Handheld OS

Jagadish Rath (2003CS10164)
Mayukh Datta Roy (2003CS10173)
Handheld devices:- pocket sized computers

--Information Appliance

--process information, signals, graphics, animation, video and audio; and exchange information with another IA device

--Smart phones

--mobile phone +(PDA or IA)

--Personal Digital Assistants (PDA)

-- Mobile phones

-- handheld game console
We try to use as few resources as possible. Less resources means

- less cost,
- less weight and size,
- less heat generation,
- more battery life, more reliability.
Resources: —

--memory

--power

-- less number of components

constraint on resource minimization:-

--Functionality

-- trade off b/w functionality and resource
Physical resources

-- fixed at design time

-- size of memory

-- one LCD and one key pad

-- no of processors

-- can’t be reduced after the product is manufactured

-- Organized and smart operation on the resources at runtime

-- can reduce Usage of power

-- sometimes can allow us to reduce the size of Physical resources at design time.

-- OS is used to control & organize the operations in a computer
Operating Systems used in Handhelds

--Symbian (EPOC)

--Palm OS

--Windows Mobile (Windows CE)
  --BREW

--Linux

--RIM for the BlackBerry

--GPE—based on GTK+/X11

--OPIE/Qtopia

--We will discuss about linux on embedded devices
Architecture

-- a variation of ARM architecture is generally used.
From Desktop Linux to Embedded Linux

for Embedded OS

--Need not include every thing
  --Single,Fixed Purpose
--flexibility is not an issue
--End-User oriented

What we need to do?

--Reduce the size and provide the same functionality
  --Reducing redundancy
--compressing the code
Reduction of Size

--File Elimination

--Library Reduction

--Alternate Implementation

--Altering the source code

--File Elimination

--elimination of

-- Documentation

--Auxiliary Programs

---Helper programs, update conversion programs etc.
Configuration Samples can also be removed
  --e.g extra conf files /etc/httpd/apache/conf/http.conf-dist

Library Reduction—
  --Avoid unused Symbols
    --use static linking
    --eliminate unused symbols from shared libraries
  --use alternative libray
    --smaller versions

Alternate Implementation
  --Replacing Desktop programs with embedded versions
  --Replacing program suites with multi-function programs
Desktop Linux from boot to shell

Each one in the circles is done by a different program.

Boot

init

Xwindows  login  getty  passwd

shell

ls  mv  tar  gzip
Embedded Linux from boot to shell

All the programs in one box is done by a multi-function program

Boot
init

Xwindows login getty passwd

Shell
ls mv tar gzip

Smaller boot loader like uboot
e.g. - Tinylogin is a multi-function program which does these
e.g. - Busy Box is the multi-function program which does all these.
Busybox

- Busybox combines tiny versions of common Unix utilities into single small utilities
- Designed for memory strappled systems, using minimum run time memory
- Highly modular to allow inclusion and exclusion of features at compile time
Busybox

• The busybox executable can act like many different programs depending on the name used to invoke it.

• Normal practice is to create a bunch of symlinks pointing to the busybox binary, each of which triggers a different busybox function.
Busybox

- Destop linux is implemented on the glibc library
- Busybox uses an embedded version of this library which is called uClibc
UClibc

- Drawbacks of glibc
  - Lacks support for MMU less systems
  - Constrained by its large size
  - Not designed for handhelds
UClibc

- Size is a major consideration
- Targeted at embedded systems
- Optimized for Linux
- Size reductions achieved by
  - Disabling features
  - Reducing performance
  - Elimination of redundancy from the code
Busybox

• Single multi-purpose executable is smaller than many small files could be. This way busybox only has one set of ELF headers, it can easily share code between different applications even when statically linked, it has better packing efficiency by avoiding gaps between files or compression dictionary resets, and so on.
Busybox

- Busybox execution starts with the main() function in applets/busybox.c, which sets the global variable bb_applet_name to argv[0] and calls run_applet_by_name() in applets/applets.c. That uses the applets[] array (defined in include/busybox.h and filled out in include/applets.h) to transfer control to the appropriate APPLET_main() function (such as cat_main() or sed_main()). The individual applet takes it from there.
Busybox

- This is why calling busybox under a different name triggers different functionality: `main()` looks up `argv[0]` in `applets[]` to get a function pointer to `APPLET_main()`. 
Fork and Vfork

• On systems that haven't got a Memory Management Unit, fork() is unreasonably expensive to implement so a less capable function called vfork() is used instead.

• Implementing fork() depends on having a Memory Management Unit. With an MMU then you can simply set up a second set of page tables and share the physical memory via copy-on-write. So a fork() followed quickly by exec() only copies a few pages of the parent's memory, just the ones it changes before freeing them.
Fork and Vfork

- With a very primitive MMU (using a base pointer plus length instead of page tables, which can provide virtual addresses and protect processes from each other, but no copy on write) we can still implement fork. But it's unreasonably expensive, because you have to copy all the parent process' memory into the new process (which could easily be several megabytes per fork). And we have to do this even though that memory gets freed again as soon as the exec happens. (This is not just slow and a waste of space but causes memory usage spikes that can easily cause the system to run out of memory.)
Fork and Vfork

- Without even a primitive MMU, you have no virtual addresses. Every process can reach out and touch any other process' memory, because all pointers are to physical addresses with no protection. Even if you copy a process' memory to new physical addresses, all of its pointers point to the old objects in the old process. (Searching through the new copy's memory for pointers and redirect them to the new locations is difficult.)
Vfork

- So with a primitive or missing MMU, fork() is just not a good idea.
- In theory, vfork() is just a fork() that writeably shares the heap and stack rather than copying it. In practice, vfork() has to suspend the parent process until the child does exec, at which point the parent wakes up and resumes by returning from the call to vfork().
Implementing Vfork

- Parent does a setjmp and then continues on (pretending to be the child) until the exec() comes around, then the _exec_ does the actual fork, and the parent does a longjmp back to the original vfork call and continues on from there.
Power Management (PM)

- Discussion before this concerned the memory limitation of handhelds
- We all need to manage the usage of power.
PM through H/W

• using Flash memory as a substitute for hard drive

• Using components which do not spin rather go to sleep while they do not have any thing to do.
Extra advantages of Flash Memory

• Faster, produces less heat, has no moving parts.
• Processor can directly address the bits stored in Flash memory like a RAM or ROM making execution in place (XIP) possible, thus reducing the overall RAM requirements.
Disadvantage

- expensive than an equal amount of hard drive space
- it has much less capacity
- as you write to Flash memories, they slowly wear out.
- These advantages are not very important
Power Management

- Traditionally APM (Advance Power Management)
- Recently ACPI (Advance Configuration and Power Interface)
- These cannot co-exist hence only one can be selected.
Advanced Power Management (APM) consists of one or more layers of software that support power management in computers with power manageable hardware. APM defines the hardware independent software interface between hardware-specific power management software and an operating system power management policy driver. It masks the details of the hardware, allowing higher-level software to use APM without any knowledge of the hardware interface.
Overview

• Components
  – APM Bios: *The system BIOS module that provides power management functionality for motherboard resident hardware adhering to the APM Interface specification.*
  – APM Interface: *is the interaction between the APM drivers and the APM bios.*
Components

• APM driver – The software module connected to the APM bios interface. The APM Driver and the APM BIOS coordinate APM system power management through the software connection.

• APM aware application - An application that participates in system power management through an operating system dependent APM Driver interface.
Components

- APM aware device drivers: *An APM-aware application that provides power management for add-in devices.*

- APM connection: *The protocol and mechanism by which an APM Driver and an APM BIOS intercommunicate.*

- Add-in device: *A device which is not on the Motherboard. This device may be a permanent part of the system like an add-in card, or it may be insertable and removable by the operator, like a PCMCIA card. APM-aware device drivers provide power management software support for add-in devices.*
Definitions

• CPU Core - The hardware of the system that must be powered on to perform any computation. The CPU core includes the CPU clock, cache, system bus, and system timers.

• Resume Time - This is a system timer used to wake up the system from a low power state at a specified time.

• System - The entire computer. It includes the CPU core, I/O devices and peripherals, the power supply, and all devices and components.

• System Idle - The system is inactive. The CPU is not processing and there is no I/O activity.
APM model

- Basic Idea
- System Power States
  - Full on
  - APM enabled
  - APM standby
  - APM suspend
  - Off
Device Control

- Modus-operandi
- Device modes

  Device on
  Device Power managed
  Device Low powered
  Device Off
CPU core control

- Basic Operation
- CPU core modes
  - Full on
  - Slow Clock
  - Stop
System Power State Transitions

- Off switch / Off call
- On switch
- APM enable /enable call
- APM disable/disable call
- Short Inactivity /Standby call
- Long Inactivity/Suspend Interrupt/Suspend call
- Resume Event
- Wakeup Event
ACPI

- Introduction (What is ACPI?)
- APM vs ACPI!
CPU management under ACPI

- CPU idle power states
- CPU frequency management
- CPU throttling