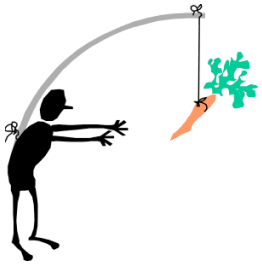
Real-Time Programming
TI00AA55

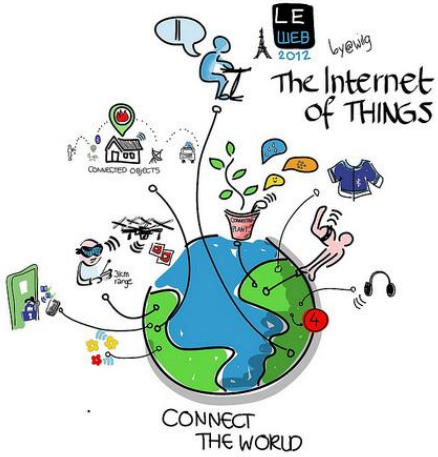
Lecture 1 - 14.01.2015

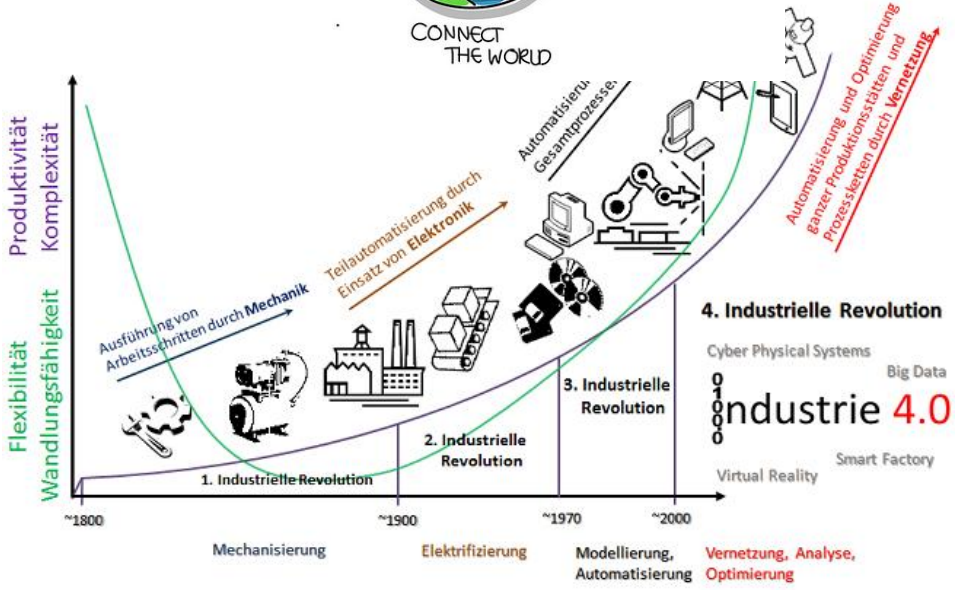
Jarkko.Vuori@metropolia.fi

Motivation



- “Internet of Things”, Internet of Everything (IoE)
  - More than 30 billion devices will be wirelessly connected to the Internet of Things (Internet of Everything) by 2020 (ABI Research, https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne)
  - Large amount of those devices have something “real” to control (room thermostat, car engine, etc.) and therefore their control systems have real-time response requirements
- Industrie 4.0 (Internet of Things in manufacturing)
  - Customized devices can be manufactured like mass-products
    - Merging of service and manufacturing
  - Manufacturing equipment must be flexible, and therefore they need to control (in real-time) larger portion of the manufacturing process
- Real-Time systems are becoming pervasive
  - Air Traffic Control Systems
  - Networked Multimedia Systems
  - Command Control Systems
  - Combustion Motor Controllers
- In a Real-Time System the correctness of the system behavior depends
  1. not only on the logical results of the computations,
  2. but also **on the physical instant at which these results are produced**





General information

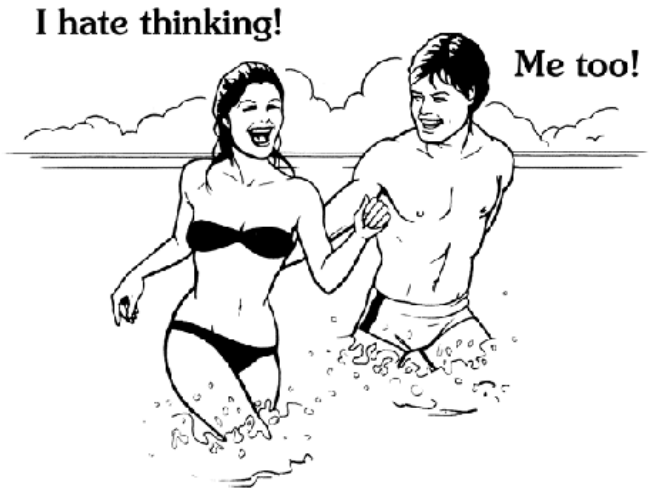
- WinhaWille system is used to enroll the course
  - You'll get to the WinhaWille system from the Metropolia web pages (www.metropolia.fi)
  - The identification code of the course is TI00AA55
  - You need to enroll the course in order to get the course material from the Tuubi
- Practical aim of this course
  - You learn to understand how real time applications are working
  - You learn to program real time applications in practical terms
  - You learn these things according the POSIX-standard that is applicable in many operating systems
  - You learn program development practices in Unix/Linux-platform
  - Students learn to understand how the computer system works as a whole
- Official aims of this course
  - Understanding principles of real-time programming
- Intended Learning Outcomes (upon successful completion of the module):
  - Knowledge and Understanding
    - The student will be able to
      - Understand multitasking problems (e.g. race conditions, deadlocks)
      - Understanding methods to avoid those multitasking problems
      - Analyze existing real-time programs
  - Professional Skills
    - The student will be able to
      - Create multitasking programs with no race conditions and no deadlocks

General information

- Prerequisites:
  - **C programming language (mandatory)**
    - Pointers, structures, prototypes, defined types
  - **Basics of Unix/Linux (mandatory)**
    - Basic of Linux environment
  - **Algorithms and Data Structures course (recommended)**
    - Algorithms, ADT, trees, dynamic memory, recursion
  - Linux & gcc programming environment
- This course is located to periods 3 and 4
- The extent of the course is 6 op
  - 2 h lectures/ one week (2×7 times, 28 h together)
  - Lab. work 2 h / week (2×7 times, 28 h together)
  - Self study (42h)
  - One lab group only
- Exam at the end of the course (2h)
  - Note that there is no exam at the end of the third period
  - If you don't pass the exam, you can attend a resit exam after the course exam. For the resit exam dates, look for the Tuubi-portal's workspace (Tuubi>For student>Important dates>re-examination dates>Technology (for Finnish: Tuubi>Opiskelijoille>Aikataulut>Uusintakoepäivät>Tekniikka, työtila TiVi Leppävaara: Opinto-ohjaus) folder Documents/4. UUSINTATENTIT for the exact resit exam dates
    - You have to fill the request (resit envelope) 10 working days before the resit exam
    - You can go to two resit exams at maximum, e.g. you can try only two times to pass the exam (after the course's main exam)
    - This course belongs to the resit exam group A1
- Supporting books (not mandatory):
  - Richard Stevens and Stephen Rago: Advanced Programming in the UNIX Environment, Addison-Wesley, 2005
  - Kay A. Robbins & Steven Robbins: UNIX Systems Programming (Communications, Concurrency and Threads), Prentice-Hall, 2003
  - Marc J. Rochkind : Advanced Unix Programming, Addison-Wesley, 2004
- Optional books (not mandatory, but for those more interested in theoretical aspects of real-time systems)
  - Liu: Real-Time Systems, Prentice-Hall, 2000
- All the course material (lecture slides, lab works, example program files, extra material) are available from the Tuubi (tube.metropolia.fi)

General information

- We have 13 lab assignments
  - You get one credit point from one exercise
    - ±0.25 can be added to the credit point depending how bad/good you have performed
  - Some labs contain also optional additional exercises
    - In order to get additional credit points
    - Note: optional labs must be done without any help from the instructor
    - Often optional exercises are more difficult
  - You must complete the assignment in the lab hours
    - If for some reason you could not complete the lab assignment, you can show it in the next lab session (one week later)
    - If you return lab late, you will get an exponentially decaying grade, e.g. $(1/2)^n$, where n is the delay in weeks (after the one week period)
    - Optional labs cannot be returned late at all, sorry
- The final grade of the course is
  - min(5, round((*exam*-8)/(10/3,5)+1 + 0.3×(*labs*-14))),
    - Where *exam* is the points of the exam (minimum 8 is needed to pass, 18 points is the maximum), and *labs* is the credit points from the labs

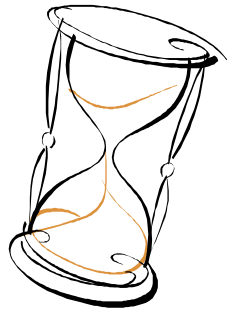


[T]here are known knowns; there are things we know we know.
We also know there are known unknowns;
that is to say we know there are some things we do not know.
But there are also unknown unknowns –
the ones we don't know we don't know.

Former United States Secretary of Defense Donald Rumsfeld

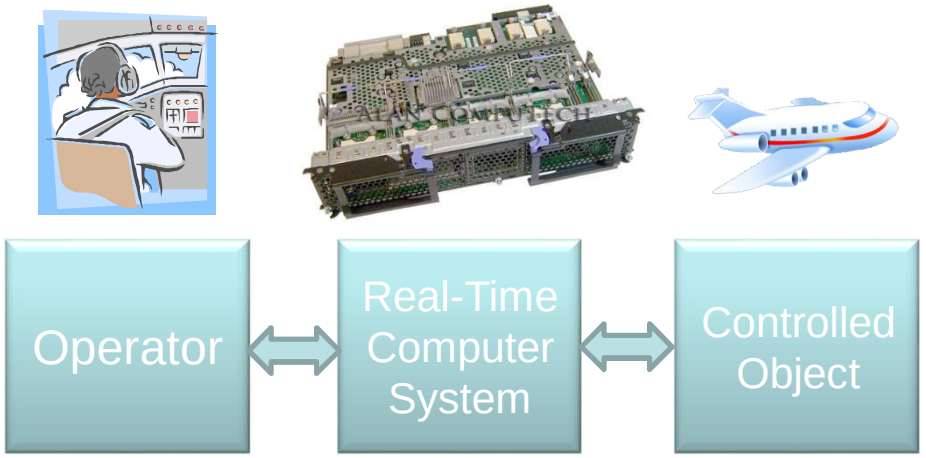
Timetable

Lecture	Topic
1	Intro, definition of Real-Time System, Program development in Linux/Unix-environment
2	Implementation of Real-Time systems, Polling, Processes, operating modes
3	System calls, I/O at system call level, Atomic operation
4	Standard I/O library, buffering
5	Process handling
6	Parent/Child relation in processes, Zombies
7	Signals (generating, receiving, buffering, masking)
8	Asynchronous and multiplexed I/O
9	Pipes and FIFOs
10	Sockets and streams
11	Real-Time signals, Timers, Clocks
12	Threads and thread synchronization
13	New features on POSIX
14	No lecture, labs only



Real-Time systems

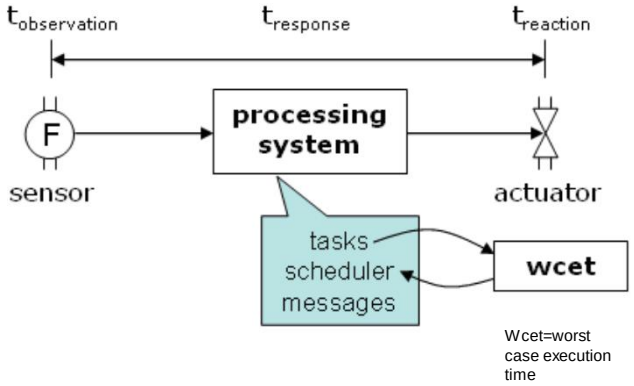
- A computer system always is a part of a larger system and has interaction with the surroundings
- Real-time feature means that computer gives responses to the input from the environment in time limits required by the system
- In some systems the response time requirements can be very strict and damages extensive, if the computer does not fulfil these requirements (many process control systems for example)
  - Also in modern games, the picture must be ready for the next frame; otherwise the video stream lags
- Real time requirements can be different in strictness (hard real time systems, soft real time systems)
- ”The problem” arises, when computer has several task to do and there are several events what to respond to (can be even hundreds or thousands)



Real-Time System

What is Real-Time System?

- As opposed to a common belief, “real time” doesn't mean “fast response time” or “low latency”
- Rather, it means, among other things, **a totally deterministic operation mode**
  - If you execute the same database query twice, each time it will take a different amount of time to execute, depending on various extraneous factors such as the number of users currently connected to the database, system load, the distance of the disk sector that contains the database records from the read/write magnetic head, the size of available memory etc. In other words, the response time of a database query is not deterministic
    - hence it's not a real time operation
- Many programming languages are unsuitable for real time programming due to their non-deterministic nature
  - Languages that have a built-in garbage collector (e.g. Java, C#) are very problematic because the garbage collector might “wake up” at the wrong moment, halting all other operations until it finishes



Examples of Real-Time systems

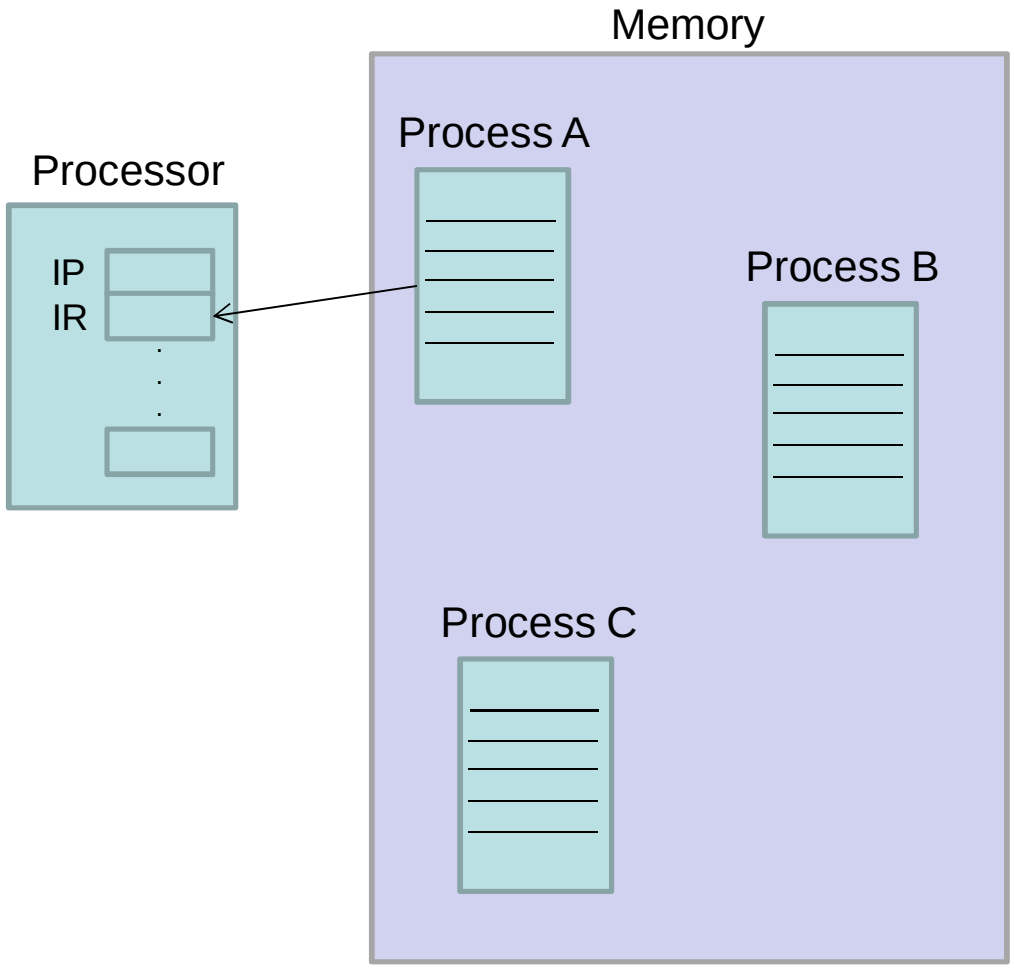
- The example list of real-time system could be almost infinite:
  - Word processing system (especially multi-user systems)
  - Web servers
  - Reservation systems (ticket-, hotel-, travel-, etc.)
  - Process control systems (monitoring and control systems for power plant, paper machine, packing line, and so on)
  - Telephone exchange
  - Air conditioning control systems
  - Inventory systems
  - Digital video playing systems
  - Etc.

Basis for Real-Time functioning

- Concurrency is typical feature in real-time system
- This means that several programs (processes) are run “simultaneously” although the computer usually has only one processor
- In computers, that have many processors (multi processor systems), programs are run genuinely in parallel (parallel processing)
  - We concentrate mainly in the situations where the computer has only one processor
- Although the processor can execute only one program at a time, it is possible to make everything to look and feel like processor is running several programs simultaneously
- The reason for this is that the processor is very fast compared to the time scale of the environment
  - It can execute very many instructions in a short time measured in the time scale of the environment (for example time scale of human beings)
  - This means that it is possible to share processor time for many programs in turn
  - It is also worth of noticing that input and output (to and from peripherals) is very slow measured in the time scale of processor
    - This is one more reason why it is useful to share processor time and execute other programs when one program has initiated i/o and is waiting it’s completion
- Example of time scale differences:
  - In 2 GHz processor the instruction cycle is 0,5 ns
  - The access time of typical disk can be 7 ms
  - In “human terms” these are 1 second and 162 days

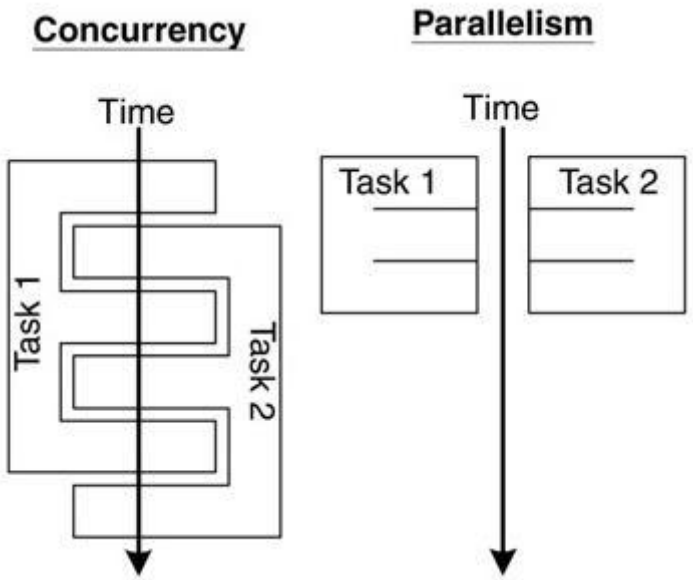
Concurrent processes

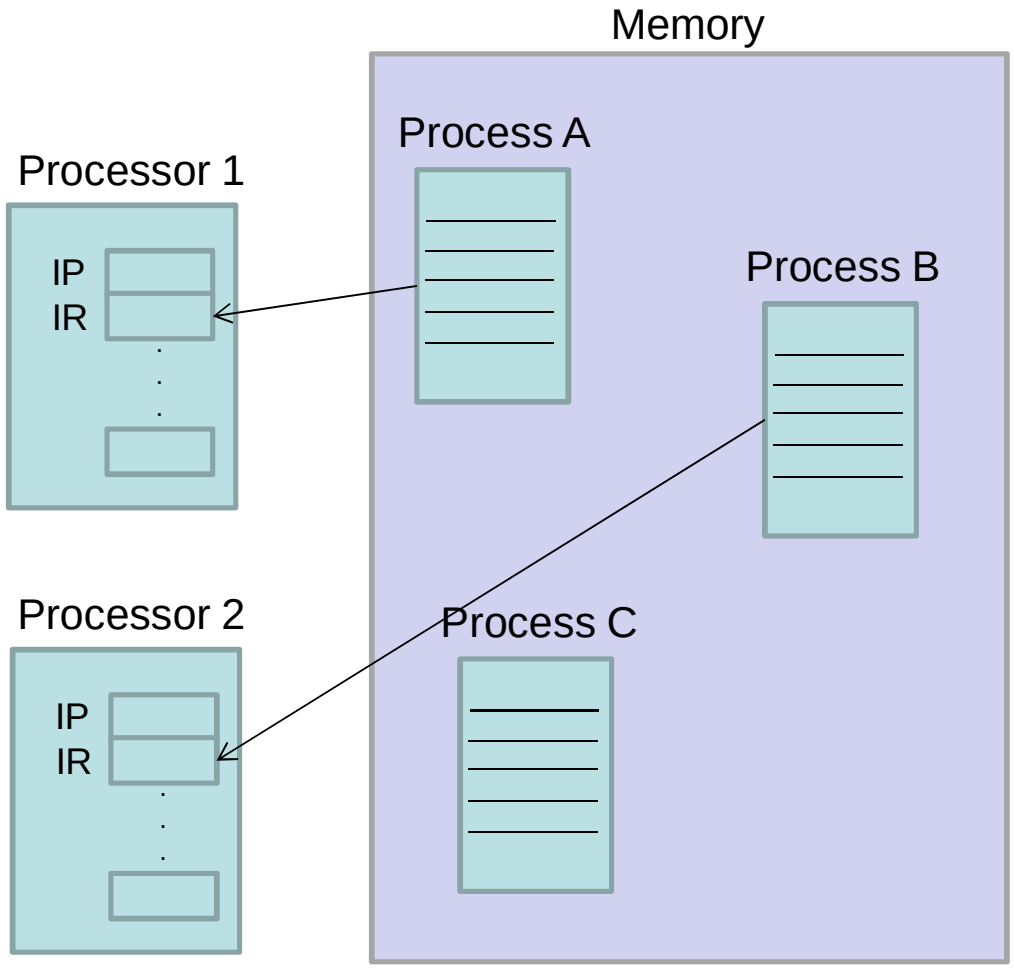
- In computing, a process is an instance of a computer program that is being executed
  - It contains the program code and its current activity
- The operating system supports real-time behavior by sharing processor's time for different processes (tasks)
- The picture illustrates how concurrent processes are run in the processor
  - One-by-one



Concurrency vs. Parallelism

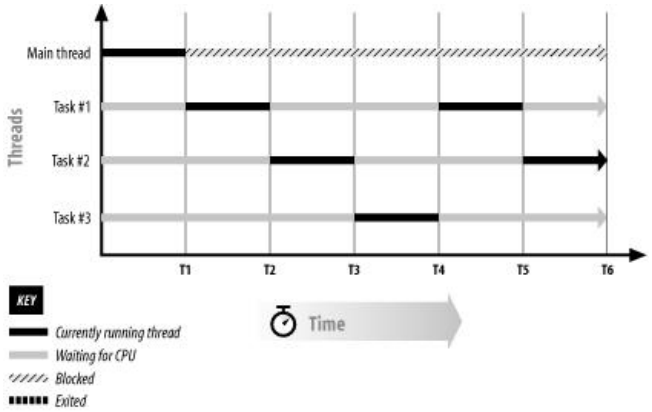
- Today it is normal to have multiple processors in one installation
  - Then we have real parallelism (vs. concurrency with only one processor)

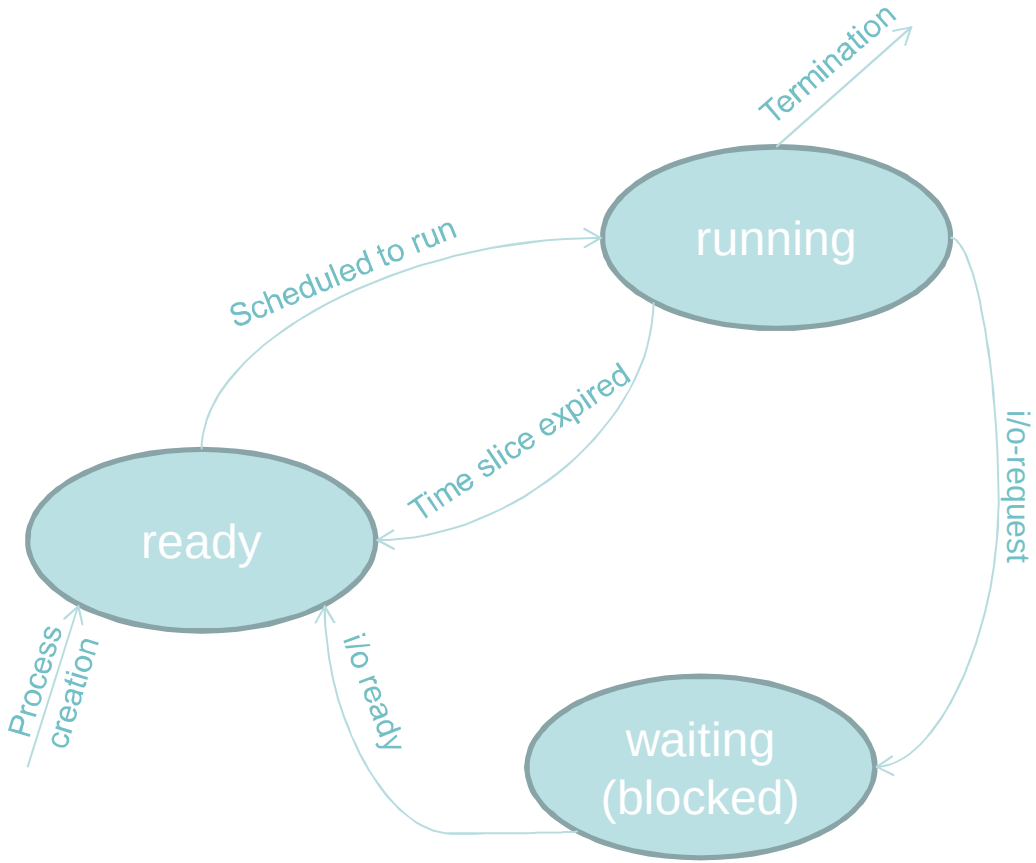




Process states

- When a program is run it is a process under the control of the operating system
- The state diagram of this kind a process is described on the right

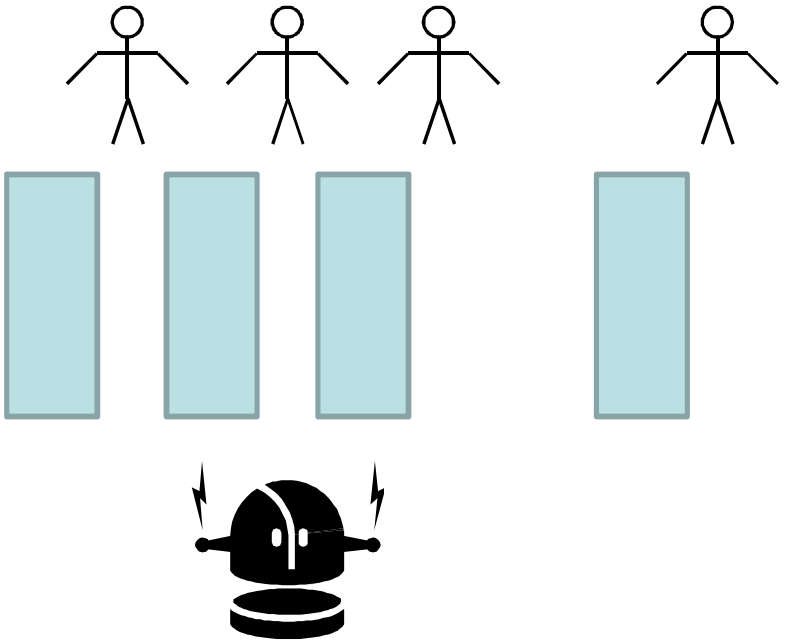




Practical illustration

- As we already learned, the computer needs to give immediate answers at the same time (in the time scale of the surrounding world) to several inputs. How this really is possible can be illustrated using one robot that takes care of all cash points in the store
- In the situation described above all customers feel that they receive immediate service all the time, even there is only one cash robot

The properties of customers (people):
Moves goods from the basket to the belt 1 pc / 2 s
Take a wallet and money from the pocket in 2 s



The properties of the robot:
Move from one cash point to another in 1 µs
Reads the bar code from one product in 1 µs
Takes a payment from the customer in 1 µs

Operating systems

- Real time systems can be implemented without any operating system at all
  - Some kind of “elementary operating system” is implemented in the context of project
- Many real time operating systems exists for different processors that support implementation of real time systems
- Examples:
  - POSIX: QNX, OS X, RTLinux
  - Others: AMX, DeltaOs, µC/OS-II, eRTOS

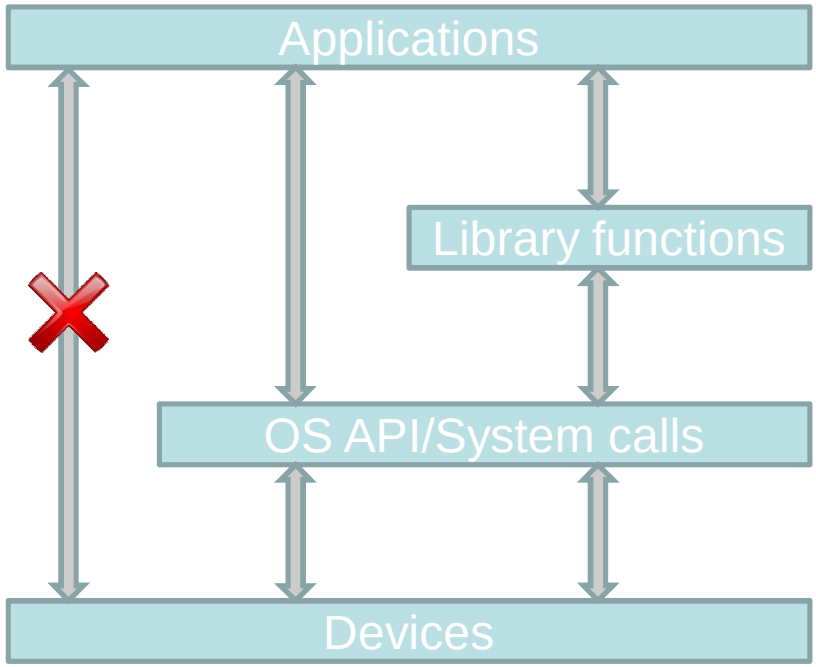
- Services of operating system
  - File and directory services
  - Memory management
  - Process management
  - Device management
  - Network management
  - Time management

POSIX standard

- There has been a tendency to standardize operating system interfaces
- Many competing standards existed earlier
- Now there is: ”Single Unix Specification, Version 3”
  - IEEE Std 1003.1- 2001(2004;2008)
  - POSIX (Portable Operating System Interface)
    - The last letter X in POSIX means Unix-like systems
  - http://pubs.opengroup.org/onlinepubs/9699919799/
- New standard consists of four parts:
  - Base Definitions volume (XBD)
  - System Interfaces volume (XSH)
  - Shell and Utilities volume (XCU)
  - Rationale (Informative) volume (XRAT)
- Earlier the division was different:
  - POSIX.1 Basic OS interface (C language)
  - POSIX.1a Misc. extensions (symlinks, etc.)
  - POSIX.1b Real-time and I/O extensions (was: POSIX.4)
  - POSIX.1c Threads (was: POSIX.4a)
  - POSIX.1d More real-time extensions (was: POSIX.4b)
  - POSIX.1e Security extensions, ACLs (was: POSIX.6)
  - POSIX.1f Transparent network file access (was: POSIX.8)
  - POSIX.1g Protocol independent communication, sockets (was: POSIX.12)
  - POSIX.2 Shell and common utility programs
  - POSIX.3 Test methods
  - POSIX.7 System administration

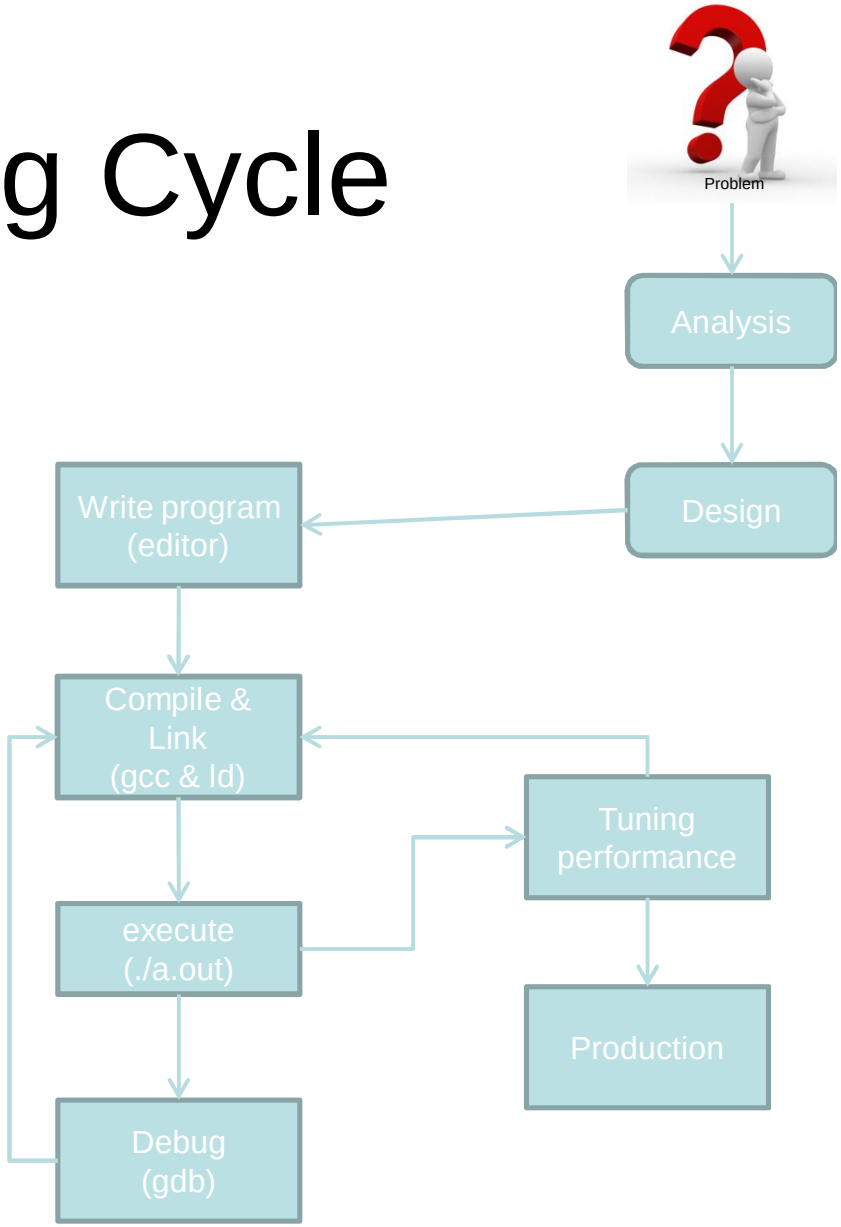
Hierarchy levels of computer system

- The figure on the right describes the hierarchy levels of computer systems
- Applications cannot use directly the devices from the device level
- They must use the API (Application Programmers Interface) level of the operating system
- The POSIX standard described on the previous page defines this API level
- Our main focus in this course is on the API interface



Programming Cycle

- First we have the problem
- After the Analysis (of the problem) and Design (of the solution) phases, we can start the programming work
- Compiler produces object files (.o in Linux, .OBJ in Windows) from the source files (.c if using C programming language)
- Linker must be used to combine these object files together with libraries in order to get the executable program



Program development tools

- As we have learned earlier, the things are handled according the Posix-standard and Linux-operating system
- The lab exercises can be done with what ever Linux/Unix machine
- In the laboratories you can use remote Edunix machine via terminal connection (e.g. using PuTTY) or virtual Linux machine Fedora that is running on the local Windows machine
- If you choose to use Edunix-machine via terminal connection you need to manage without X-window system, meaning that you need to rely on the command line tools like mentioned on the right

- Editors
  - You can choose yourself
  - Emacs
  - ViM
  - Jed
  - Pico
  - gEdit
  - and so on
- Compiler GCC
  - GNU Compiler Collection
  - Supports C and C++
  - Supports other languages too
  - http://gcc.gnu.org/
- Other auxiliary programs

Program development tools

- Linker ld
  - gcc calls linker, so that usually it is not necessary to call it explicitly
- How to compile and run a simple program:
  - Compilation (executable to the file program.exe):
    - ...$ gcc -o program.exe program.c
    - If you omit '-o' option, the result filename will be a.out
  - Running:
    - ....$ ./program.exe
    - If you want to execute the program.exe without writing './' prefix, you need to modify your .bash_profile file:

```
# User specific environment and
startup programs

PATH=$PATH:$HOME/bin:"."

export PATH
umask 077
```

- Other auxiliary command line tools:

gdb	GNU debugger
ar	To manage libraries
as	Assembler
make	To manage programming projects consisting of several parts and building phases

- If you use Fedora virtual machine the graphical integrated development environment Eclipse is also available for you

Summary of tools

1. Terminal connection to Edunix
   - Do all things using command line tools in Edunix (also editing)
2. Terminal connection to Edunix
   - Build and run the program from the command line in Edunix.
   - Edit the program using the editor in the local Windows machine (eg. notepad++)
3. Edit the program in the virtual machine Fedora using for example gedit
   - Compile, link and run the program using command line tools in terminal window in the same Fedora virtual machine
4. Use integrated development environment Eclipse in the virtual machine Fedora

- All these ways are possible to use in the lab B114/0.143
- Remark. You can work also at home
  1. using PuTTY to connect to the edunix.metropolia.fi Linux computer
  2. with the computer that has Linux installed (for example Ubuntu etc.)
  3. with windows machine by installing VMPlayer and loading the Fedora virtual machine from the network of the school
  4. using Mac-computer that have Mac OS X operating system