1 Printed Circuit Board

- Copper plated fibreglass manufactured with photolithography (etched with UV light)
- Typical thickness 1.6mm, 1.2mm
- Holes drilled for components
- Copper removed to create wires
- Single, Double, Multilayer types

Soldermask
Epoxy finish that protects copper

Silkscreen
Finish used to create labels, lines and designators

Materials

- FR4 (standard)
- Ceramic
- Aluminium
- Flexible

Surface Finishes

- HASL with lead (standard)
- HASL lead free
- Immersion Gold (Flat, good for Pick and Place)
- Hard Gold (Good for connectors, does not scratch)

Vias
Used to connect layers

- Buried Vias
- Through Vias
- Blind Vias

Vias should be tented when underneath components to prevent accidental shorts

Traces
Used to connect components

- High frequency traces should be wider for less inductance
- High current traces should be wider for less resistance
- 45 degree traces save space and prevent reflections at high frequency signals due to the sudden change in impedance

Ground Plane

- Minimises inductance by providing a large plane (for high frequency signals)
- Minimises resistance to ground (for high current signals)
- Good to have one side with a large ground plane

1.1 High Frequency Considerations

Crosstalk

- Occurs when a high frequency signal induces a signal on a parallel track
- Capacitave coupling induces a positive spike that propagates away from the wave front
- Inductive coupling induces a negative spike propagating forward from the wave front, and positive spike propagating backwards
- Can pick mutual capacitance and inductance to cancel the forward wave
- Can source terminate to ensure no backwards reflection to receiver

Terminations

- A signal travelling down an unterminated wire will reflect doubling voltage on the way back (this is good for antennas but not much else)
- Can use the thickness of the track to match impedance with the input impedance of the signal receiving device

Track length matching

- Signals do not arrive at destination immediately
- When high precision timing is required, can create extra bends in track to ensure the signals arrive at the same time

2 Components

2.1 Crystals

- Used to generate a clock for the MCU, essential for operation
- All transfers between registers occur on clock edges
- Piezoelectric material creates a charge when squeezed

Capacitors
The 18pF capacitors placed with the crystal put 'slack' in the system so that the crystal resonates with the system that locks onto the required frequency,

Placement
High frequency component, and hence requires short traces/low parasitic capacitance
Can be achieved by placing close to MCU, surrounding with a ground plane, and using a 'guard ring' of vias which absorb crosstalk and noise

2.2 MOSFET

Can be used to amplify the limited current from the MCU

Base Resistor
Put a resistor (15Ohm) in series with the MOSFETS gate to avoid the rush of current coming in when it is turned on (the gate can be treated initially as a capacitor).

2.3 Boost Converter

- A type of regulator that converts with much more efficiency than linear regulators (using a voltage divider to step down) which waste lots of power.
- Uses principles behind inductor (non-instantaneous current change) to create large voltage spikes which are filtered to create a voltage

Layout
Ensure the components are laid out close to each other to minimise parasitic capacitance and inductance will affect the operation of the converter. Solid planes around the converter are good

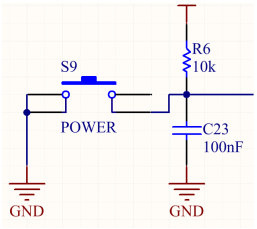
Switching regulators are also noisy so it is advantageous to keep the regulator subcircuit contained and away from the rest of the components

2.4 D Flip Flop

Clocks the output to the current input on a rising clock edge.
Can be used to provide soft power by feeding the output back into the input to let the flip flop toggle between states

2.5 Buttons

Debouncing Adding a capacitor to the switch slows down the change in voltage, removing any inherent bounce to the mechanical actuation



2.6 LCD

Input only, does not output anything and hence the the MCU must remember what is currently written to the LCD using a frame buffer

- Has 102 x 64 grayscale pixels = 6528 pixels/8 = 816 bytes
- There are 8 pages of 132 columns
- In normal orientation, the first 102 columns are visible
- In inverted orientation, lines 30 to 132 are visible

2.7 FRAM

Ferrous Random Access Memory

- Stores data bits in spins of electrons and is non-volatile (persistent through power cycles)
- Fast to write and does not degrade with number of writes
- Bidirectional interface, as it must output data when requested

2.8 Misc

Floating Inputs
Minimize these as they will take whatever voltage they come into contact with (e.g static from a finger) as opposed being pulled high or low

Decoupling Capacitor

- Absorbs fluctuations in supply voltage to components
- Should be placed as close to power pins as possible to minimize chance of noise/crosstalk occurring between the capacitor and the pin

2.9 Packaging

Integrated Circuits

- DIP - Dual Inline Package (Breadboard Opamp)
- QFP - Quad Flat Package (ATMEGA16)
- BGA - Ball Grid Array
- SOIC - Small Outline IC (FRAM, NAND)

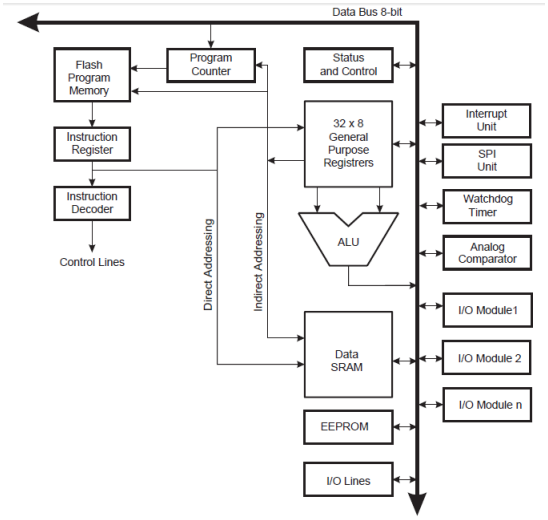
Passive Components
Resistors, capacitors, inductors, diodes

- DO-15
- SIL
- MCF
- 0603 (Like 0805 but smaller)
- 0805 (Used in game console)
- 1206 (Like 0805 but bigger

3 Microcontroller (ATMEGA16)

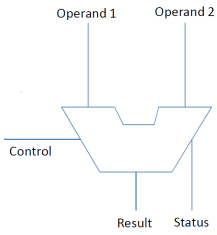
Instructions are stored in RAM, the program counter increments and a combination of logic gates processes the instruction

Architecture



Arithmetic Logic Unit (ALU)

- Takes two data inputs (operands)
- Takes one control input (add, subtract, etc)
- One data output
- One status output (signals the output is ready)



Reset Button
Restarts the program

Use an external pull up and possibly a capacitor to ensure no accidental resets due to noise

Sourcing vs Sinking Current
The microcontroller pins can sink more current than sourcing
IO pins can source a max of 40mA

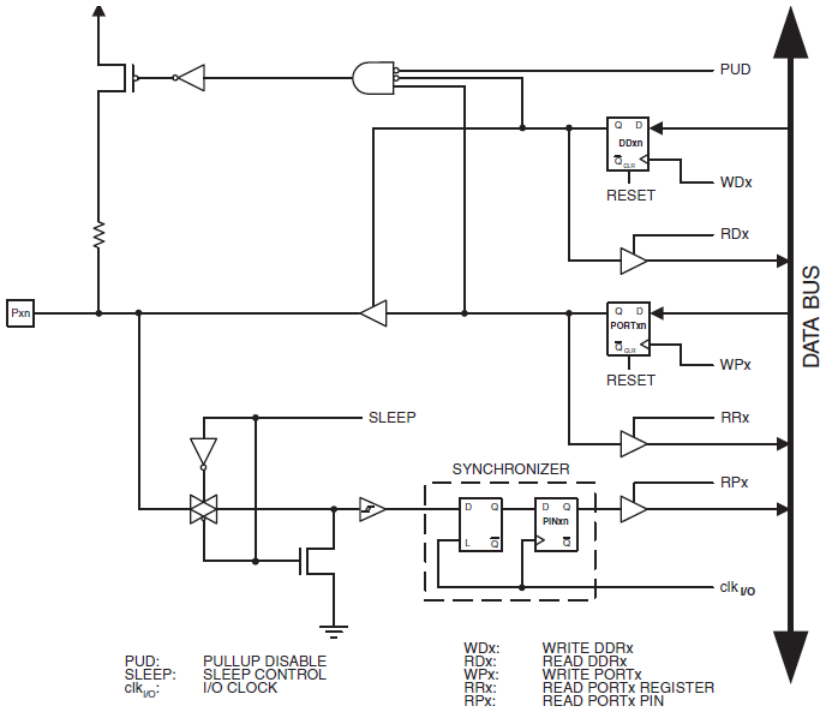
3.1 General Programming

Building Code

1. High level .c .h code gets converted into assembly code
2. Assembler forms an object file
3. Linker uses object file to find definition of each operator being used from the libraries and addesd startup code to form the .elf file
4. The .elf is converted into a .hex file which is given to the MCU via object copy

Memory Allocation
It is unwise to use malloc() and calloc() functions as microcontrollers don't have an operating system that makes sure important reserved memory is left untouched.

3.2 IO Pins



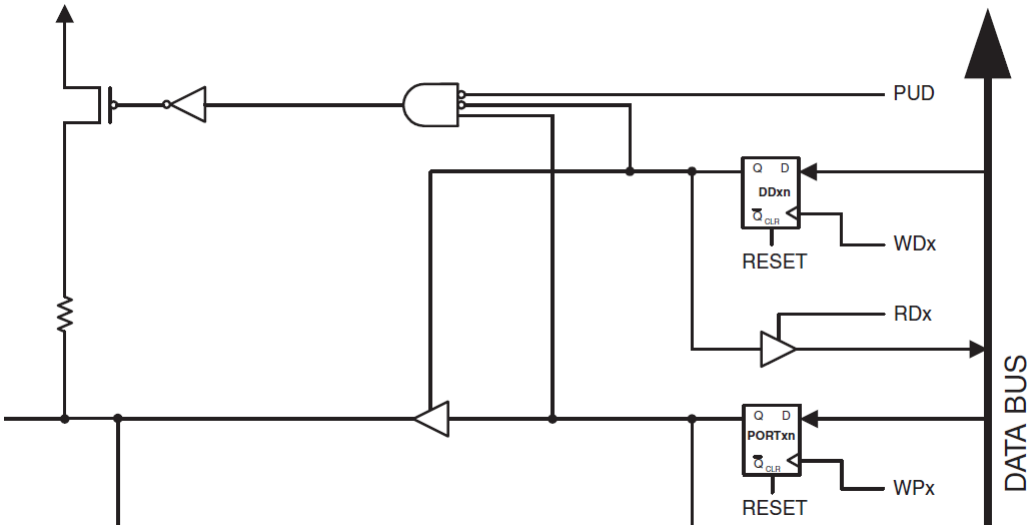
Maximum Current
Each port is only capable of sinking/sourcing 100mA of current.
The max DC current per IO pin is listed as 40mA.

Writing to an IO pin

- Set port direction register (DDRx, set as input by default)
- Set PORTX (register that stores data available on physical pin on WPx rising edge)
- Can use a mask to set individual bit

Pull Up Register

- Disabled by default
- A MOSFET shorts the pin to a high voltage through a resistor when activated



Synchronizer
Uses two flip flops to decrease probability of an input change on clock edge causing metastability
Comes at the cost of an additional clock cycle

3.3 Firmware

3.3.1 Analog to Digital Converter

Takes an analog input and outputs a digital representation

Quantisation Bits
The ADC is capable of 10 bit (1024) conversion from the ADCL/ADCH registers or alternatively 8 bits (256) from the ADCH register if the ADLAR (ADC Left Adjust Result) bit is set to 1

Prescaler
Reduces a higher frequency clock (i.e the 8Mhz crystal) to allow the timer to be clocked at a desired rate.
By default, the successive approximation ADC requires an input clock frequency between 50kHz and 200kHz for maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200kHz to get a higher sample rate

Reference Voltage
Can set the desired reference voltage for the ADC in registers

- Internal 2.56V
- AVCC pin (Same voltage as VCC pin with noise isolated externally)
- AREF pin

Initialising the ADC

1. Set the desired pin in the ADMUX register
2. Set the ADMUX register to use the internal 2.56V reference
3. Set ADEN on (Enable)
4. Set ADSC (Start conversion)
5. Results will go to ADCH/ADCL (ADC Data Register)

3.3.2 Interrupts

- Stops execution of current code and jumps to a specific point in memory when a trigger happens
- Good for power saving
- Makes testing hard as code can branch at any point
- Nested interrupts can be dangerous and consume all RAM
- Good practice to keep short as possible

Interrupt Service Vectors
ATMEGA16 has 21 interrupt vectors and 2 bytes per vector, it has a minimal ISR that only stores the program counter

- 4 clock cycles to store program counter and branch
- 3 clock cycles for a direct jump
- 4 clock cycles to return from interrupt

Timer Interrupts

- More reliable than while loops for creating events at a given time
- Ensure that the previous task is completed before the next timer interrupt or else risk stack overflow

Interrupt Service Routine
What happens during an ISR

1. Interrupt arrives at interrupt unit
2. Global Interrupt Enable (GIE) is disabled (can be re-enabled manually)
3. Program counter is pushed to stack
4. Processor jumps to interrupt vector
5. Interrupt vector has jump instruction to ISR
6. Once ISR is complete it runs a return from the interrupt instance
7. Returns the previous program counter from the stack
8. Enable GIE
9. Execute one more instruction before servicing any more ISRs (to prevent the program never moving forwards)

Interrupt Types

- Sets a flag - if GIE is not set, this will store the interrupt for later
- Triggered when active - If GIE is not set it will not trigger unless set active

3.3.3 Serial Peripheral Interface (SPI)

- Synchronous communication protocol
- Supports multiple slaves
- Short distance due to high frequency clock line

Pins

- MOSI - Master Out Slave In
- MISO - Master In Slave Out
- SCK/SCL - Clock, controlled by master
- CS/SS - Slave select

Procedure

1. Master pulls slave select low for intended recipient
2. Master supplies a clock
3. Data is sampled on clock edge

```
// Sends a data byte
byte LCD_data_tx(byte tx_byte) {
        byte tx_processed;
        byte tx_mask = 0x80;
        LCD_CHIP_SELECT;
        LCD_DATA;
        while (tx_mask != 0x00) {
                tx_processed = tx_byte & tx_mask;
                SCK_SET_HIGH;
                if (tx_processed) {
                        MOSI_SET_HIGH;
                }
                else {
                        MOSI_SET_LOW;
                }
                SCK_SET_LOW;
                tx_mask>>=1;
        }
        SCK_SET_HIGH;
        LCD_CHIP_DESELECT;
        return(TRUE);
}
```

Hardware SPI
SPCR = Serial Peripheral Control Register
SPDR = Serial Peripheral Data Register
SPSR = Serial Peripheral Status Register

1. Set slave or master (SS pin direction)
2. Enable SPI (SPCR)
3. Set Master/Slave (SPCR)
4. Set Clock frequency (prescaler) (SPCR)
5. Set Operating mode (when data is sampled, idle state of SCK, etc) (SPCR)
6. Set interrupts enabled/disabled and write ISR if needed
7. Write data to SPDR
8. Wait for SPIF to become 1 (SPSR) to indicate finished (if using interrupts, no need)
9. Read data from SPDR if intending to receive data

3.3.4 LCD Communications

In order to write a pixel, the page and column must be selected before writing the data bit.

Select Page
Set page address 1011ABCD where ABCD is the binary address of the page

```
#define CMD_PAGE 0xB0
select_page (byte page) {
        byte page_cmd_address;
        page_cmd_address =(CMD_PAGE | page);
        LCD_command_tx(page_cmd_address);
        return(true);
}
```

Select Column
Need to send two bytes to select a column, one each for the MSB/LSB.
Command sequence is 0000 EFGH for LSB, 0001 IJKL for MSB

```
#define CMD_COL_LSB 0x00
#define CMD_COL_MSB 0x10

select_column (byte column) {
        byte page_cmd_address_MSB;
        byte page_cmd_address_LSB;
        page_cmd_address_LSB =(CMD_COL_LSB | (column&0x0F));
        page_cmd_address_MSB =(CMD_COL_MSB | (column >> 4));
        LCD_command_tx(page_cmd_address_LSB);
        LCD_command_tx(page_cmd_address_MSB);
        return(true);
}
```

3.3.5 Timers

There is one 8 bit timer (Timer0) and two 16 bit timers (Timer1, Timer2) on the ATMEGA16

PWM
PWM outputs can be created using the inbuilt timer.
Need to set:

- Direction of the desired pin to out
- Clock source (clock, prescaling, external source)
- Waveform Generation Mode
- Inverting/Non Inverting - Inverting mode pulses at the beginning of a period, non inverting mode pulses at the end of a period
- Compare Output Mode (disabled, toggle, clear, set on compare match)

Registers

- TCNT - The Timer/Counter Register, counts system clock ticks, prescaled system clock ticks, or the external pin
- TCCR - Time/Counter Control Register, used to set mode, prescaler, if the Compare Output pins are connected to the timer, PWM. Split into two registers A/B for each of Timer1, Timer2

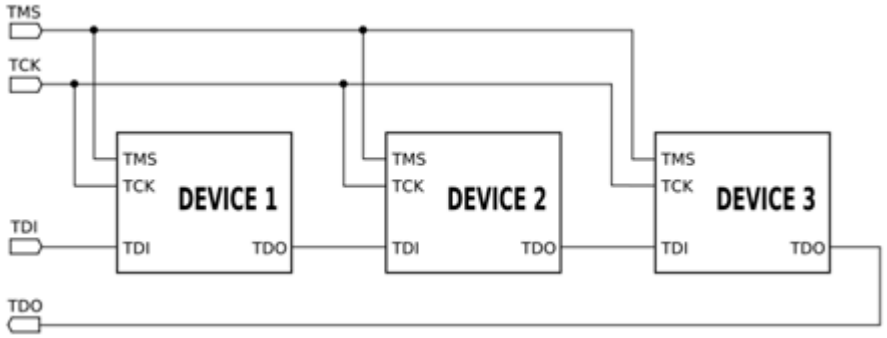
- OCR1 - Output Compare Register - used to generate an interrupt after the number of clock ticks written to it matches the value in TCNT1
- ICR1 - Input Capture Register - Measures time between pulses on external Input Capture Pin
- TIMSK and TIFR - Timer Interrupt Mask Register and Timer Intterupt Flag Register are used to control which interrupts are valid by setting their bits in TIMSK and to determine which interrupts are currently pending (TIFR)

Waveform Generation Mode

- Normal - The timer will trigger the overflow interrupt when it reaches overflow. Need to load a start value manually
- CTC - Clear on Timer Compare, the counter counts up to a configurable TOP value and then starts with 0 allowing for more granular timer overflow frequency than just prescaling
- Phase Correct PWM - Counts up to the TOP, and then down to zero, and up again, etc. This ensures a 'centered' waveform
- Fast PWM - The pulse waveform is set high when the timer reaches the output compare value, and is then reset at zero when the count overflows. Duty cycle is variable with output compare but not the PWM frequency

3.4 JTAG

Joint Test Action Group, used for accessing on chip debugging information (registers, RAM, counter) and programming flash.
Supports many IC's in a chain, where the output of TDI is connected to the TDO input on the next chip.



Architecture

TDI - test data in
TDO - test data out
TMS - test mode select
TCK - test clock
TRST - optional test reset

4 Miscellaneous

4.1 Oscilloscope Probes

- Intermittent problems can go away due to the added load/capacitance of the probe lead
- Large loops can pick up background noise
- In very high speed applications special connectors can be used to connect GND and signal with minimal loop
- Auto set button can set gain very high and you might just be picking up background noise (50Hz)

Wires
Using wires as oscilloscope probes is not very accurate

- High line capacitance (square wave can become sine wave)
- Large loops will pick up external noise
- Adds a 1MOhm resistor and 13pF capacitor + parasitic line capacitance in parallel to the signal being measured

Oscilloscope Probes

- A 10x probe increases the resistance seen by the device to 10MOhms
- A 9MOhm resistor is added in series to create a voltage divider (Hence 10x signal = true value)
- A capacitor is added in series to divide by 10AC signals. This cancels RC and provides a flat response

5 Real Time Systems

Correctness of system behaviour depends not ononly on the logical results of the computation but also on the physical time when these results are produced

Examples:

- Flight Control
- GPS
- Transport System

Functions

- Data Collection
- Direct Digital Control
- Man-machine interaction

5.1 Controllers

Low-level

- Workload is purely or mostly periodic
- Runs on one computer or a few computers

High-level

- Handles sporadic events and operators commands
- Large distributed system

5.2 Deadlines

Hard Deadline

- Severe consequences if missed
- Examples: Flight control, Traffic Signal

Firm Deadline

- A result has no utility beyond deadline
- Examples: Weather forecast, real time transport update

Soft Deadline

- A result has utility even after the deadline has passed
- Examples: Video streaming

5.3 Hard vs Soft Real-Time Systems

Characteristics	**Hard Real-Time**	**Soft Real-Time**
Response Time	Hard (required), milliseconds	Soft (desired), seconds
Peak load performance	Predictable	Degraded
Control of pace	Environment	Computer
Safety	Critical	Non-Critical
Size of data file	Small/Medium	Large
Redundancy Type	Active	Checkpoint-recover
Data integrity	Short-term	Long-term
Error Detection	Autonomous	User

6 Real Time Systems & Time Measurement

Jitter (ϵ)
Difference between maximum and minimum values of the delay of the computer
Should be a small fraction of the delay

Dead Time
Time interval between observation and the start of a reaction of the controlled object

Minimal Error-Detection Latency
Error detection latency must be in the same order of magnitude as the sampling period of the fastest critical control loop for it to be possible to take action in time.

6.1 Dependability

Reliability

- $R(t)$ of a system is the probability it will provide the service until time t
- Probability that a system will fail in a given time is measure in FITs (Failure In Time)
- Average time it takes a device to fail is MTTF (Mean Time To Failure)
- For a constant failure rate of λ failures/h the reliability at time t is

$$R(t) = e^{-\lambda(t-t_0)}$$

Safety
Defined as the reliability regarding critical failure modes, where the cost of a failure can be orders of magnitude higher than the utility of the system.

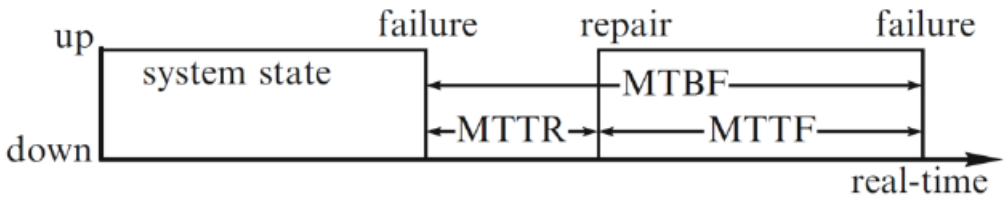
Maintainability

- Time taken to repair a system after failure
- $M(d)$ is the probability that a system is restored within a time interval d after the failure
- Mean Time To Repair (MTTR) is defined for a system with constant repair rate

Availability
Defined as the fraction of time that the system is ready to provide the service

$$A = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}$$

$MTTF + MTTR =$ Mean Time Between Failures (MTBF)



Security
Authenticity and integrity of information

6.2 Time

Instant Point in time

Duration Section of time between two different instants

Event Takes place at an instant of time and does not have a duration

7 Time Synchronization

Causes of time error

- Ageing of hardware
- Temperature
- Phase noise (interrupts, OS calls)
- Frequency noise
- Asymmetric delay (communication path delay)
- Clock glitches (sudden jumps

7.1 Time standards

TAI (International Atomic Time)
High precision atomic coordinate time standard
Weighted average of over 200 cesium clocks in laboratories worldwide

Coordinated Universal Time (UTC)

- Derived from TAI, used for timekeeping
- Defines grouping of seconds into minutes, hours, days, months, years
- Incorporates leap seconds to compensate for clock drift from solar time

GPS Time

- Not corrected to match the rotation of the earth
- Remains at constant offset from TAI
- Clocks tick faster about 38μs per day because of relativity

7.2 Terminology

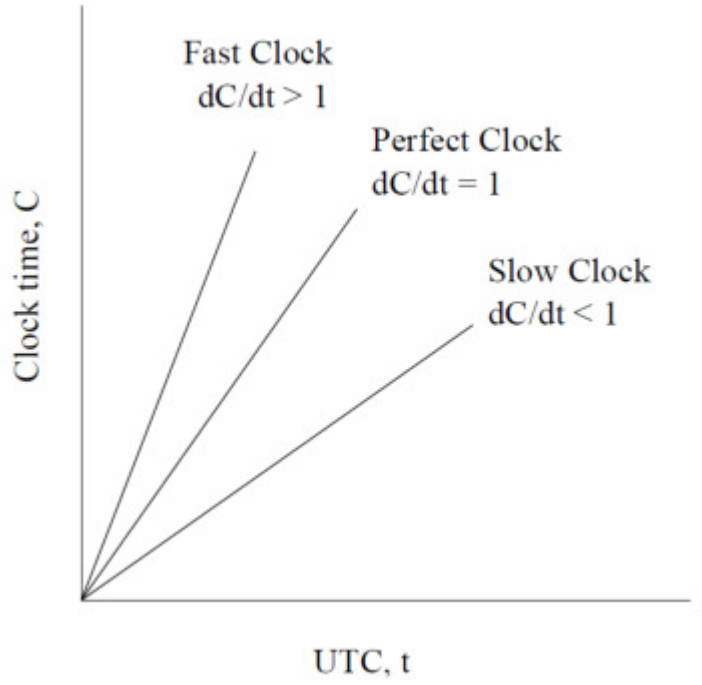
Time
The time of a clock in a machine p is given by the function $c_p(t)$. For a perfect clock $c_p(t) = t$

Frequency
Frequency at time t of a clock is $c'_a(t)$

Offset
Difference between clock time and real time
Given by $c_a - t$

Skew
Difference in the frequencies of the clock and the perfect clock
Skew of c_a relative to c_b is given by $c_a'(t) - c_b'(t)$

Drift (Γ)
Defined as the second derivative of the clock time, the rate at which the skew increases
Drift of c_a relative to c_b is $c_a''(t) - c_b''(t)$



Precision (Π)
Given a group of n clocks, the maximum offset between any two clocks of the group is the precision Π

Accuracy
Maximum offset of a given clock from the external time reference

Reference Clock
An external observer to which clocks in a group are measured against

Microtick
A periodic event measured in local clocks

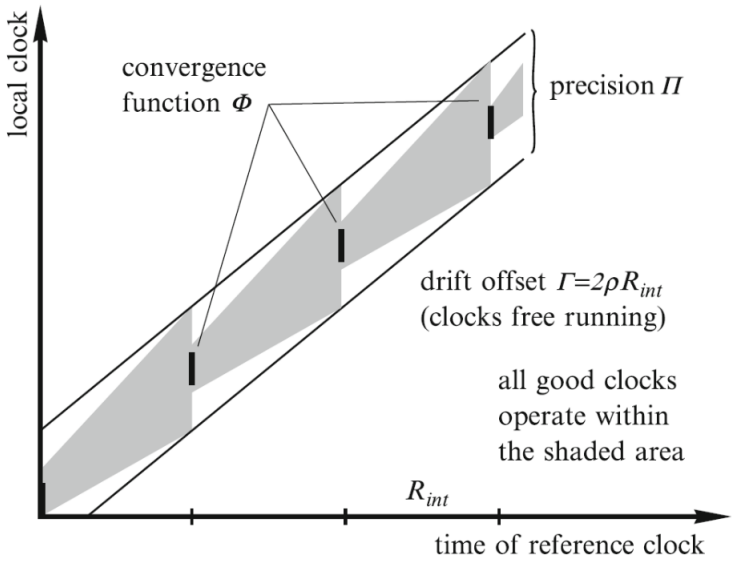
Granularity (g_z)
The duration between two consecutive microticks

7.3 Synchronization

Internal re-synchronization
Mutual re-synchronization of a group of clocks to maintain a bounded precision

$$\text{Drift Rate} = \rho$$

$$\text{Re-synchronization interval} = R_{int}$$



Synchronization Condition
A group of clocks can only be synchronized if the following condition holds between convergence function ϕ, drift Γ, precision Π

$$\phi + \Gamma \leq \Pi$$

Reasonable Condition
Global time t is reasonable if for all local implementations

$$g^{global} > \Pi$$

This ensures that the synchronization error is less than one macrogranule for any event e where $|t^j(e) - t^k(e)| \leq 1$

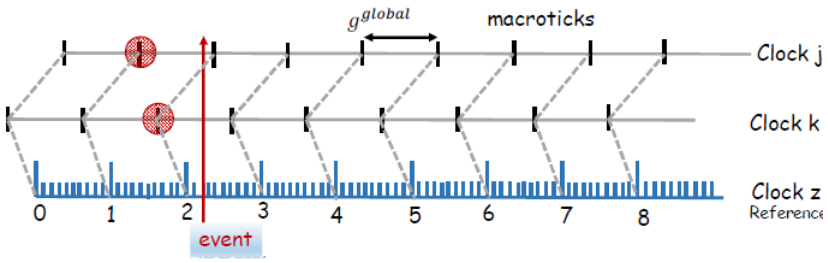
Impossibility Result
It is not possible to internally synchronize clocks of a group consisting of N nodes with precision Π and jitter ϵ to a better precision than:

$$\Pi = \epsilon\left(1 - \frac{1}{N}\right)$$

It can be seen a small jitter is important to achieve high precision.

External re-synchronization
Synchronizing with an external time reference
If all clocks of a group are externally synchronized with accuracy A, the maximum internal precision is 2A.

Global Time
Global time is an abstract notion as real clocks are not perfect, local clocks of nodes approximate global time.
Macroticks form the local representation of global time with granularity g^{global}



Assuming all clocks are internally synchronized with a precision Π then for any two clocks j, k and microticks i then:

$$|z(microtick_i^j) - z(microtick_i^k)| < \Pi$$

Fundamental Limits of Time Measurement

1. If a single event is observed by two different nodes, there is always the possibility that the time-stamps differ by one tick

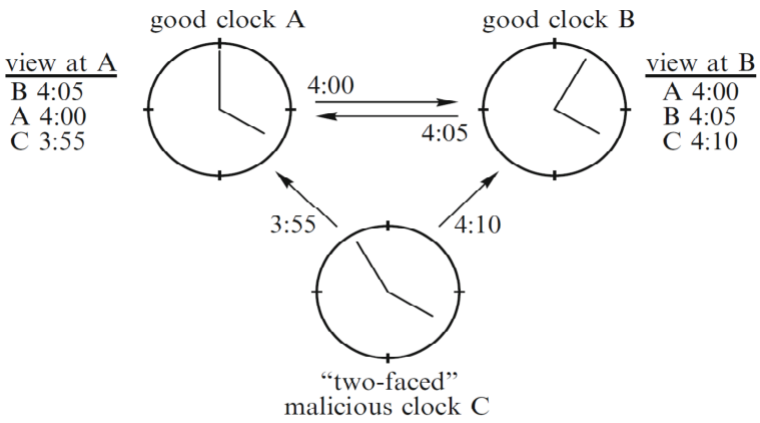
2. To reconstruct the temporal order of two events, the global timestamps of the events have to differ by at least two ticks.

$$|t^j(e_1) - t^k(e_2)| \geq 2$$

3. d_{true} of an interval is bounded by

$$(d_{obs} - 2g^{global}) < d_{true} < (d_{obs} + 2g^{global})$$

Byzantine Error
When a malicious clock sends different times to two good clocks, which calculate incorrect average times using this corrupted timestamp, which prevents the two good clocks from converging to an acceptable precision



Central Master Algorithm
Simple non-fault tolerant algorithm used in start-up phase of distributed systems

1. Master periodically sends the value of its time counter in synchronization messages to all nodes
2. The slave records the time-stamp of message arrival
3. Difference between the master's time and the recorded slave's time-stamp of message arrival
4. Corrected by the known latency of the message transport
5. This gives a measure of the deviation of the clock of the master from the clock of the slave

6. The slave then corrects its clock by this deviation to bring it into agreement with the master's clock

Precision of the algorithm is

$$\Pi_{central} = \epsilon + \Gamma$$

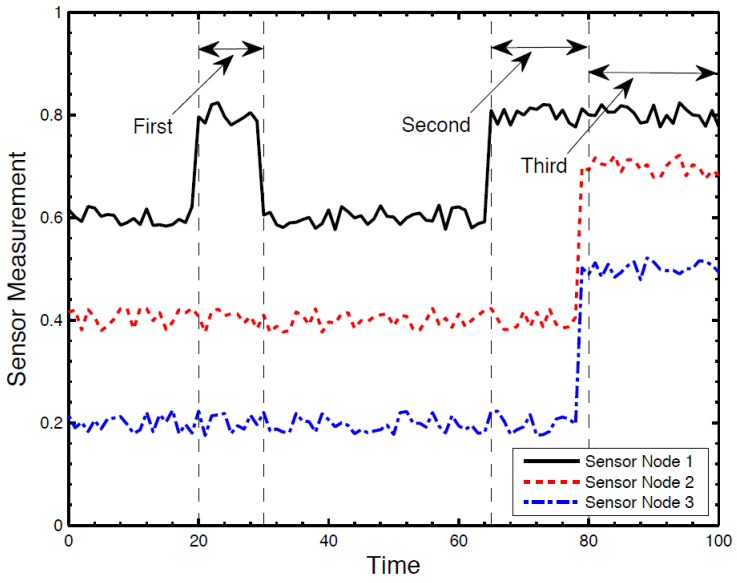
Fault-tolerant Synchronization
Every node acquires knolwedge about the state of the global time counters in all the other nodes and analyzes the information to detect errors, then executes the convergence function to calculate a correction value.

8 Errors, Faults and Anomalies

First Order Anomaly
Partial data measurements are anomalous at sensor node

Second Order Anomaly
All data measurements at a sensor node are anomalous

Third Order Anomaly
Data from a set of sensor nodes is anomalous

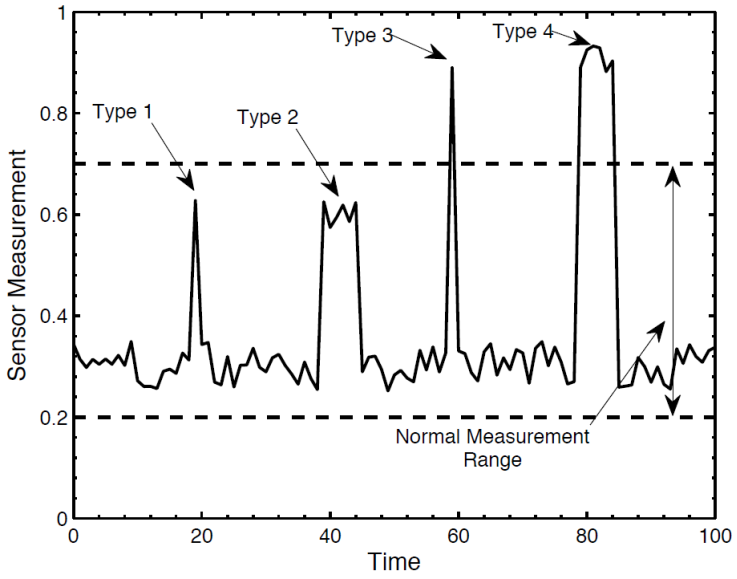


Type 1 Anomaly: Incidental Absolute Errors
A short-term extremely high anomalous measurement

Type 2 Anomaly: Clustered absolute errors
A continuous sequence of Type 1 errors

Type 3 Anomaly: Random Errors
Short-term observations not lying within the normal threshold of observations

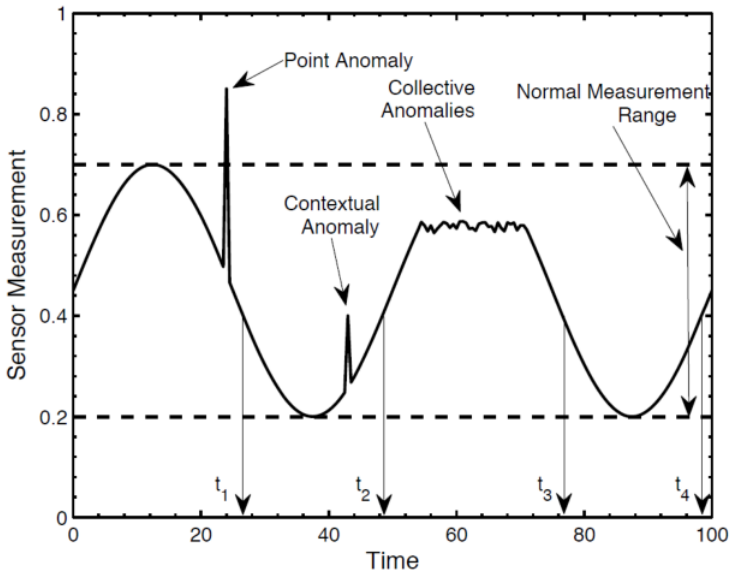
Type 4 Anomaly: Long Term Errors
A continuous sequence of type 3 errors



Point Anomaly
An individual data instance that is anomalous with respect to the data set

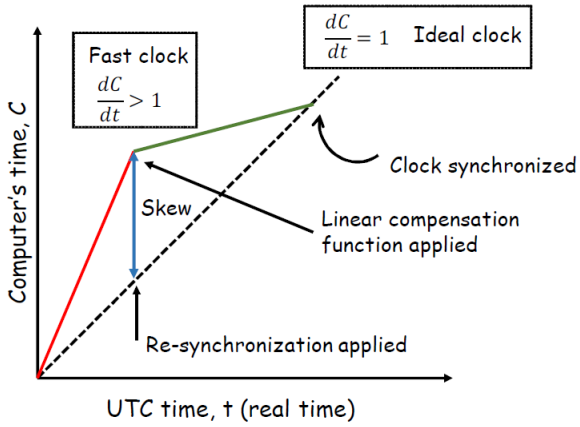
Contextual Anomaly
A data instance that is considered an anomaly in the current context, may be normal in another dataset

Collective Anomalies
A collection of related anomalies


Point Anomaly
Collective Anomalies
Normal Measurement Range
Contextual Anomaly
Sensor Measurement
Time
0
0.2
0.4
0.6
0.8
1
0
20
40
60
80
100
t1
t2
t3
t4

9 Time Synchronization in Distributed Applications

Compensating Drift
Not ideal to set clock back, will create confusion in message ordering. Either make clock slower or faster until it synchronizes to true time.
Can change the rate of interrupts/sampling due to change in slope of time



9.1 Cristian's Algorithm

Procedure

1. Request time from server
2. Time how long it takes for the server to respond with a message
3. Using the RTT (round trip time), set current time to the synchronization packet time plus half the round trip

$$T_p = T_{server} + \frac{T_1 - T_0}{2}$$

$$\text{Accuracy of the result} = \pm(\tfrac{T_1 - T_0}{2} - T_{min})$$

Where T_{min} is the channel delay

Limitations

- Suitable for deterministic LAN

- Single point of failure (server)
- Imposter server possible

9.2 Berkeley Algorithm

Each node runs a daemon, and the master polls each slave periodically. Using the RTT, it computes and broadcasts a new time.
The assumption is that the average cancels out the the errors of each individual node.
Master time must be set by an operator manually

Procedure

1. Master requests timestamps from all slaves
2. Compute a fault tolerant average, if a clock deviates by more than a specified limit it is ignored
3. Offset is sent to the slaves and master corrects its own clock

9.3 Network Time Protocol

Reliable for long periods of no connectivity
Clients synchronized frequency to offset clock drifts
Connected in a hierarchical server structure called a synchronization subnet

- Offsets usually < 128 milliseconds
- Polling interval is maximum 1024 seconds, Minimum 64 seconds

Procedure

1. Client sends timestamp $T0$
2. Server receives timestamp $T1$
3. Server transmits timestamp $T2$
4. Client receives timestamp $T3$

Transmission delay

$$\delta = (T4 - T1) - (T3 - T2)$$

Offset

$$\theta = \frac{1}{2}((T2 - T1) + (T3 - T4))$$

The minimum of the last eight delays δ_0 and the lowest offset θ_0 becomes the NTP update value (δ_0, θ_0). Using the derived offset the clock frequency is then adjusted gradually.

9.3.1 Modes

Unicast
Server to client (one-one)

Broadcast
Server to all clients, clients will be listening

Multicast
Similar to broadcast, but directed to specific clients

Manycast
Client sends manycast, if server replies then client switches to unicast and synchronizes

9.4 Wireless Devices

Synchronization is difficult because of non-deterministic delay

- Send time (constructing message and processing, dependent on load)
- Access time (waiting for transmit channel)
- Propagation time
- Receive time

9.5 Reference Broadcast Synchronization (RBS)

Can synchronize nodes to the resolution necessary for wireless sensor network applications

Scheme 1
Broadcast a single pulse to two receivers to estimate offset

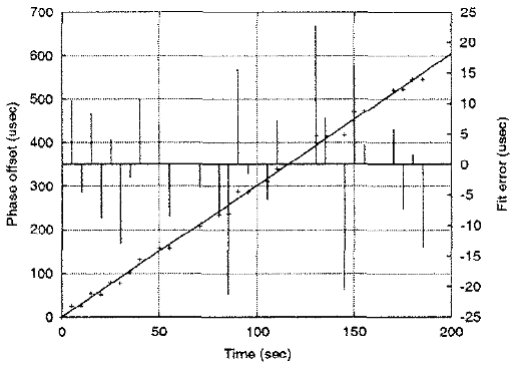
- Broadcast reference packet to two receivers (i and j)
- Each receiver records the time that the reference is received according to local clock
- The receivers exchange observations and form a relative timescale

Scheme 2
Increase precision of synchronization statistically. Does not account for clocks not running at the same rate (skew)

- Broadcast m reference packets
- Each of n receivers records the time that the reference is observed according to local clock
- Receivers exchange observations

Each receiver i can compute its phase offset to any other receiver j as the average of phase offsets implied by each pulse received by both nodes and i

Scheme 3
Instead of averaging the phase offsets from multiple observations, perform least-squares linear regressions to find a best fit line through the phase error observations over time.
Frequency and phase of local node clock can be recovered from the slope (skew) and intercept of line.



10 Timing, Precedence Relations & Scheduling Constraints

10.1 Terminology

Process
A computation that is executed by the CPU in sequential fashion
Synonyms: Task, Thread

Scheduling Policy
The assignment of the CPU to execute a set of concurrent tasks according to a predefined criterion

Scheduling Algorithm
The rules which determine the order in which tasks are executed

Dispatching
The act of allocating the CPU to a task

Waiting Task
A task waiting for CPU availability

Active Task
A task that can potentially execute , independently of its actual availability

Running Task
The task in execution

Ready Task
A task waiting for the processor

Ready Queue
The ordered list of next tasks to be dispatched

Preemption
Suspending the running task and inserting it into the ready queue.

- Used in exception handling
- Used when critical tasks arrive

- Allows higher efficiency
- Introduces a runtime overhead
- Destroys program locality

Task Priorities

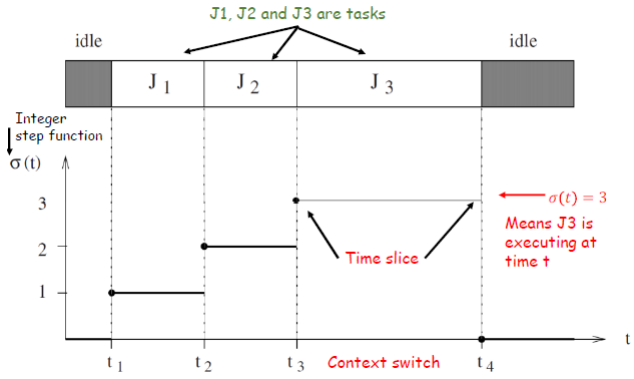
- **Hard tasks** should be guaranteed offline
- **Firm tasks** should be guaranteed online (abort if deadline cannot be met)
- **Soft tasks** should be handled to minimize average response time

10.2 Scheduling

Schedule
A set of tasks $J = \{J_1, J_2, \ldots, J_n\}$, an assignment to the processor that each task is executed until completion.
Can be defined as a function

$$\sigma : \mathbf{R}^+ \to \mathbf{N}$$

such that $\forall t \in \mathbf{R}^+ \exists t_1, t_2$ such that $t \in [t_1, t_2)$ and $\forall t' \in [t_1, t_2)$ $\sigma(t) = \sigma(t')$
$\sigma(t)$ is an integer step function and $\sigma(t) = k$ with $k > 0$, means that task J_k is executing at time t, while $\sigma(t) = 0$ means that the CPU is idle.



Preemptive Schedule
A schedule in which the running task can be arbitrarily suspended at any time to assign the CPU to another task according to a predefined scheduling policy

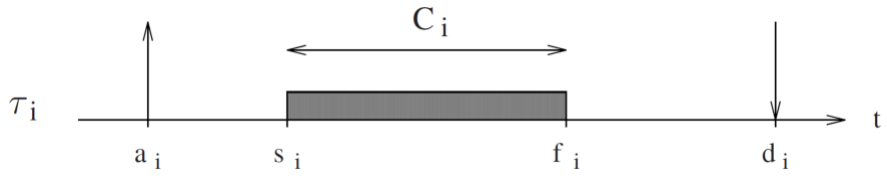
Feasible Schedule
If all tasks can be completed according to their constraints

Schedulable
If there exists at least one algorithm that can produce a feasible schedule

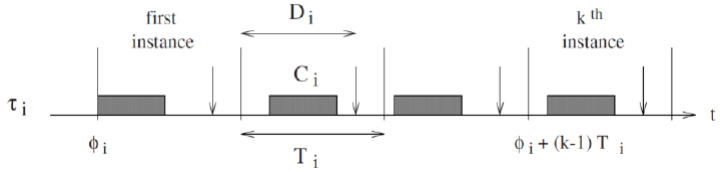
10.3 Tasks

A real time task τ_i can be characterized by the following parameters

Request time r_i
Computation time C_i
Arrival Time a_i
Start Time s_i
Finishing Time f_i
Absolute Deadline d_i
Response Time $R_i = f_i - r_i$
Relative Deadline $D_i = d_i - r_i$
Lateness $L_i = f_i - d_i$
Tardiness/Exceeding Time $E_i = \max(0, L_i)$
Laxity/Slack Time $X_i = d_i - a_i - C_i$



Periodic Task
An infinite sequence of identical activities activated at a constant rate
Phase ϕ: the activation time of the first periodic instance
Period T_i: The activation time of the kth instance



Aperiodic Task

An infinite sequence of identical jobs, activations are not regualrly interleaved

Sporadic task
An aperiodic task where consecutive jobs are separated by a minimum inter-arrival time

Precedence Constraints
Some tasks cannot be executed in arbitrary order, and must have other tasks performed prior

10.4 Resources

Any software structure that can be used by the process to advance its execution e.g a data structure, a file, a set of registers

Private resource
A resource dedicated to a particular process

Shared resource
A resource that can be used by more than one task

Mutually exclusive resource
A resource that cannot be accessed by a task if another task is manipulating it

Mutual exclusion is important because otherwise if the task is preempted while updating variables it may leave the buffer in an inconsistent state
Can use a semaphore to protect access to the resource, all tasks blocked on the same resource are kept in a queue associated with the semaphore protecting the resource

Critical section
A piece of code executed under mutual exclusion constraints

10.5 Scheduling Problems

Scheduling problems are difficult to computationally solve (NP Complete).

- Preemptive vs Non-preemptive
- Static vs Dynamic
- Offline vs Online

- Optimal vs Heuristic

Average response time is generally not of interest, most algorithms optimise for weighted (by a utility function) sum of completion times

Guaranteed Algorithms
Only accepts a new task if the new task set is found schedulable. If it is not schedulable, the task is rejected to preserve feasibility.

- Used in hard real-time applications where feasibility needs to be guaranteed in advance
- Due to the static nature, runtime overhead does not depend on the complexity of the algorithm
- If new tasks are created at runtime (Dynamic real-time system) the guarantee must be reverified on every task add

Best Effort Algorithm
Tries to do its best to meet deadlines

- No guarantee of finding a feasible schedule
- Tasks can be queued according to policies
- Tasks can be aborted during execution
- Perform better in average case compared to guarantee-based schemes

Utility Function
A function of the utility of the task at a given instance in time
Performance of an algorithm can be measured by the cumulative value

$$\text{Cumulative Value} = \sum_{i=1}^{n} v(f_i)$$

10.5.1 Common Cost Functions

Average response time

$$\bar{t}_r = \frac{1}{n} \sum_{i=1}^{n} (f_i - a_i)$$

Total completion time

$$t_c = \max(f_i) - \min(a_i)$$

Weighted sum of completion times

$$t_w = \frac{\sum_{i=1}^{n} w_i f_i}{\sum w_i}$$

Maximum lateness

$$L_{max} = \max(f_i - d_i)$$

Maximum number of late tasks

$$N_{late} = \sum_{i=1}^{n} miss(f_i)$$

$$miss(f_i) = \begin{cases} 0 & f_i < d \\ 1 & \text{otherwise} \end{cases}$$

11 Aperiodic Scheduling

Algorithm Classification

- α - the machine environment on which the task set has to be scheduled (uniprocessor, multiprocessor, distributed, etc)
- β - task and resource characteristics (preemptive, independent vs precedence constrained, synchronous activations. etc)
- γ - optimality criterion (performance measure) to be followed in the schedule

Example:

$$1 \mid \text{prec} \mid \text{Lmax}$$

Denotes the problem of scheduling a set of tasks with precedence constraints ona uniprocessor machine in order to minimize the maximum lateness

11.1 Earliest Due Date (EDD)

Scenario
A set J of n aperiodic tasks has to be scheduled on a single processor, minimizing the maximum lateness. All tasks consist of a single job, have synchronous arrival times, but can have different computation times and deadlines.

No other constraints are considered, hence tasks must be independent and cannot have precedence relations or share resources in exclusive mode.

Feasibility
Maximum lateness must be negative

Algorithm
Choose the task with the earliest due date
This is good for initialising a queue.

11.2 Earliest Deadline First (EDF)

Scenario
Precedence constraints and dynamic activations can exist

Algorithm
Choose the task with the earliest relative deadlines and preempt existing task

Precedence Constraints
Modify all release times and deadlines so that each task cannot start before its predecessors and cannot preempt their successors.
Given a task J_a which is an immediate predecessor of J_b

- The release time of J_b can be replaced by $\max(r_b, r_a + C_a$, as J_b cannot start before the minimum completion time of J_a
- In any feasible schedule that meets precedence constraints, the deadline of J_a can be replaced by $\min(d_a, d_b - C_b$, as the latest completion time for J_a to preserve feasibility must be such that J_b can be completed before deadline

12 Periodic Scheduling

Period tasks area major computational load in embedded systems, and involve activities needing to be cyclically executed at specific rates.
The OS has to guarantee that each periodic instance is regularly activated at its proper rate and meets deadline.

Procesor Utilization Factor (PUF)
The processor utilization factor U is the fraction of processor time spent in the execution of the task set.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

Least Upper Bound
For an algorithm A, the least upper bound $U_{lub}(A)$ of the processor utilization factor is the minimum of the utilization factors over all task sets that fully utilize the processor

$$U_{lub}(A) = \min U_{lub}(\Gamma, A)$$

- Any task set whos PUF is less than or equal to the upper bound is schedulable by the algorithm
- When $U_{lub} < U < 1.0$ the schedulability can be achieved only if the task periods are suitably related
- If the PUF is greater than 1, it is unschedulable

12.1 Time Scheduling

- Major cycle is equal to the least common multiple of all the periods
- To guarantee a priori that a schedule is feasible, it is sufficient to know the task worst-case execution times and verify that the sum of the executions within each time slot is less than or equal to the minor cycle

Limitations

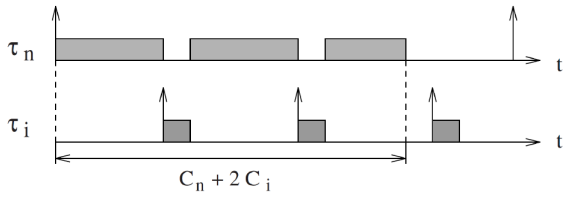
- Fragile during overload conditions, if a task does not terminate the minor cycle boundary it can be continued or aborted
- If updating a task requires increasing its time or frequency, the entire scheduling sequence may need to be reconstructed
- Difficult to handle aperiodic activities efficiently without changing the task sequence

12.2 Rate Monotonic Scheduling (RM)

Assign priorities to task according to request rates

- Higher request rates have higher priorities
- Priorities are fixed because periods are constant
- Preemptive, the current executing task is preempted by a newly arrived task with a shorter period

Critical instant
The arrival time of a task that produces the largest task response time



Optimality
Check task schedulability at critical instants. If all tasks are feasible at their critical instants, then the whole task is schedulable
Optimality is justified by showing that if a task set is schedulable by an arbitrary priority assignment, it is also schedulable by RM

Schedulability Analysis
For a set of n arbitrary periodic tasks, the least upper bound

$$U_{lub} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$$

If the inequality is satisfied then the tasks can be scheduled, otherwise schedulability cannot be concluded (dependant on the task periods being suitably related)

12.3 Earliest Deadline First

Can be used for periodic as well as aperiodic tasks

Schedulability
A set of periodic tasks is schedulable if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$