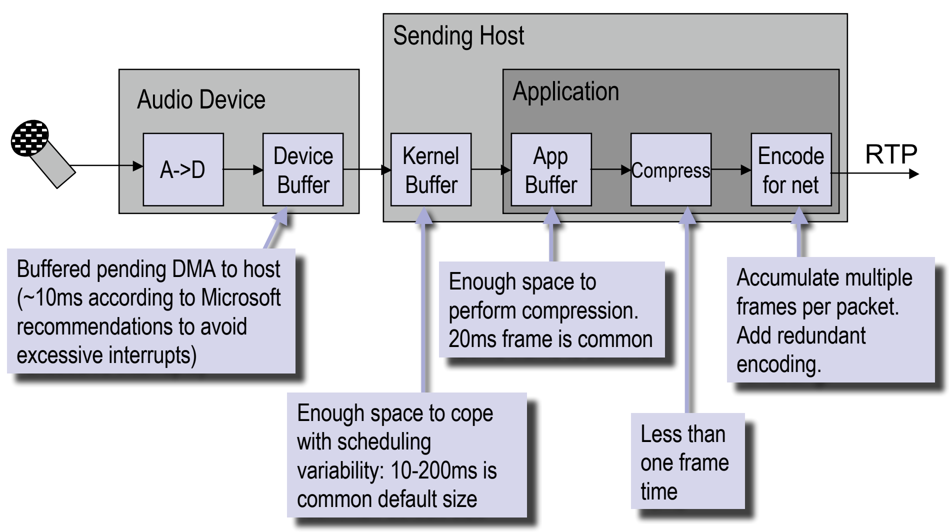
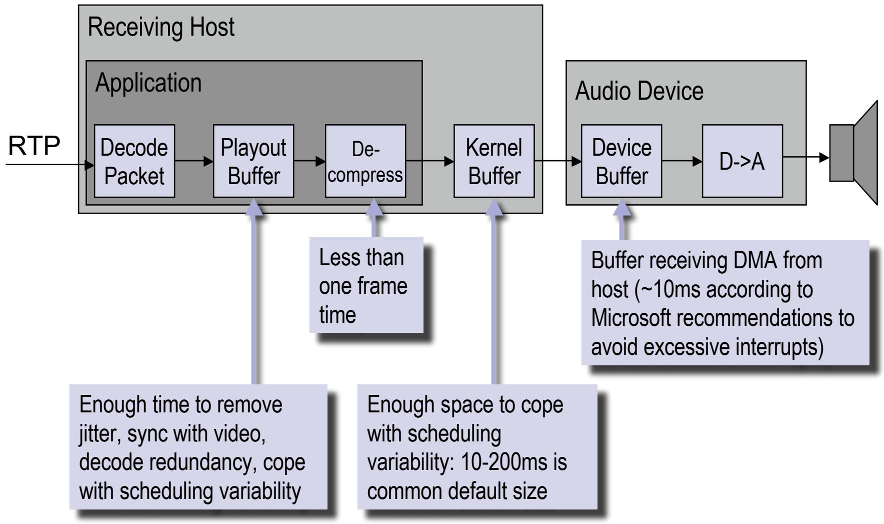
15: OS Scheduling and Buffering

Mark Handley

Typical Audio Pipeline (sender)



Typical Audio Pipeline (receiver)



Interrupts

- Audio device captures sample-by-sample.
  - Writes to a buffer in the device.
- Every so often, needs to transfer a block of data to host.
- Two ways:
  1. Send interrupt. Host copies data.
  2. Use Direct Memory Access (DMA). Interrupt on completion of DMA transfer.

DMA is more common for A/V devices.

Interrupts

- Signal from a device to the CPU.
  - Eg audio device, video capture card, disk, timer, etc
- Causes the CPU to stop what it was doing, switch into kernel mode, preserve state, and then execute an *interrupt handler*.
  - Interrupts may be disabled to stop interrupts interrupting the interrupt handler.
  - Handler must be quick to avoid other interrupts being processed too late.
  - CPU copies data from device buffer across the bus to kernel memory.
- Can't interrupt too many times per second without performance problems.

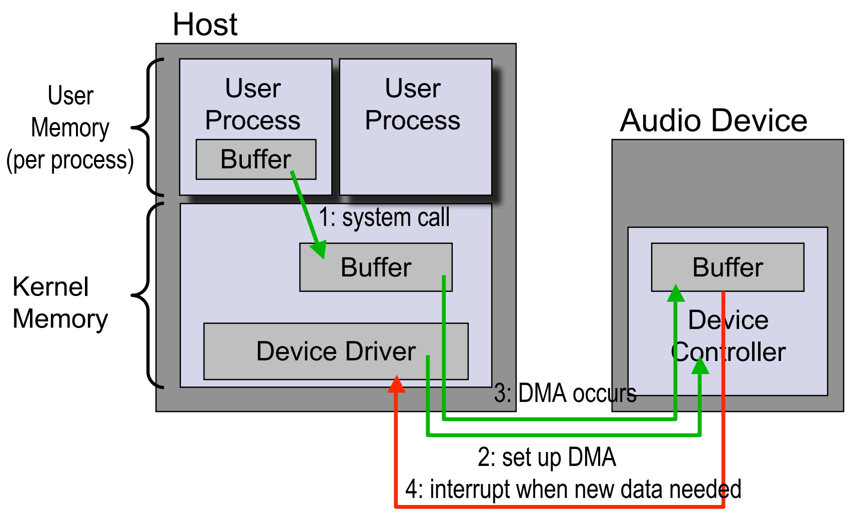
DMA

Programmed I/O wastes the CPU's time copying data

DMA (Direct Memory Access) allows the device to copy data to main memory *before* interrupting the CPU.

1. CPU sets up the DMA chip:
   - How many bytes to transfer, the device and memory addresses, direction of transfer.
   - Says "go".
2. Device reads/writes memory directly over the bus without bothering the CPU.
3. When DMA chip is done, it causes an interrupt, which is handled as before.

Kernel vs User Space



Processes and Threads

A ***process*** is just an *executing program*, together with the values of its program counter, variables, registers, and memory.

- Conceptual model is that multiple processes run in parallel.
- In reality, the CPU switches between them rapidly.

A ***thread*** is the *unit of scheduling* for the OS.

- One process can be comprised of many threads.
- Threads within a process share memory.
- When a thread issues a system call that can't immediately complete, it blocks. The OS then runs another thread.

Scheduling

Non-preemptive scheduling

- OS lets a process run until it blocks or voluntarily gives up the CPU.
- Process gets predictable performance once its scheduled.
- Misbehaving process can take all of CPU.

Preemptive scheduling

- Clock interrupts occur every ~10ms (allows OS to run)
- OS lets a process for a certain about of time (eg 100ms).
- Then suspends it and switches to another process.
- Goal is to make multiple processes seem to run simultaneously.
- Eg Windows, Linux, MacOS X.

Preemption and Multimedia

If the OS lets something else run for 100ms, what happens to an audio application?

Sender:

- Kernel audio buffer fills. DMA stops. Audio device buffer fills, so samples are discarded.

Receiver:

- Kernel audio buffer empties. Silence is played out. Abrupt transition to silence can result in loud clicks.
- More audio packets arrive. Adds to perceived jitter - need to remove this using playout buffer.

Either make sure kernel buffer is large enough, or make sure audio application gets scheduled often enough.

Real-Time Systems

- Hard Real-Time:
  - there are guarantees that MUST be met.
- Soft Real-Time:
  - deadlines should be met, but no hard guarantee.

- Hard real-time processes generally short, predictable, and run to completion quickly.
- Scheduler handles external events so as to ensure that all guarantees are met.

Scheduling in Real-Time Systems

- Given a system with *m* periodic events, and where event *i* occurs with a period of P_i seconds and requires C_i seconds to process.
- The load can only be handled if:

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

- Such a system is said to be *schedulable*.
- Periodic events might be audio data read/write, or video frame capture.

Real Systems: Windows 2000

- Priority-based, preemptive scheduling of threads.
- 32-level priority scheme determines execution order.
  - Highest priority runnable task is run first.
  - Two classes:

    real-time: priorities 16-32

    variable: priorities 1-15
  - For variable class processes:
    - Priority is reduced when a quantum expires.
    - When unblocked, priority is boosted depending on why it was blocked. Eg: keyboard event gives high boost when gives good interactivity.
    - Foreground process on screen gets higher priority.

Embedded Systems

Desktop OSes not originally designed for multimedia.

- Can work well, but you need to be smart.
- Lots of places to accidentally add to delay.

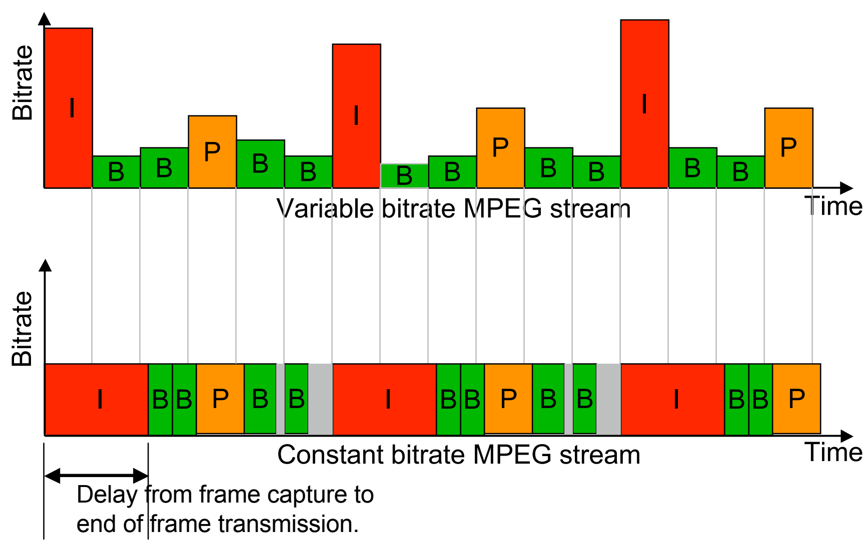
Simple Embedded Systems:

- Single memory image, non-preemptive.
- Much simpler to minimize delay.
  - Fewer competing demands on CPU.
  - No context switch overheads.
  - Application doesn't need to hedge against scheduler.
  - Schedule application every 20ms as soon as audio data is ready. Let it run to completion.

Network Smoothing

- Codec output rate may be variable:
  - I frames > P frames > B frames.
  - Variable motion content.
- Network may demand constant bitrate output
  - Eg H.221 ISDN, DVB-T
- Need to smooth the variable codec output to fit the constant network capacity.
  - Use a buffer on output.
  - If buffer starts to fill, adjust quantization, etc.
  - Buffering adds delay.
- May also need to smooth for packet net - bursty traffic may be dropped.

Network Smoothing



Shared Memory

For video, copying uncompressed data multiple times will seriously impact performance.

- Shared Memory
  - Processes communicate via a memory segment that is accessible to more than one process.
  - Eg. Shared-memory X
- Memory Mapping
  - An area of memory used by the kernel and/or device can be mapped directly into user space.
  - Eg video framebuffer.