+Streaming media, telephony and control (anti-lock braking) all require real-time computing.

**Hard real-time** : A missed deadline is a system failure. (eg nuclear power systems, avionics, etc)

**Soft real-time** : A missed deadline is a loss of service quality. (eg streaming audio & video)

[More here](https://stackoverflow.com/questions/17308956/differences-between-hard-real-time-soft-real-time-and-firm-real-time)

Real-time requirements:

* An event should not occur **until** a given time has elapsed
* An event must take place **before** a given time has elapsed
* A certain computational activity must have taken place **within a specified period** of time
* Responding to one event must take **priority** over another event

Scheduling

Given

* the *‘readiness’* of certain tasks
* knowledge of *how long* they take to *execute*
* knowledge of their respective *deadlines*

We can determine which task to execute next.

Responding to external events

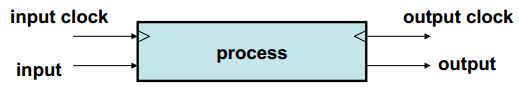
**Polling:** Cycle through possible input events until an active (waiting) input is found. However this wastes a lot of power.

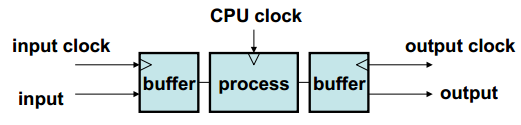
**Interrupt:** An event flag causes the processor to switch attention to handling the event. However does incur context-switch (storing/restoring system state) overheads.

**DMA (Direct Memory Access):** Move the data for processing later. Allows memory access without interrupting the processor.

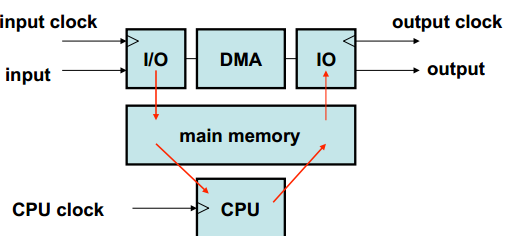
Buffering

Handling individual inputs and outputs in real time is near-impossible.



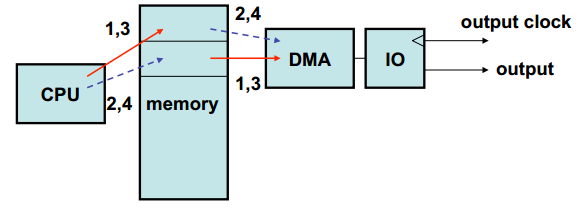
Buffers allow the CPU to process blocks of data handling both input and output timing.

In practice we may use DMA to transfer data to buffers in main memory.



Double buffers

We assemble data into blocks, work on one block while the other’s data is sent to output, then switch over to work on the other block while the first is sent to output. A good analogy for this is described as filling up buckets with water over on this [wikipedia page.](https://en.wikipedia.org/wiki/Multiple_buffering)



Timing and Watchdog Timers

Many processor systems will incorporate one or more **timer/counters.**

* Run off a system clock
* Can be programmed to interrupt
  + after a certain time
  + at regular intervals
* Can generate output events at precise times
* Provides the processor with a real-time reference

But what happens if there is a glitch and the system crashes?

A **watchdog timer** can be used to interrupt or reset the system.

1. Normally the software resets the watchdog regularly before it triggers
2. If the software crashes, the watchdog will (eventually) trigger
3. Functionality is restored, perhaps after a brief loss of real-time performance.

Real-time communications

Sending real-time data streams over unreliable, variable-delay networks. Example protocol: RTP (Real-time Transport Protocol)

What can we do about dropped packets? Resending is not an option as they will arrive too late.

RTP sends packets with:

* Sequence number (to identify what is missing)
* Timestamp (to get accurate real-time output, synchronisation of audio/video streams)
* However there is no flow control, error control, acknowledgement, or retransmission request (RTCP provides this- but you don’t need to worry about this: COMP28411 Networks stuff)

RTP interpolates to approximate lost data.

Power-efficiency

***In a real-time system there are no bonus points for finishing early!***

When there are no events to process, the CPU should go into a low-power mode.

An interrupt will wake a CPU up, which then processes the event, then goes back to sleep.

An **event-driven** model of computation.

Real-Time Operating Systems

Special class of OS for real-time systems.

Features **event-driven priority scheduling**. Switchest task only when an event of higher priority needs service.

Task states (these should look familiar from COMP25111):

* **running -** currently executing
* **ready -** will run when all higher-priority tasks have been completed
* **blocked -** can’t run until some *enabling event* occurs (most tasks are blocked most of the time)

Context switch

A major issue in real-time computing is *context-switching overhead.*

Saving the state of the old task, loading state of the new task and then restoring the state of the old task back can take 10s to 100s of CPU cycles.

Interrupts cause an automatic context switch for the interrupt handling process. Often optimised by *dedicated hardware* resources. May cause a preemptive task switch.

Co-routines

An RTOS may *reduce* context switch overhead through the use of coroutines.

* We share the stack and other process state
* Employ cooperative multi-tasking, instead of preemptive
* A task will hand over control to a higher-priority task at a “mutually convenient time”
* Must observe various programming restrictions

Summary

* Mobile systems often have real-time computation requirements
* Radio communication is a particular problem, because of unavoidable packet loss (the network is unreliable)
* However *careful system design* minimises problems.
  + buffering, DMA, scheduling, coroutines