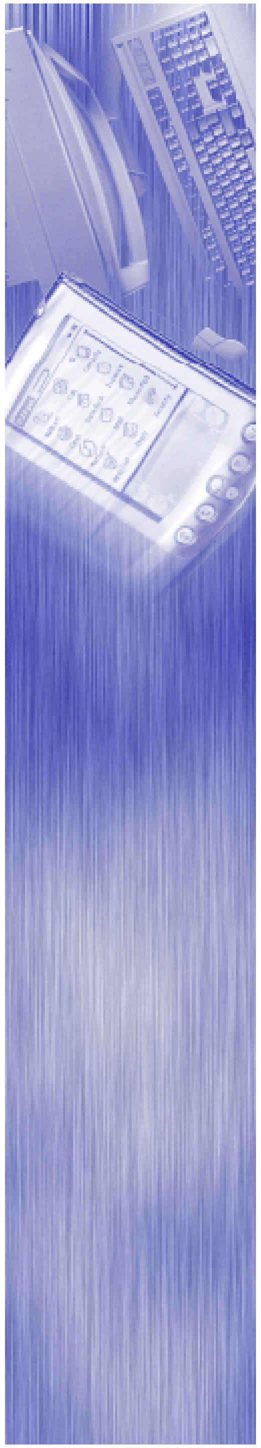


CS632/SEP564: Embedded Operating Systems (Fall 2008)

Embedded Software Optimization

Making Things Smaller

- Kernel
- Root filesystem
- Application

Making Things Smaller (1)

- **Downsizing the kernel**
  - Include only the necessary modules by a correct configuration
    - Device drivers, network stacks, file systems, etc.
  - Remove unnecessary kernel functionalities
    - Turn off debugging and error logging flags
    - Remove support for sysctl (used to tweak kernel parameters at run time)
    - Get ~18KB by removing msg* (SysV) and mq_* (POSIX)
  - Use the optimization flags when compiling the kernel
    - –O1, –O2, –O3, and –Os
    - Generally select –O2 to achieve a balance between size and performance

Making Things Smaller (2)

- **Downsizing the kernel (cont'd)**
  - Decrease the size of the static buffer and array allocated in the kernel
    - – Modify options in the menuconfig of the kernel
      - » The maximum number of supported peripherals, etc.
    - – Use nm to identify the allocated size of each variable in an object file.
  - Use strip to remove unnecessary symbols in the kernel and kernel modules
  - Disable KALLSYMS which is used to print out symbolic crash information and symbolic stack backtraces

Making Things Smaller (3)

- **Linux-tiny patches**
  - http://elinux.org/Linux_Tiny
  - A series of patches to reduce the memory and disk footprint of the kernel
  - Originally started by Matt Mackall in Dec. 2003
  - Stalled in late 2005 (Linux 2.6.14)
  - Revived by Tim Bird (Sony) & CELF (Consumer Electronics Linux Forum) in 2007
  - The latest release is for Linux 2.6.23
  - Goals
    - ~300KB of a compressed kernel for simple configurations running comfortably with 2MB of RAM

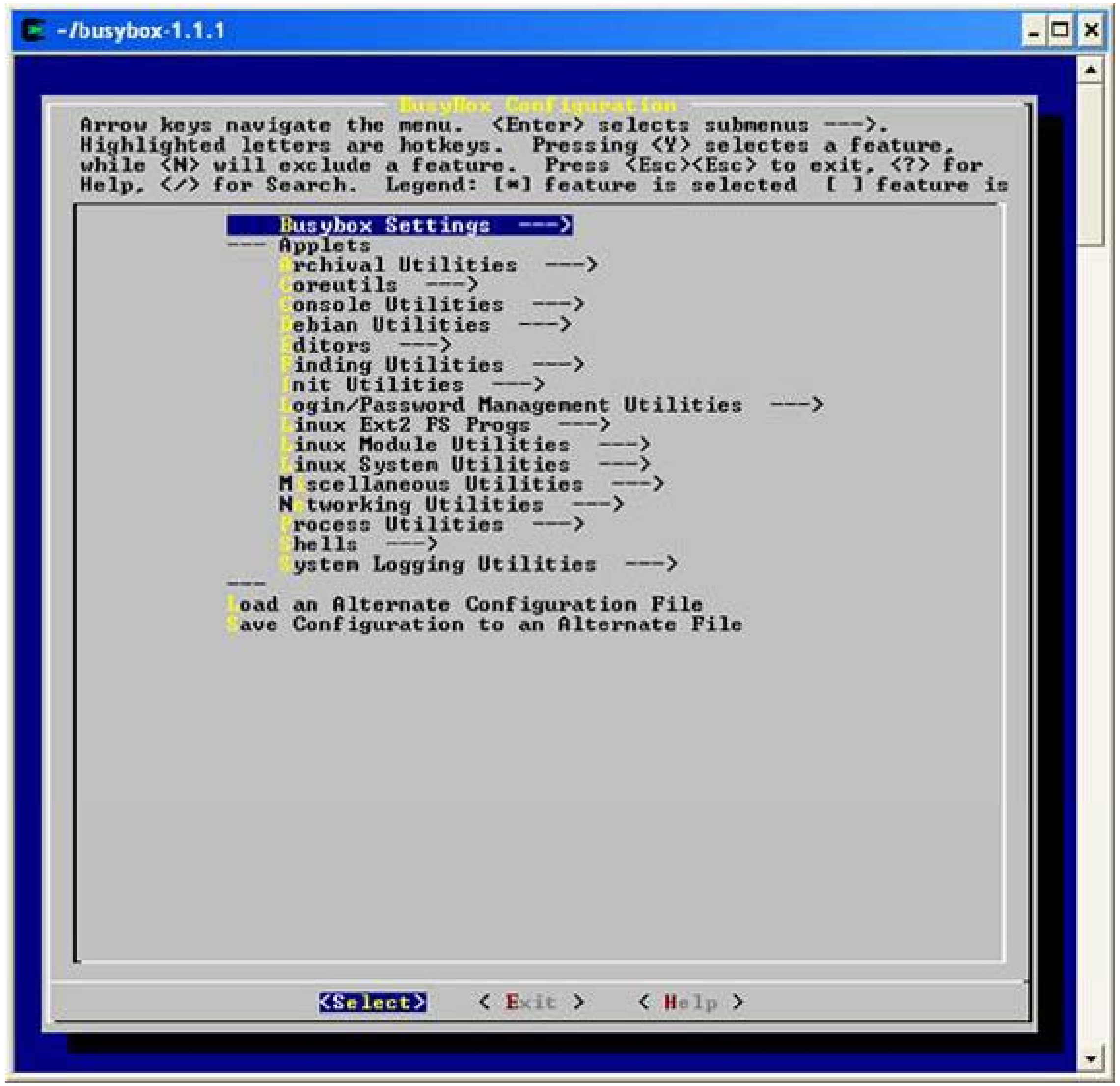
Making Things Smaller (4)

- **Linux-tiny patches (cont'd)**
  - Fine-grained control of printk() message compilation
    - ~ 60,000 printk() calls in Linux 2.6
    - All the static strings contribute to 5 ~ 10% of the kernel size
    - Control over the files that can use printk()
  - Allow using function instead of table for CRC32 calculation
    - During packet processing in Ethernet driver
    - 2KB saving
  - Network tweaking: buffer sizes, open sockets, …
  - Reduction of inlining
  - Make some data structure sizes configurable, …

Making Things Smaller (5)

- **Downsizing the root filesystem**
  - Use busybox (http://busybox.net)
    - BusyBox combines tiny versions of many common UNIX utilities into a single small executable file
    - Highly modular, allowing commands to be included and excluded at compile time
    - Each utility can be accessed by calling the single BusyBox binary with various names
      - Supported by having a hard link or symbolic link
    - Reduces the overheads introduced by the executable file format (typically ELF)
    - Reduces internal fragmentation in memory and disk
    - Allows code to be shared between multiple applications without requiring a library

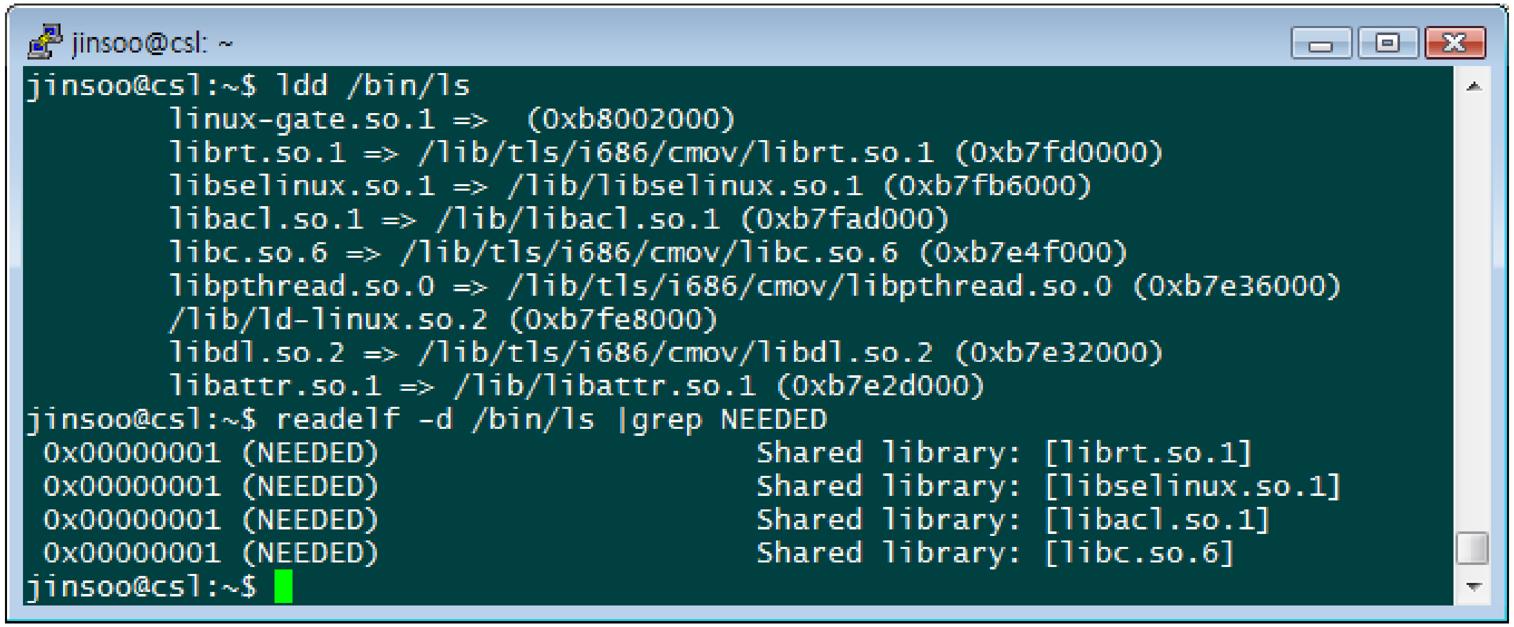
Making Things Smaller (6)



```
~/busybox-1.1.1

                    BusyBox Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> selectes a feature,
  while <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for
  Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is

            Busybox Settings  --->
        --- Applets
            Archival Utilities  --->
            Coreutils  --->
            Console Utilities  --->
            Debian Utilities  --->
            Editors  --->
            Finding Utilities  --->
            Init Utilities  --->
            Login/Password Management Utilities  --->
            Linux Ext2 FS Progs  --->
            Linux Module Utilities  --->
            Linux System Utilities  --->
            Miscellaneous Utilities  --->
            Networking Utilities  --->
            Process Utilities  --->
            Shells  --->
            System Logging Utilities  --->
        ---
        Load an Alternate Configuration File
        Save Configuration to an Alternate File

              <Select>    < Exit >    < Help >
```

Making Things Smaller (7)

- **Downsizing the root filesystem (cont'd)**
  - Remove unnecessary libraries
    - Use ldd to identify the required shared libraries for each program
    - (cf.) readelf -d



```
jinsoo@csl:~$ ldd /bin/ls
        linux-gate.so.1 =>  (0xb8002000)
        librt.so.1 => /lib/tls/i686/cmov/librt.so.1 (0xb7fd0000)
        libselinux.so.1 => /lib/libselinux.so.1 (0xb7fb6000)
        libacl.so.1 => /lib/libacl.so.1 (0xb7fad000)
        libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0xb7e4f000)
        libpthread.so.0 => /lib/tls/i686/cmov/libpthread.so.0 (0xb7e36000)
        /lib/ld-linux.so.2 (0xb7fe8000)
        libdl.so.2 => /lib/tls/i686/cmov/libdl.so.2 (0xb7e32000)
        libattr.so.1 => /lib/libattr.so.1 (0xb7e2d000)
jinsoo@csl:~$ readelf -d /bin/ls |grep NEEDED
 0x00000001 (NEEDED)                     Shared library: [librt.so.1]
 0x00000001 (NEEDED)                     Shared library: [libselinux.so.1]
 0x00000001 (NEEDED)                     Shared library: [libacl.so.1]
 0x00000001 (NEEDED)                     Shared library: [libc.so.6]
jinsoo@csl:~$
```

Making Things Smaller (8)

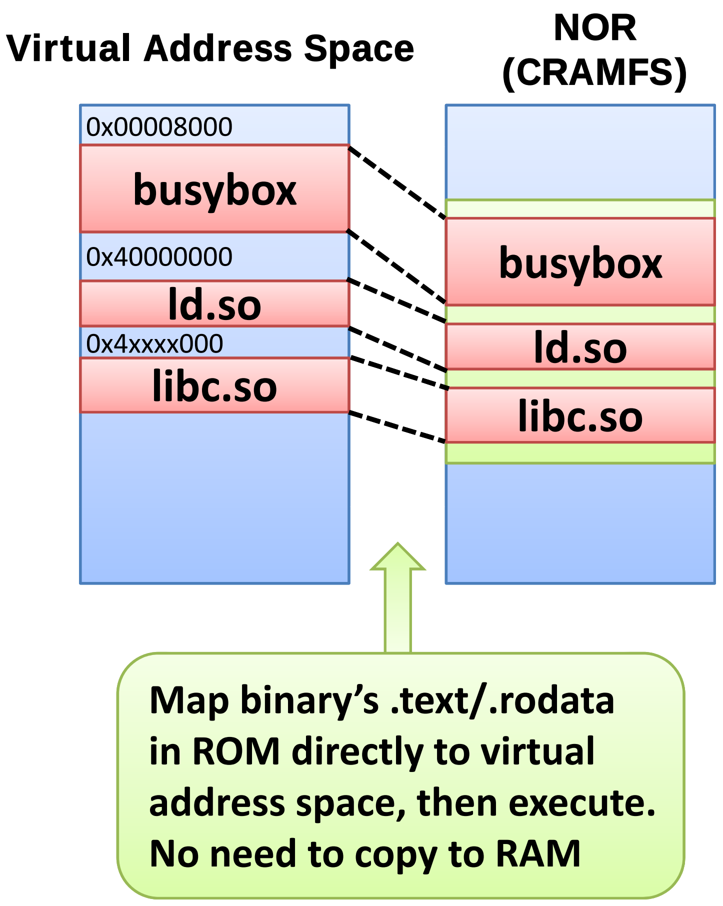
- **Downsizing the root filesystem (cont'd)**
  - Replace the standard C library with a small C library
    - uClibc, uClibc++, Newlib, or dietlibc
    - Such libraries remove the unused functions
  - Use a library optimizer Libopt to rebuild the libraries
    - Include only the necessary functions for the executable programs and shared libraries found in the root filesystem
    - Libopt utilizes objdump and nm to gather information about library object files, shared libraries and executable programs
  - Remove unnecessary commands and packages
  - Remove unnecessary directories and documents
    - /home, /mnt, /opt, /root, /boot, and /proc
    - Man, info, include, example directories

Making Things Smaller (9)

- **Downsizing the root filesystem (cont'd)**
  - Use compressed file system such as CRAMFS or SquashFS
    - Supports random read access with on-the-fly decompression
  - Eliminate unused code/data
    - CFLAGS +="–ffunction-sections –fdata-sections"
    - LDFLAGS += "–gc-sections"
  - Do not compile with –g
  - Use strip to remove all symbols
  - Compile with 16-bit instruction set (Thumb in ARM)

Making Things Smaller (10)

- **Reducing memory usage**
  - Application XIP (eXecute-In-Place)
    - Map code pages on NOR CRAMFS to Virtual Address Space directly.
    - Space vs. execution speed
    - Total ~45% reduction of page cache allocated to processes at stand by after bootup
    - The needed NOR flash size increases x 2 (Compression is omitted)



Map binary's .text/.rodata in ROM directly to virtual address space, then execute. No need to copy to RAM

Making Things Smaller (11)

- **Reducing memory usage (cont'd)**
  - "Allocate on Write" for .data sections
    - – Some pages in .data sections remain unwritten throughout process lifetime
    - – The kernel allocates page cache to a process whenever it accesses pages in .data sections.
    - – The page allocation can be deferred for .data until the process writes to the page
    - – XIP + COW mechanisms
    - – Total ~26% reduction of page cache allocated to processes at stand by idle state.
    - – NOR flash access latency slows down process execution

Making Things Faster (1)

- **General bootup procedure**
  - Boot loader
    - System initialization
    - Kernel image loading from NOR/NAND/OneNAND Flash
  - Kernel initialization
    - Kernel initialization
    - Device drivers initialization
    - Mount root filesystem
    - /sbin/init invocation
  - Startup scripts (/etc/inittab, rc.*)
    - Load dynamic kernel modules
    - Mount filesystems
    - Run applications

Making Things Faster (2)

- **Boot loader**
  - Reduce the kernel image size
  - Do not enable unnecessary devices (e.g., NIC)
  - Compressed vs. uncompressed kernel image?
    - Depends on the flash speed, memory bandwidth, CPU performance, etc.
  - Flash speed matters!
    - OneNAND synchronous burst mode?
  - Kernel XIP
    - Allow kernel to be executed in-place in ROM or NOR Flash
  - Use DMA for copying the kernel image
    - Save the time and CPU resources

Making Things Faster (3)

- **Kernel**
  - Preset loops_per_jiffy (LPJ)
    - – Avoid a delay loop calculation (calibrate_delay())
  - Avoid the RTC (Real-Time Clock) edge synchronization
  - Disable console output
  - Lazy device driver initialization
    - – If a device driver is not necessary on system init, separate it as a kernel module
  - Probe and initialize several devices in parallel
  - Disable page-cache readahead option
  - Bypass flash sector-mapping layer for read-only filesystems

Making Things Faster (4)

- **Startup scripts**
  - Insert kernel modules into the kernel
    - Reduce the time for kernel symbol resolution
    - Reduce the time for copying the entire image to memory
  - Lazy loading for not-so-urgent kernel modules
    - e.g., USB (usbcore, usb-hub, usb-storage), networking, …
  - Remove unused services on init/rc.* script
  - Run rc.* scripts in parallel
  - Use static linking to avoid run-time overhead
    - Busybox, utilities, applications
  - Directly call bootup commands inside busybox
  - Compressed or uncompressed filesystem?

Making Things Faster (5)

- **Application**
  - Focus on the urgent things first
    - e.g., Screen & OSD display on DTV
    - Global constructors?
    - Lazy process/thread creation
  - Preallocate memory used during bootup sequence
    - Avoid calling malloc()
  - Use mmap() instead of read() for initial application data
    - Only the required pages are loaded on demand
    - More page faults, but less memory copy
    - e.g., Fonts, images, etc.

Making Things Faster (6)

- **Suspend / Resume**
  - Resume the snapshot image at boot time
  - swsusp, Suspend2 (TuxOnIce), uswsusp,
    - Only support IA32, PPC, x86_64, IA64
  - No official support for ARM and MIPS
    - (cf.) TP InstantBoot (http://www.tripeaks.co.jp)
  - A snapshot image is created only once, and used repeatedly
  - The snapshot image can be compressed
  - The page cache can be populated in advance
  - Resuming device states is tricky
  - Who is resuming, kernel or boot loader?

Making Things Faster (7)

- **Dynamic loading of shared libraries**
  - A new application is loaded via do_execve()
  - The type of executable is determined by parsing the ELF header
  - For dynamically-linked executable, transfer the control to the dynamic linker (ld.so)
  - ld.so loads all shared libraries needed by the application
  - Relocate external references to shared libraries (Lazy binding of function calls)
    - Indirectly via GOT (Global Offset Table) and PLT (Procedure Linkage Table)
  - Transfer the control to libc runtime

Making Things Faster (8)

- **Prelink**
  - Symbol lookup and relocation handling increases the application startup time significantly
  - Originally developed for GUI programs with lots of shared libraries (OpenOffice, etc.)
  - Prelinking steps
    - Identify executables to be prelinked and their dependent shared libraries
    - Assign non-overlapping virtual address space to each shared library
    - Apply the relocation
    - The dynamic linker is modified to handle prelinked executables/shared libraries

Making Things Faster (9)

- **Profile-driven functions reordering**
  - Profiling phase
    - CFLAGS += “-finstrument-functions”
    - cyg_profile_func_enter(): output the address of the called fn
  - Analysis phase
    - Identify all the functions and libraries, if any, they belong to.
    - Generate a linker script
  - Reordering phase
    - Relink the application with the new linker script
- **Profile-driven data reordering**
  - Data sections: .rodata, .data, .bss
  - How can we identify data touched during startup?

Making Things Faster (10)

- **Profile-driven function/data reordering**
  - Reduces the number of pages faults
  - Reduces the number of pages read from flash
  - Better locality in TLB and cache
  - Reliable as is done by the standard linker

  - Requires the instrumented version of objects/libraries
  - May require full source code and recompilation
  - Standard libraries? System call stubs?
    - libc, libm, libpthread, libstdc++, libgcc, libgcc_eh, …
  - Data access profiling is not precise

Thank You!

- **Final exam.**
  - December 16, 2:30pm – 4:00pm
  - CS Building #4448
  - Scope: everything
  - Closed-book exam.

  - Count as 30% of your final grade!
  - Submit your final project report on time. (another 70%)