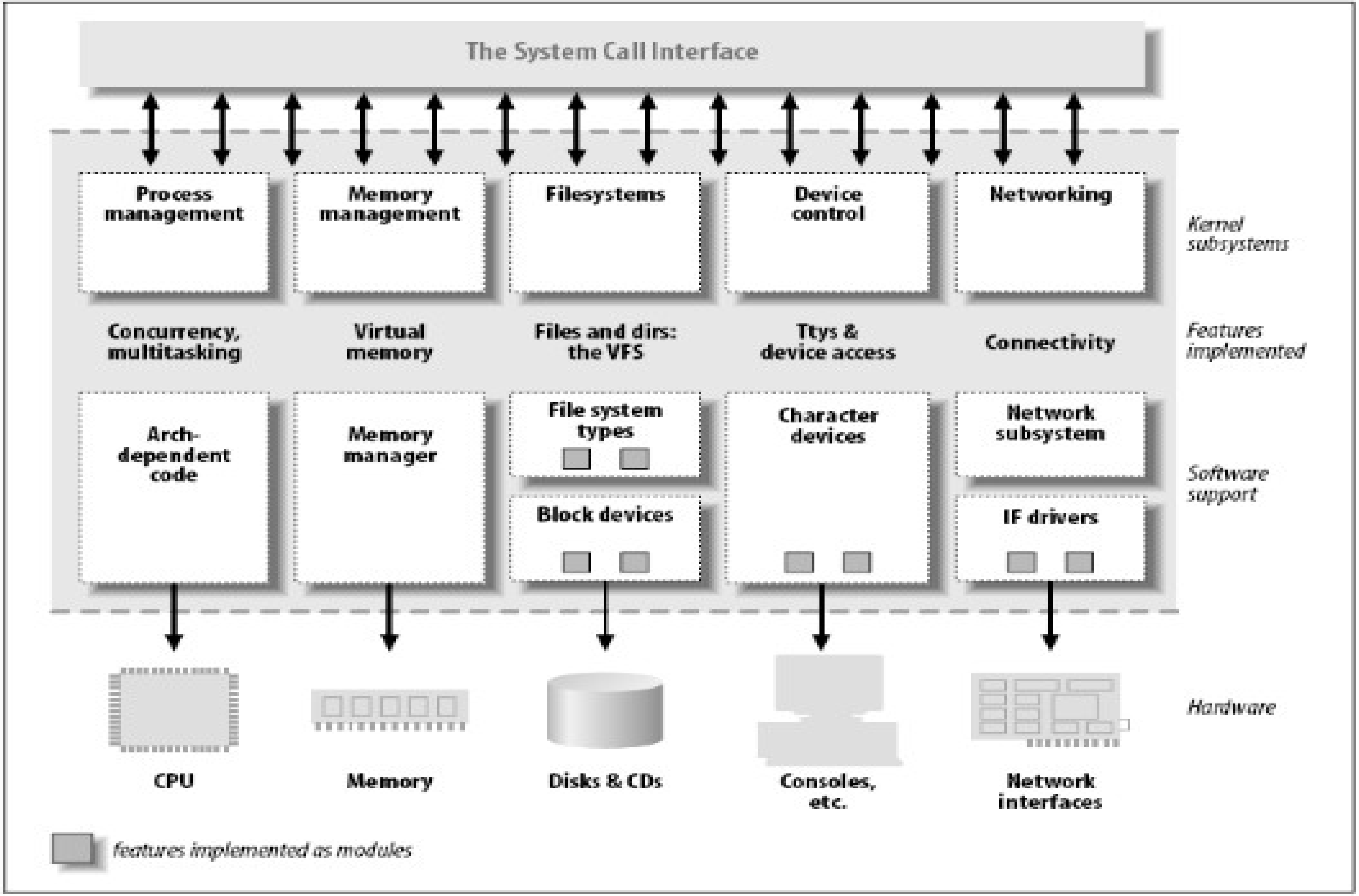
KERNEL COMPILATION

AGENDA

- What is a kernel?
- Why to custmize the kernel?
- Steps involved in the kernel compilation?
- Importance of initramfs file system?

Kernel


The System Call Interface
Process management
Memory management
Filesystems
Device control
Networking
Kernel subsystems
Concurrency, multitasking
Virtual memory
Files and dirs: the VFS
Ttys & device access
Connectivity
Features implemented
Arch-dependent code
Memory manager
File system types
Block devices
Character devices
Network subsystem
IF drivers
Software support
CPU
Memory
Disks & CDs
Consoles, etc.
Network interfaces
Hardware
features implemented as modules

Why to Customize?

- Update to latest kernel version
- To apply security patch
- To support new hardware
- Debugging the kernel

Tools to compile the kernel

- Inorder to build kernel we need three components
  - Compiler
  - Linker
  - Make utility

Steps to recompile the Kernel

Steps involved in compiling the kernel

- Retrieving the kernel source code
- Configuring the Kernel
- Building or compiling the kernel
- Installing the kernel

Retreieving the source code

- Download source code from www.kernel.org
- Difference between archieve and compression?
- How to create a tar file & how to decompress it?
- How to use gzip, bzip commands?

Configuring the kernel

- Configuring from scratch
  - make config
  - make menuconfig - (ncurses based)
  - make xconfig - (QT based)
  - make gconfig – (gtk+ based)
- Default configuration options
  - make defconfig
  - make oldconfig
  - make allmodconfig
  - make allyesconfig

Module or Built-in (Static)?

Building drivers into the kernel makes the kernel FAT – require more memory and overall slower execution.

YET certain drivers are better of being built-in (e.g: motherboard drivers)

Building drivers as modules results in a thinner kernel that can load external modules as an when needed.

BUT make sure your kernel has access to essential drivers required to boot, through an ***initrd*** or making them built-in.

Compiling/Building & Installation

- Compiling
    - make -(j * double the no. of processors)
- Installation
    - make modules_install
    - make install

make install

- Copy arch/i386/boot/bzimage to /boot direcory
- Copy *System.map* file to /boot
- Copy .config file to /boot
- Create initramfs file in /boot
- Editing the Grand unified boot loader (GRUB) / Linux Loader(LILO)

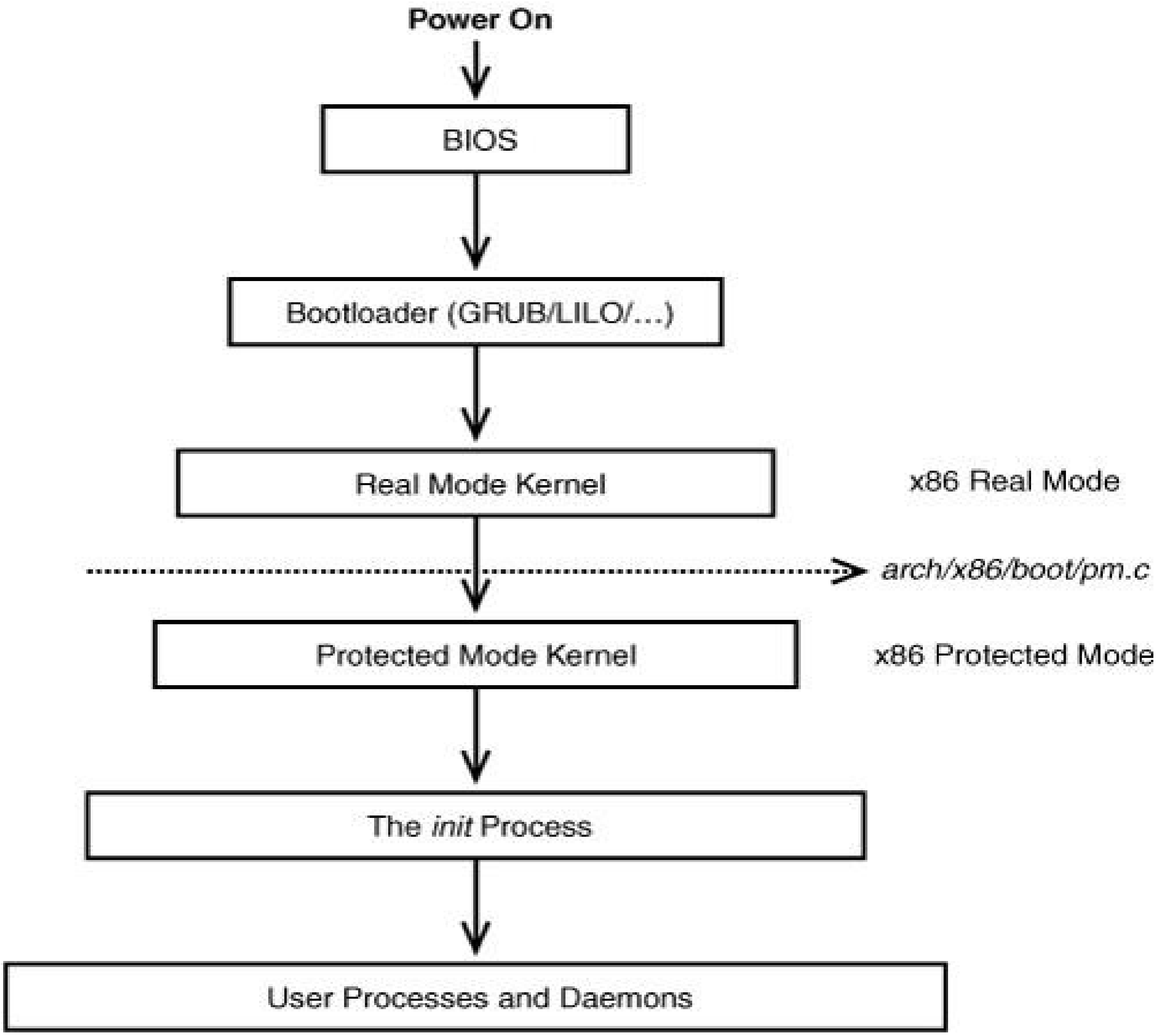
make clean

- Make clean – cleans all precompiled binaries
- Make mrproper – cleans all precompiled binaries

  and .config file
- Make distclean – cleans precompiled binaries,

  .config files & patch files

Linux boot sequence on x86-based hardware.



Power On

BIOS

Bootloader (GRUB/LILO/...)

Real Mode Kernel

x86 Real Mode

arch/x86/boot/pm.c

Protected Mode Kernel

x86 Protected Mode

The *init* Process

User Processes and Daemons

Linux Boot Sequence

- BIOS loads the Master Boot Record (MBR) from the boot device.
- Code resident in the MBR looks at the partition table and reads a Linux bootloader such as GRUB, LILO, or SYSLINUX from the active partition.
- The final stage of the bootloader loads the compressed kernel image and passes control to it.
- The kernel uncompresses itself and turns on the ignition.

Boot Loader

- The boot loader, such as GRUB, identifies the kernel using **BIOS** calls that is to be loaded and copies this kernel image and any associated initrd into memory.

- After the kernel and initrd images are decompressed and copied into memory, the kernel is invoked

- The kernel loads and begins doing some checks, but then stops because it can’t read the root filesystem which is in the hard disk.

Why initrd

- Unable to boot since boot driver for hard disk controller (i.e can't mount root file system) is not included as part of the kernel
- Can't include all the boot driver in the kernel
- Even hard-wiring tons of special case behavior into the kernel doesn't help with device enumeration, encryption keys, or network logins that vary from system to system

initramrd

initramrd(initial ramdisk) is a temporary root file system that is mounted prior to the real root file system during system boot to support the boot process

The initrd contains a minimal set of directories and executables to achieve this, such as the insmod tool & hold the relevant drivers as modules.

Boot Loader

- After the kernel and initrd images are decompressed and copied into memory, the kernel is invoked

- When the computer boots, the boot loader passes the initrd address to the kernel, which loads it into RAM and treats it like a disk. The kernel can then load modules stored in the RAM disk, keeping the kernel size small while still providing a wide variety of drivers.

Initramfs

- Initrd was replaced by initramfs. Initramfs works similar to initrd but most of the code is shifted to user space
- Currently, much of this work is done inside the kernel itself, leading to kernel code which duplicates user-space tools - but with less review and maintenance. Moving this work into a user-space boot-time filesystem promises to shrink the kernel, make the boot process more reliable, and allow distributors (and users) to customize the early bootstrap process in interesting ways.

Creating initramfs

- Mkinitramfs -o initrdname name in /lib/modules

- Mkdir temp
- Copy initrd image to temp directory
- File initrd   :- u will see gzip
- Use gunzip or gzip -d to uncompress it. If you are using gzip then you must rename initrd image with initrd.gz
- Now u will get cpio archieve
- cpio --help . U can see the command
- Cpio -i < initrdimage(in cpioformat).

  i for extraction

Steps to retreive initrd image

- mkdir temp
- cd temp
- cp /boot/initrd.img-2.6.28 initrd.img-2.6.28.gz
- gunzip initrd.img-2.6.28.gz
- cpio -i --make-directories < initrd.img-2.6.28

thank
you