

1. Introduction

- *Leaking abstractions*
- Structure of a mobile device
 - Hardware
 - Basic software concepts and run-time infrastructure
 - Software stack and installation
- Summary



Example: Null-terminating string

```
char * strcat(char * c1, char * c2)
{
    int i, j;
    while(i = 0; 0 != c1[i]; i++);
        while(j = 0; 0 != c2[j]; j++, i++)
            c1[i] = c2[j];
    c1[i+1] = 0;
    return c1;
}
```



Leaking abstraction

- Leaking abstraction reveals the details of an implementation in some special case
 - TCP/IP when connection is terminally broken
 - SQL can be terribly slow in some queries that are poorly organized
 - Run-time infrastructure can behave in a complex fashion
 - ...
- Are there leak-proof abstractions?



Leaking abstractions and mobile devices

- Scarce resources
 - Memory, processor, bandwidth, disk, etc.
 - Implementation becomes visible more easily as there is only limited reserve
- Resource allocation responsibility
 - White-box resource management
 - Black-box resource management
- In the end, the programmer still holds all the strings in both approaches
 - Infrastructure's effect must be taken into account!



White-box resource management

- Assumes an educated developer who always anticipates the implementation and performs necessary actions in any resource allocation
 - Avoids leaking abstractions by giving control and responsibility to the programmer
- Patterns (and antipatterns) of resource management
- Coding style, idioms and standards
- Example systems
 - C
 - C++
- Errors and their effect?



Black-box resource management

- Assumes an infrastructure that allows and lives with any allocation strategy for the programmer
 - However, leaking abstractions of the infrastructure remain a problem that must be solved by the programmer
- Example systems:
 - Java
 - C#
(Btw., when do Java and/or C# abstractions leak?)
 - Scripting languages (Python, JavaScript)
 - Web runtime (e.g. widgets)
- Error cases?



Content and goals

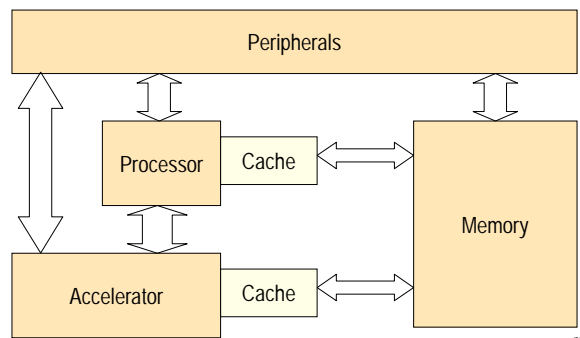
- Leaking abstractions
- *Structure of a mobile device*
 - *Hardware*
 - Basic software concepts and run-time infrastructure
 - Software stack and installation
- Summary

Mobiiliohjelmointi, kevät 2009

7



Sample Hardware



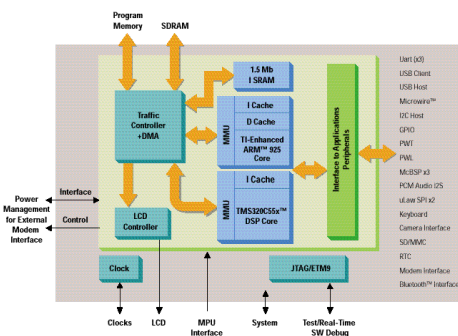
Mobiiliohjelmointi, kevät 2009

8



Another Sample Hardware

OMAP1510 Application Processor for 2.5 and 3G Wireless Devices



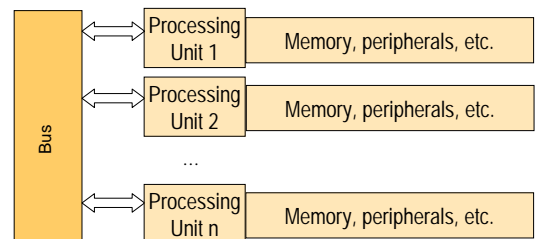
Mobiiliohjelmointi, kevät 2009

9



Nokia Terminal Architecture (NoTA)

(http://en.wikipedia.org/wiki/Network_on_Terminal_Architecture)



Mobiiliohjelmointi, kevät 2009

10



Hardware – Components

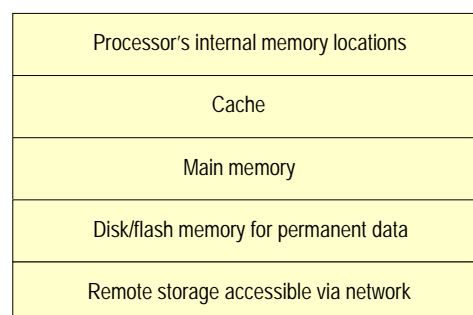
- Processor executes instructions that manipulate memory
 - ARM
- Accelerators (DSP, custom) provide enhanced computation facilities
- Memory stores program and data (RAM: 32Mb-128Mb; 64Mb a common figure, ROM: 32Mb-64Mb; ever increasing)
 - ROM, RAM; Flash (NOR flash is direct memory space, NAND flash is like a disk)
 - Memory management unit (MMU) helps in organizing memory usage
 - Paging becoming an option as the size of the flash disk increases; some devices use real hard drives
- Peripherals provide an access to the outside world
 - Bluetooth, radio interface, screen, additional memory, battery, ...

Mobiiliohjelmointi, kevät 2009

11



Memory hierarchy



Mobiiliohjelmointi, kevät 2009

12



Memory Allocation: What Is Allocated Where?

- Constants
 - ROM if enabled
- Program binaries
 - ROM (in-place execution; how about e.g. code encryption for protection or upgrades?)
 - RAM (additional RAM required for programs)
- Data
 - RAM
 - Saving to disk/flash memory? What would the user be expecting?

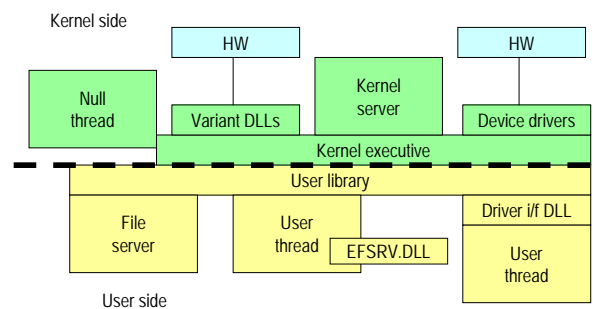
Content and goals

- Leaking abstractions
- Structure of a mobile device
 - Hardware
 - *Basic software concepts and run-time infrastructure*
 - Software stack and installation
- Summary

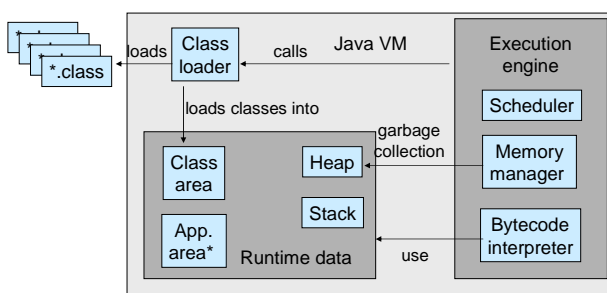
Basic Software Concepts

- Hardware abstraction layer, HAL
- Kernel
 - Scheduling (pre-emptive, non-pre-emptive)
 - Interrupt handling
- Device drivers for providing access to hardware
 - Physical vs. logical drivers
- Processes for resource reservation
 - Files, semaphores, etc.
- Threads as units of execution
 - Individual flows of control
- Virtual machine when needed

Example: Symbian OS



Example: Mobile Java Virtual Machine

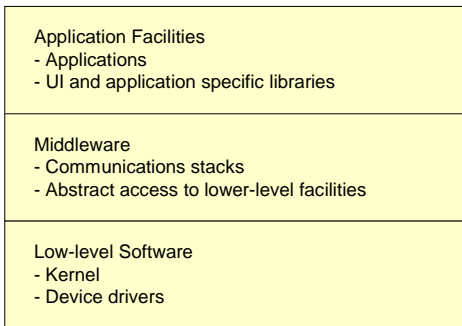


* App. area includes e.g. Program Counter

Content and goals

- Leaking abstractions
- Structure of a mobile device
 - Hardware
 - Basic software concepts and run-time infrastructure
 - *Software stack and installation*
- Summary

Software Stack

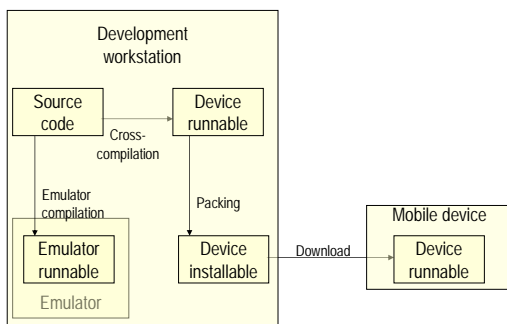


Common device-specific concerns

- Device-inflicted variance of software
 - Different screen sizes and types
 - Provided performance (accelerators etc)
 - Amount of memory
 - Etc.
- Stability of interfaces between different releases (and individual devices)
- Binary compatibility between different releases
- Compatibility between different generations of systems
- Supported standards



Development Process (binary)

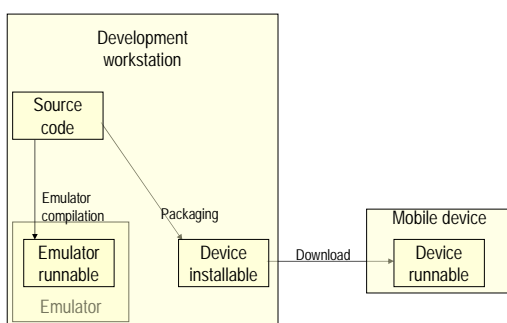


Different approaches to cross-compilation

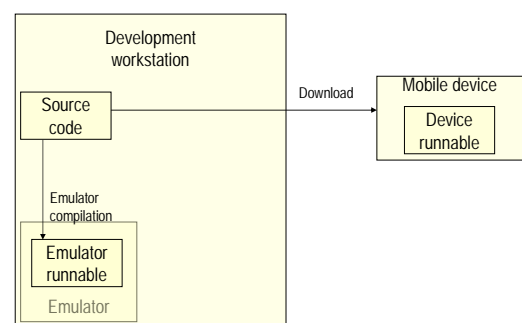
- **MIDP Java**
 - Direct generation of runnable bytecodes
 - Same in laptop and device
- **Symbian**
 - Emulator to test on PC
 - Different compiler for actual device
- **Maemo**
 - Scratchbox cross-compilation installed for development
 - Possible to define what output is generated



Development Process (scripting)



Development Process (web)



Content and goals

- Leaking abstractions
- Structure of a mobile device
 - Hardware
 - Basic software concepts and run-time infrastructure
 - Software stack
- *Summary*



Summary

- Mobile devices' programming hardened due to leaking abstractions
- Basic architecture
 - Processor, memory hierarchy, auxiliaries
 - Operating system, middleware, applications
 - Infrastructure reuse preferable
- Programmer may be responsible for allocating variables to memory and data to disk (in particular binary apps)
- Overhead in run-time infrastructure – performance may vary!
 - Virtual function tables
 - Virtual machines

