Image Processing & Device Drivers

Hardware Software CoDesign

December 2011

Agenda

Image Processing

1. Introduction to Video Image Processing
2. Introduction to Device Drivers
3. Answering Machine Assignment (End Date 31st Dec 2011)

Video Image Processing

1. The JPEG system for compressing static images could be applied to a sequence of images, compressing each individually. This is called *motion JPEG*

2. Motion JPEG takes no advantage of any correlation between successive images.

3. In a typical scene there will be a great deal of similarity between nearby images of same sequence.

Video Image Processing

The Motion Compensation Approach

1. Many "moving" images or image sequences consist of a static background with one or more moving foreground objects.

2. Can we get a coding advantage from this feature ?

3. First code the frame by baseline JPEG and use this frame (snapshot) as a "reference image".

4. Divide the second image into blocks, and compare each block with the same block in the reference image.

5. For blocks that have identical block in reference image, we only send a special code instead of whole code.

6. For other "non identical" blocks, just encode them as usual.

Video Image Processing

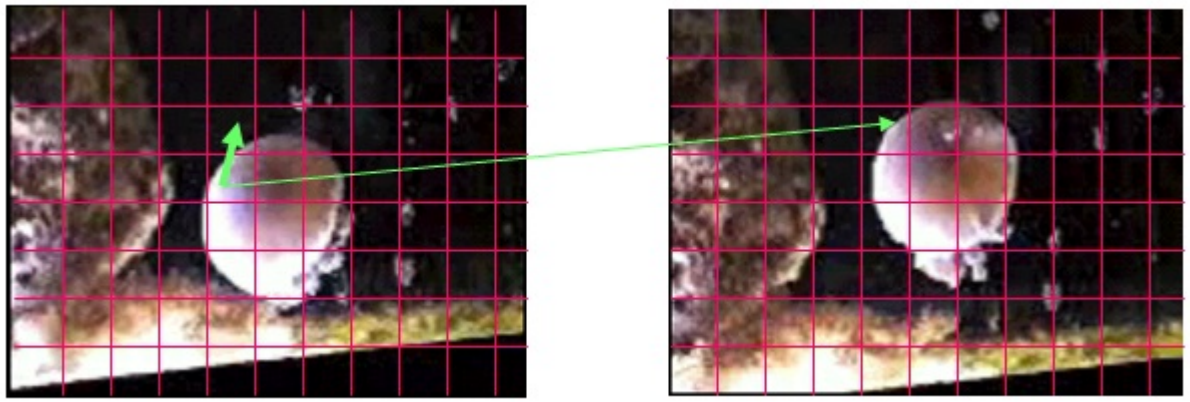
The Motion Compensation Approach → Motion Vectors

1. Static background is a very special case.

2. In general, if there is a block or moving object, there is a displacement of the block.

3. A Motion Vector is used to inform the decoder "exactly where in the previous image" to get the data.

4. Motion vector would be zero for a static background.



5. GROUP DISCUSSION ::
   (a) The object can have different shapes. How to recognize the shape.
   (b) The object can be a part of in different blocks.
   (c) How does one find the matching block.

Video Image Processing

The Motion Compensation Approach → Motion Vectors

1. GROUP DISCUSSION ::
   (a) The object can have different shapes. How to recognize the shape.
   (b) The object can be a part of in different blocks.
   (c) How does one find the matching block.
2. In practice we could'nt expect to find "exact" identical matching block.
3. Instead look for a close match.
4. Most Motion Estimation schemes look for a minimum mean square error (MMSE) between blocks.

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(I_n(x,y) - I_n^{'}(x,y))^2$$

5. What should be the matching block size :: How large the matching block will affect coding efficiency.
6. Block size MPEG used is 16 x 16.
7. GROUP DISCUSSION :: Typically the displacement on a per frame basis is low. So, what should be the reasonable search range for a match ?

Video Image Processing

The Motion Compensation Approach → Motion Vectors → Search Range

1. GROUP DISCUSSION :: Typically the displacement on a per frame basis is low. So, what should be the reasonable search range for a match ?

2. Itś reasonable to consider an displacement of 360 pixels or about 60pixels / image in standard definition television.

3. In real-world scenes there is usually more or faster motion horizontally than vertically, generally the width of such area should be twice the height.

4. Suggested search range:: -/+60 pixels (horizontal) x 30 pixels (vertical)

5. GROUP DISCUSSION :: We used the MMSE. What about Residuals (delta part).

Video Image Processing

The Motion Compensation Approach → Motion Vectors → Residuals

1. GROUP DISCUSSION :: We used the MMSE. What about Residuals (delta part).

2. The differences between the block being coded and itś best match are known as residuals.

3. The residuals maybe encoded and transmitted along with the motion vector, so the decoder will be able to reconstruct the block.

4. We should compare the bits of transmitting the motion vector plus the residuals with the bits of transmitting the block itself and use the most efficient mechanism.

5. Thought :: Can we use motion estimation, within of the same frame also.

Video Image Processing

MPEG Hierarchy The six layers of MPEG video bit stream are :-

1. Sequence Layer :: video clip, complete program item.
2. Group of Pictures Layer (GOP) :: includes three different coding ways.
3. Frame Layer
4. Slice Layer :: In case the data is lost or corrupted.
5. Macroblock Layer :: 16 x 16 Luminance block.
6. Block Layer :: DCT unit.

Video Image Processing

MPEG Hierarchy → Frame Types

1. Intra Frames (I-frames) :: An I-Frame is encoded using only information from within that frame (intra coded). There is no temporal compression (no coding wrt next or prev frames).

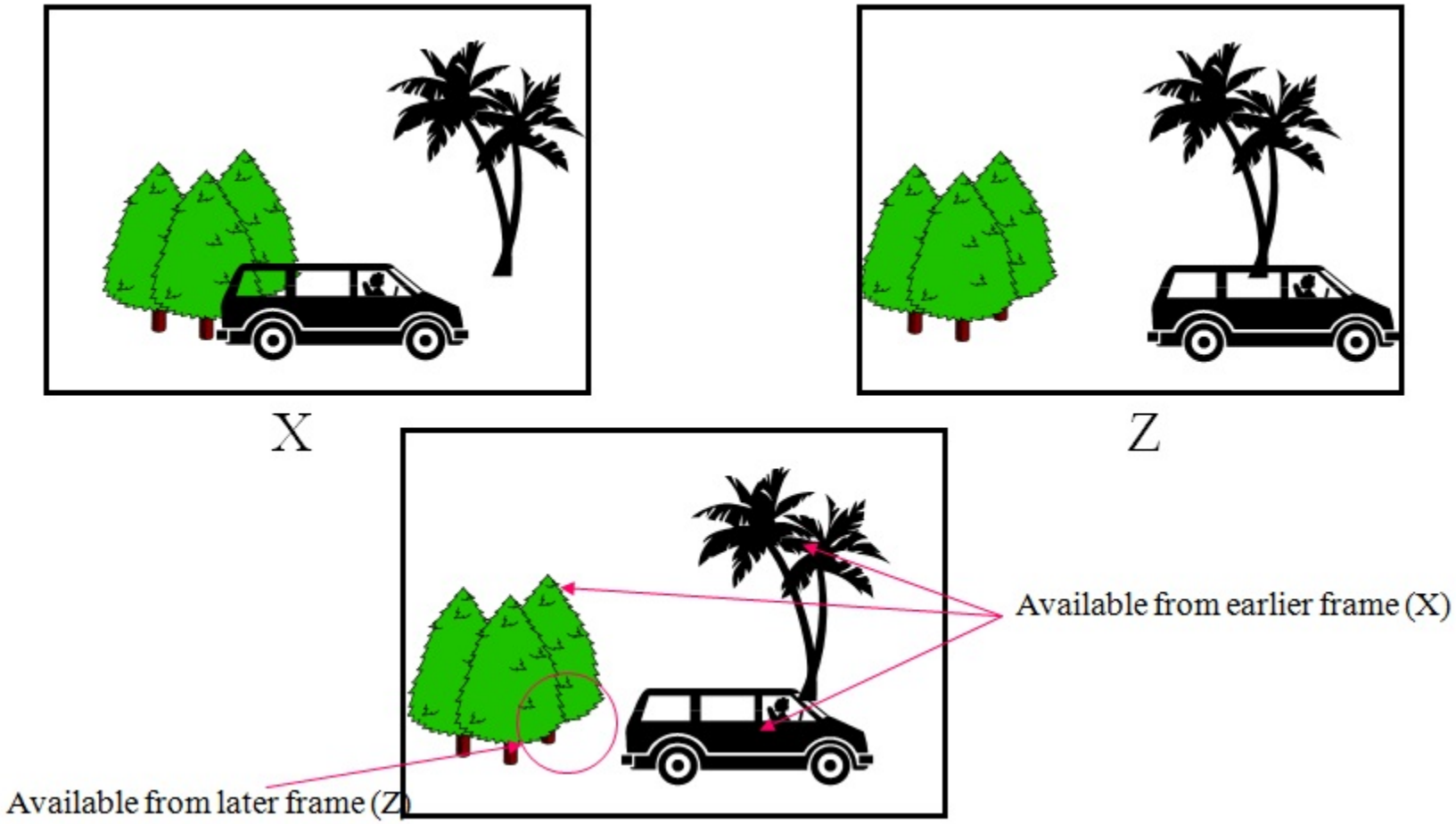
2. Non Intra Frames (P-Frames and B-Frames) :: Motion compensation is used for coding.

3. Predicted Frame (P-Frame) :: use preceding frame as reference image.

4. Bidirectional Frame (B-Frame) :: use both preceeding frame and following frame as reference images.

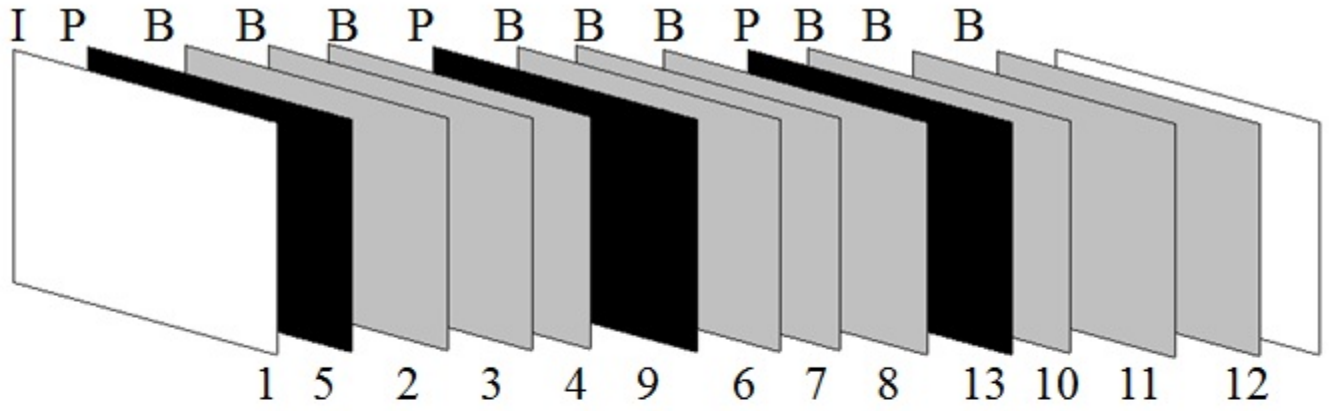
Video Image Processing

MPEG Hierarchy → Frame Types

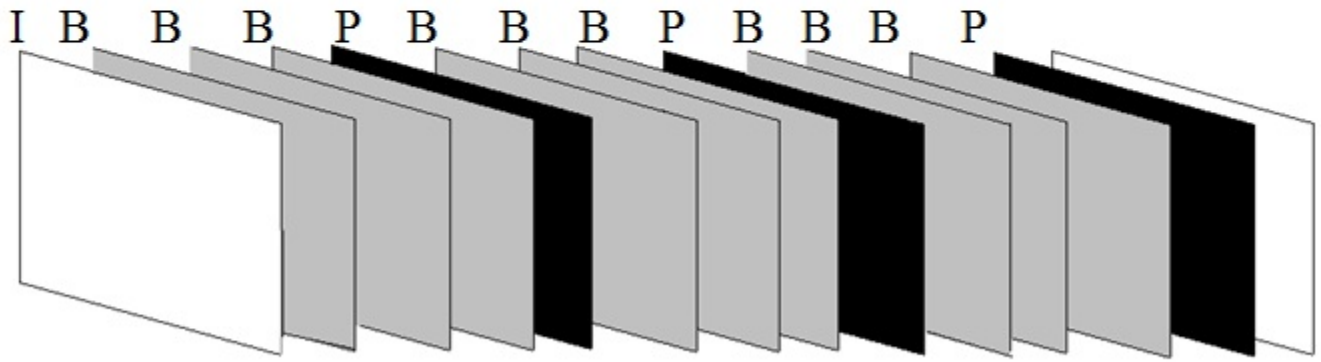


Video Image Processing

MPEG Hierarchy → Frame Types



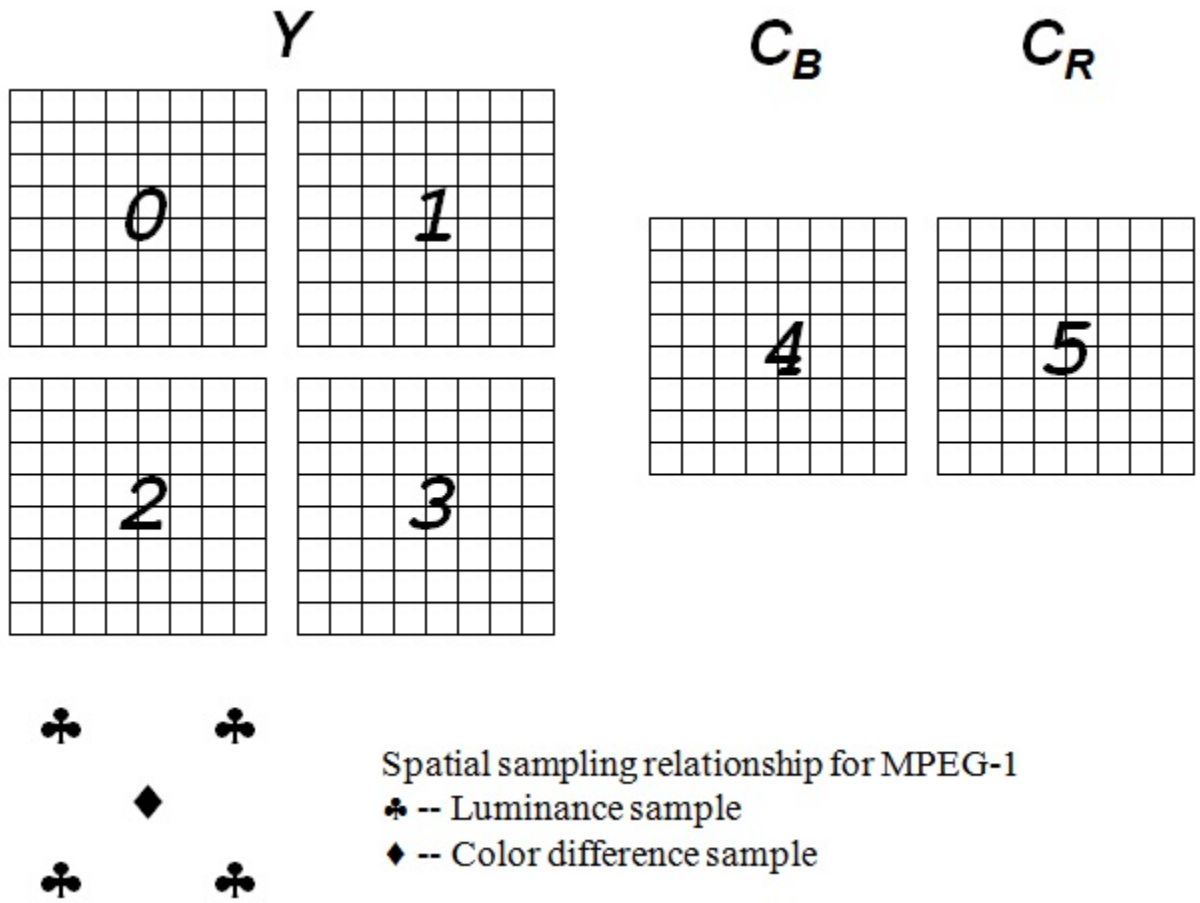
A typical group of pictures in coding order



A typical group of pictures in display order

Video Image Processing

Coding of Macroblock → 4:2:0



Spatial sampling relationship for MPEG-1
♣ -- Luminance sample
♦ -- Color difference sample

Video Image Processing

Coding of Macroblock → Intra Coding

1. Same as what JPEG does

2. MPEG has two default quantization tables, one for intra coding, another one for non-intra coding of residuals

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

MPEG quantization table(for intra coding)

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

JPEG quantization table(luminance)

Video Image Processing

Coding of Macroblock → Non-Intra Coding

1. First Step would be to intra code the macroblock → Just in case if there is a failure to find a reasonable match in motion estimation.

2. Then use Motion Estimation to find the nearest match and get the Motion Vector. Only Luminance samples are used in motion estimation.

3. Then each DCT block in macroblock will be treated separately. The residuals will be encoded by DCT and quantization same as in intra coding.

4. This process is applied to all blocks in the macroblock

5. Motion vectors are coded predictively

Video Image Processing

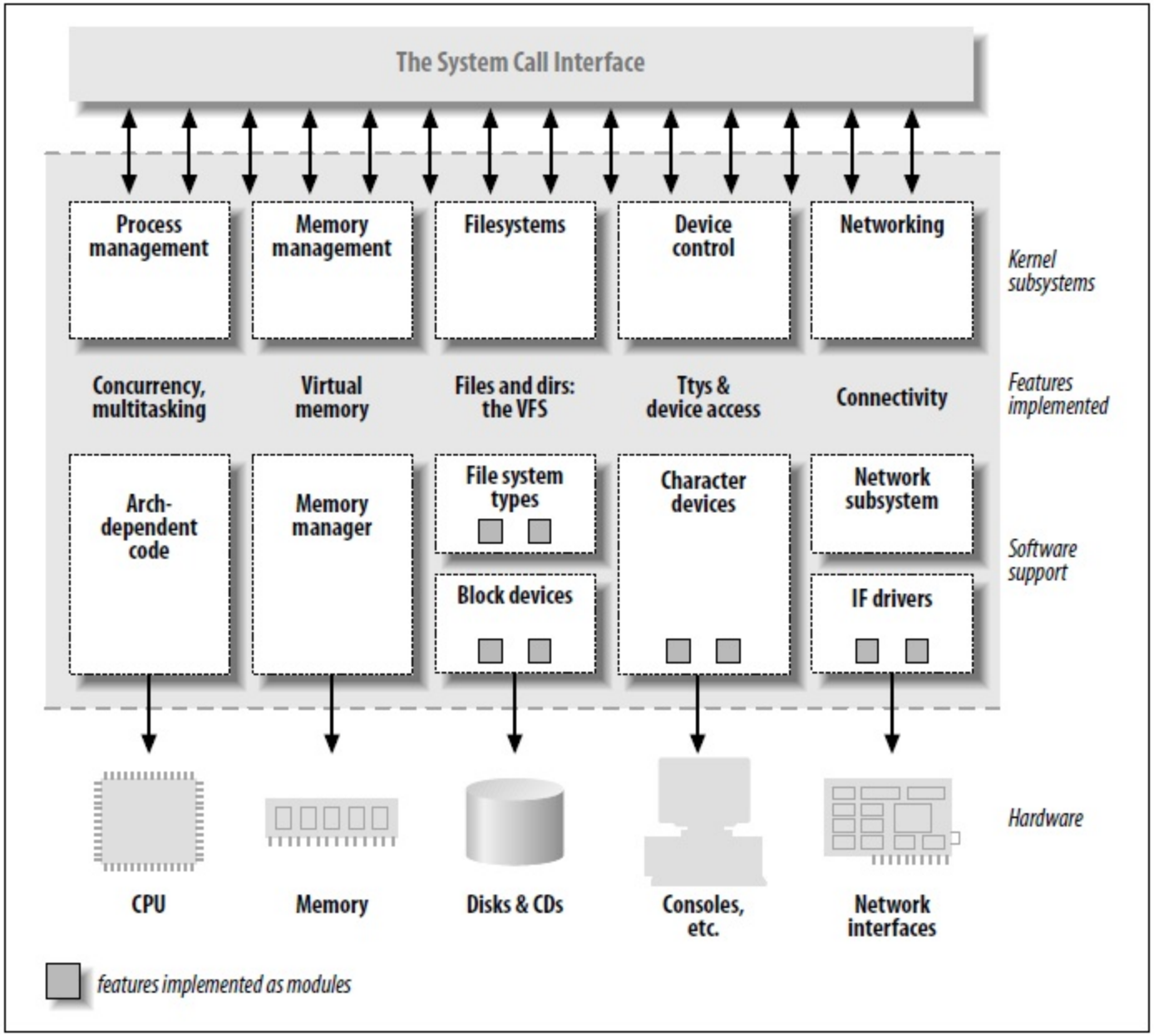
Coding of Macroblock → Non-Intra Coding

1. P-Frames
   (a) If the block can be skipped, just send a "skip" code
   (b) Otherwise, compare the number of total bits of inter and intra coding, choose the more efficient one.
   (c) Mark the block accordingly.
2. B-Frames
   (a) comparision among the methods of encoding

Logical Break

Device Drivers

Kernel view of Things



A split view of the kernel

Device Drivers

Kernel view of Things

1. The Kernel is the big chunk of executable code in charge of handling all requests → computing power, memory, network connectivity, any other resource allocation.

2. The distinction between the different kernel tasks isn't always clearly marked, the kernel's role can be split into :-

3. *Process Management* :: Creating and destroying processes and handling their connection to the outside world (IO). Communication among different processes (through signals, pipes, IPC). The scheduler, which controls how processes share the CPU, is part of process management.

4. *Memory Mangement* :: The computer's memory is a major resource, and the policy to deal with it is a critical one for overall system performance. The kernel builds up a virtual address space for all processes. The different parts of the kernel interact with the memory-management subsystem through simple `malloc / free` and other more complex memory functions.

5. *FileSystems* :: Unix is heavily based on the filesystem concept; almost everything in Unix can be treated as a file. The kernel builds a structured filesystem on top of unstructured hardware, and the resultant file abstraction is heavily used throughout the whole system.

6. *Device Control* :: Almost every system operation eventually maps to a physical device. With the exception of the processor, memory and a very few other entities; any and

all device control operations are performed by code that is specific to the device being addressed. That code is called *Device Driver*. The kernel must have embedded in it a device driver for every peripheral present on a system (hard drive, printer, kbd, tape..)

7. *Networking* :: Networking must be managed by the operating system, because most network operations are not specific to a process. Incoming packets are asynchronous events. These packets must be collected, identified, and dispatched before a process takes care of them. The system is in charge of delivering data packets across program and network interfaces, and it must control execution of programs according to their network activity.

Device Driver

What is a Device Driver

1. A set of software procedures or apiś which enable higher level of programs (applications) to interact with hardware device

2. A Device Driver ”Controls”, ”Manages” and ”Configures” devices connected to the system

3. The driver ha few functions and provides the logic to initialize and communicate with the hardware

Device Driver

Device Driver Types

1. Service Device Driver :: If the device driver will be communicating directly with the hardware device, this interface will be the memory ranges, registers and/or ports through which I/O to the device takes place.

2. System Device Driver :: If the device driver will be communicating with its device via an intermediate device, this interface will be whatever APIs the driver for the intermediate device exposes. Eg. a GPS device driver will use the serial port driver API – the stream interface functions – to communicate with a GPS receiver.

Device Driver

Device Driver Need & Challenges

1. By providing a clear abstraction between the driver and the application, one can essentially free the application of the specifics of a certain peripheral and port it more easily to new hardware

2. Device Drivers have a tight connection to the target device and the development environment, these are usually not portable. Eg., device drivers across microcontroller families.

3. Device Drivers consist of a lot of "bit bashing" and register programming. One needs to get all the details right $\rightarrow$ the bits, sequences of initialization and exit, timing (say flash, ddr), or else the hardware would malfunction or not provide the desired output.

4. Device Drivers also enable Debug of system malfunction (due to modularization). E.g. 85% of failures in WinXP were from buggy device drivers.

5. GROUP DISCUSSION :: So, How would you go about developing a Device Driver?

Device Driver

Device Driver Development → HardWare Side

1. Reading the hardware manual and learning the chip internals

2. Learn about the hardware / platform. Identify the interface between the driver and the device

   (a) How the device is reset

   (b) How the device is "address mapped"

   (c) Return codes and software protocols recognized by the device

   (d) What are the types of DMA transfers possible with the device

   (e) How the device reports hardwar failures

   (f) How the device sends / responds to interrupts

3. Test the hardware to make sure it is functioning. This becomes important for a newly developed device.

Device Driver

Device Driver Development → SoftWare Side

1. What kind of device driver library will enable the application ?
   (a) Identify the APIs that the driver must expose
   (b) Initialization and De-initialization routines (eg. setting up of baud rate or timer periods)
   (c) Run time control (Read, Write, Interrupts, Signal Responses..)
   (d) Notification and any additional interfaces that the Device Driver needs to use or other OS APIs or services
   (e) Version Control for Software
   (f) Bug Tracking and closure utility
   (g) Suite of testcases to test scenarios where the Device Driver is used
   (h) Integration into the System and the OS Kernel

Device Driver

Device Driver Development → SoftWare Side

1. Monolithic device driver library
   (a) Based on a single piece of code that exposes hardware devices functionality directly to the OS
   (b) Accesses device(s) directly
2. Layered device driver library
   (a) A model device driver based on an "upper" layer which is logical and "lower" layer which is physical
   (b) RTOS abstraction layer
   (c) Compiler abstraction layer ?
   (d) Hardware abstraction layer ?

Device Driver

Device Driver Development → SoftWare Side PROs and CONs of Monolithic Vs Layered

1. Monolithic :: Performance is better as it avoids calling many functions.

2. Layered ::

   (a) Because of modularity, the structure of software is easy to understand.

   (b) It is easy to add or modify features of the overall application as it evolves and gets deployed.

   (c) Because there is one module that ever interacts directly with the peripherals registers, the state of the hardware device can be more accurately tracked.

   (d) Software changes that result from hardware changes are localized to the device driver, making software more portable.

   (e) Enhance the reusability, but bit of extra effort up front, at design time, in order to realize the savings.

Introduction to Video Processing and Device Drivers

Acknowledgements

1. An Introduction to MPEG :: School of Computer Science :: U Central Florida :: VLSI and M5 Research Group :: Tao Tao

2. Class Notes and Slides from Puneetha Mukherjee

3. Linux Device Drivers :: 3rd Edition