registers.
  - device driver can be written in C.
  - protection is easy (the OS just don't allocate that space to user programs).
  - Each device can (and should) have its own memory page.
  - no caching possible.

DMA[24]

Having the CPU request one byte at a time for each device is wasteful.
A DMA has access to the system bus while being CPU independent.

Without DMA

1. the disk controller reads the block bit by bit till the whole block is in the controller's internal buffer.
2. the checksum is compute.
3. the controller causes an interrupt.
4. the OS can then read the buffer word by word and store it in the memory.

[24] Direct Memory Access

DMA

With DMA

1. the CPU programs the DMA controller through the DMA registers.
2. When valid data are in the disk controller's buffer the DMA controller ..
3. .. initiates the transfer with a *read* request.
4. When the write (from buffer into memory) is complete, the disk controller sends a signal to the DMA controller.
5. The DMA controller, once all the *read*-operations are done, interrupts the CPU.

DMA

DMA differences

- Some DMA can handle several transfers at once, each controllers having one channel for each sets of internal registers.
- Some DMA can operate in word-at-a-time mode and block mode.
- In block mode the DMA can tell the device to acquire the bus and perform a series of transfers, then release the bus. This is called "burst mode".

If the CPU needs the bus it has to wait. This mechanism is called "cycle stealing".

DMA

Disk read. Why this internal buffer?

- To compute checksum.
- Without, when transfers start, the disk would have to write into the memory. What would happen if the data bus is busy? Words would be lost.

Not all computer have DMA. With fast CPU, it is cheaper to deal with transfers on software level and also faster than wait for a slow DMA.

Interrupts

1. A device causes an interrupt by asserting a signal on its assigned bus line.
2. The interrupt controller detects this signal and decides what to do:
    - No other interrupts are pending: the interrupt is processed immediately.
    - Another interrupt is in process or higher-priority one has be requested: the interrupt is ignored for the moment.
3. The interrupt controller write a number on the address lines to specify the device causing the interrupts and sends a signal to the CPU.

Interrupts

4. The CPU stops what it was doing, read the number used as an index into a table called the interrupt vector to fetch a new program counter.
5. The program counter points to the start of the interrupt service procedure.
6. From this point, traps and interrupts use the same mechanism and usually share the same interrupt vector.
7. After the interrupt service procedure starts, the interruption is acknowledged so the controller can fire another interrupt.

By delaying the acknowledgement a race-condition is avoided.

Interrupts

Information saved before starting the service procedure

- Program counter,
- The program counter points to the start of the interrupt service procedure.
- From this point, traps and interrupts use the same mechanism and usually share the same interrupt vector.
- After the interrupt service procedure starts, the interruption is acknowledged so the controller can fire another interrupt.

Precise Interrupt

Four properties characterize a precise interrupt:

1. The program counter is saved in a known place.
2. All instructions before the one pointed to by the PC have been fully executed.
3. No instruction beyond the PC has been executed.
4. The execution state of the instruction pointer to by the PC is known.

If an interrupt does not meet these properties, it is called an imprecise interrupt.

I/O Software

Key concepts

- Device independence: a program that reads a file should not have a different procedure to read a file form the disk, or a USB stick, or the keyboard.
- Uniform naming: file name should be a string and should not depend on the device.
- Error handling: errors should be handle as close as the hardware as possible.

I/O Software

Key concepts

- Synchronous blocking: the OS should make I/O operations look like blocking as it is easier to program.
- Buffering: network packet or audio streams, just like many other, need to be buffered. The OS should take care of that.
- Sharable/dedicated device: while some I/O devices can be used by many users at the time (NIC, HDD), some cannot (printer). The OS must handle the stemming issues.

I/O Software

Performing an I/O

- Programmed I/O: the CPU does all the work so it's simple, but the polling, or busy waiting, makes it is costly.
- Interrupt-driven I/O: the process requesting the I/O operation is blocked until the operation is done, unless the CPU that can execute other process waiting for the next interruption coming from the device. Usually an interruption occurs at every character and takes time to be executed.
- I/O using DMA: the DMA feed the characters to the device to let the CPU free. Interrupt number is down from one per character to one per buffer.

Presentation Outline

Definition by example

Two processes, *P0* and *P1*, needs to request to two resources, *R0* and *R1*. *P0* and *P1* and independent and differently programmed.

1. *P1* requests, and obtains, access to *R0*.
2. *P0* requests, and obtains, access to *R1*.
3. Reads/Write-operations are performed on *R0* by *P1* and *R1* by *P0*.
4. Now *P1* needs access to *R1*, owned by *P0*, while *P0* needs access to *R0*, owned by *P1*.
5. Both processes will wait the other to release the needed resource.

That's a dead**lock**.

Former definition

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.

Resources

Resources types

Any object[25] that must be acquired, used and released.

- Preemptable resource: resource that can be taken away from the process without bad effect (e.g., disk, memory).
- Nonpreemptable resource: resource that, if taken away, will fail the processing of a task.

Steps to use a resource

1. Request the resource.
2. Use the resource. (optional)
3. Release the resource.

[25] either hardware or software

Deadlock

Deadlock conditions, Coffman et al. (1971)

1. Mutual exclusion condition. Each resource is either currently assigned to exactly one process or is available.
2. Hold and wait condition. Processes currently holding resources that were granted earlier can request new resources.
3. No preemption condition. Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.
4. Circular wait condition. There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain.

Deadlock

Deadlock conditions, Coffman et al. (1971)

1. Mutual exclusion condition. Each resource is either currently assigned to exactly one process or is available.
2. Hold and wait condition. Processes currently holding resources that were granted earlier can request new resources.
3. No preemption condition. Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.
4. Circular wait condition. There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain.

Deadlock modelisation

- Mutual exclusion[26] condition. Each resource is either currently assigned to exactly one process or is available.
- Hold and wait condition. Processes currently holding resources that were granted earlier can request new resources.
- No preemption condition. Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.
- Circular wait condition. There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain.

Schematic Representation

- *P0* requests the resource *R0*: $(P0) \longrightarrow [R0]$
- *R0* is owned by *P0*: $[R0] \longrightarrow (P0)$

Deadlock

1. *A* owns R and requests S.
2. *B* owns nothing but requests T.
3. *C* owns nothing but requests S.
4. *D* owns U and requests S and T.
5. *E* owns T and requests V.
6. *F* owns W and requests S.
7. *G* owns V and requests U.

Is there a deadlock?

Deadlock Prevention

Mutual Exclusion

Assign a resource when that is absolutely necessary. Make sure that as few processes as possible claim the resource.

Hold and Wait

All resources are requested before the program starts. Problems stemming from this: 1. a program may not know the needed set of resources, 2. resources will not be optimally used.
Another way to break this is: a process requesting a resource first temporarily releases all the resources it currently holds, then tries to get everything it needs all at once.

Deadlock Prevention

No preemption

Virtualizing a resource can prevent deadlock. For instance a printer will not be assigned to a process but rather to its daemon, and the daemon will be assigned some disk space where processes write to print. Nevertheless, not all resources can be virtualized.

Circular wait

If a process can own at most one single resource this condition is eliminated. But what if a process need to print a huge file sent through a serial communication? It costs too much.
A solution to avoid circular wait is to order resources. If *P0* and *P1* must request *R0* and *R1* in this order then no deadlock would had occurred.

Deadlock Prevention

Two-Phase Locking

1. the process tries to lock all the resources it needs, one at a time.
   - On succeed: goto 2
   - If one lock fails: release all locks, goto 1.
2. The process performs its updates and releases the locks. No real work is done in the first phase.

Deadlock Issue

Communication

Deadlocks appears also in communication system. A node S requests a resource to node R and blocks, waits, for the response. What if the request is lost?
This is a communication deadlock. They cannot be prevented by ordering resources or avoided by scheduling. Timeout can prevent them.

Starvation

When several process requests the same resource, processes may starve if new processes, arriving endlessly, always get the resource before them.
Starvation can be avoided by using a first-come, first-served, resource allocation policy.

I hope you liked it and learnt something new !

Figure: sys-101.auzias.net