Embedded 1-12

Lecture 1

•Real Time system: Computer system that interacts w external events
•Embedded system: RTS that interfaces w external HW.

RTOS vs OS.

Common: Both are interfaces between HW & SW.
Differ: RTOS gives user access to low level system devices.

RTOS processes

- Periodic: system monitoring, polling, sampling
- Sporadic: event driven (interrupt)

Lecture 2

Livelock: Tasks are allocated CPU time by the scheduler but don't execute further instructions
Spinlock: Tasks are running & continuously polling on each other's unavailable ressources
Deadlock: Tasks are not running due to unavailable ressources.

CPU usage \ Resource usage	Yes	NO
Yes	Spin	Live
No	Dead	Starvation

Lecture 3

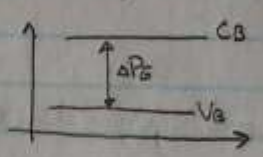
Sources of e^- Ep.

- Electric field $\sim \oint \vec{E} \cdot d\vec{0}$ → Most commonly used
- Magnetic field $\sim \vec{\mu} \cdot \vec{B}$ → Most efficient
- Heat $\sim nKT$ → most problematic
- Radiation $\sim hy$ → most flexible.

EMB

pol

tunneling: Quantom mecanical process used by transistors to allow electrons to jump from the valance band to the conduction band. The energy can come from heat, radiation, electric or magnetic.



	ΔPE
Conductors	↓
Semi-Conductors	med
Insulators	↑

$E_T = E_p + E_k \rightarrow = \frac{1}{2}mv^2 = \frac{1}{2}\frac{|P|^2}{m}$

Lecture 4: Transistors

BJTs: (B,E,C), bipolar since operation involves holds e^- & p^+
EB is forward biased & BC is reverse biased. Problem w/ heat dissipation & power loss.

MOSFET: Electronically varying the width of a conduction channel. Charge enter via Source & leaves via drain. Voltage controlled by the gate. Advantages: gate insulated, so no current from gate to chanel. Problems: SiO_2 so thin that susceptible to permanent damage from electric charges & does not perform well at weak RF signals.

Functions in
- depletion mode: As voltage applied, conduction ↓
- Enhancement mode: Voltage applied = conduction ↑

Power loss $\propto I^2$

	BJT	MOSFET	
Power (Heat)	more	less	
Switch speed	Slow	Fast	Verr ∝ Power loss ∝ $\frac{dI}{dt}$
Gain speed	Fast	Slow	Fast switching speed = $\frac{dI}{dt}$ ↑ (BJT faster)
Current	mA	μA	
Susceptible to Noise	Less	more	$V_{EMF} = I_{ind}R + L\frac{dI}{dt}$ (∝ Heat)

EMF
por

CMOS: High speed, low noise, low power
No net current traveling across device ⇒ only power for switching = less heat

Memory cells
MRAM (Magnetoresistive): depends on SPINTRONICS (study of electron spin polarity) and uses spin transistors.
Fast speed, high density, low power, non volatile, radiation hardness.

NRAM (Nanotubes): Can be insulator, semic or conductors depending on Carbon gyration. Maintains charge w/o power
Advantage: High density & speed.

FeRAM (Ferroelectric) 2MOSFET + 2Ferroelectric capacitors
Polarization maintained w/o power. R/W faster than flash
(DIFF vs flash used bombardement of electrons)

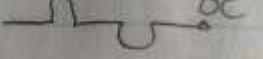
Architectural views
- Component view: large scale peices & how they fit together
- Distribution view: Concurency & comunication protocols
- Resource view: Resource management for concurent applications
- Reliability view: Redundancies & safety protocols, fault tolerance & recovery
- Deployment view: SW maping on HW, memory maping, task distributions, data flow.

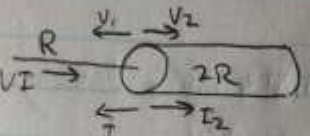
LECTURE 5

	I/sec =	Clk	× Ins Eff.
DSP	10GI/s =	10GHz	100%
CISC	$\frac{1}{5}$GI/s =	1GHz	50%
RISC	2GI/s. =	1GHz	200%

EJB
P03

Boundary Conditions

Reflection  Short 



$V = V_1 + V_2$

$IR = (I_1 R) + (I_2 2R)$

$I = I_1 + 2I_2$

Also, $I = I_2 - I_1$

$I_2 - I_1 = I_1 + 2I_2$

$\boxed{-2I_1 = I_2}$

Real time monitoring

- SW monitor to breakpoint
  - pros: No extra HW support needed
  - Cons: Requires target mem & CPU time
- On chip background debugger program
  - pros: Reduces CPU overhead for RT applications
  - Cons: Eats up target resources
- Incircuit Emulator (ICE) ex. VxWorks
  - Pros: No overhead on target, non disruptive
  - Cons: Expensive, hard to simulate RT, unreliable socket contacts.

Addressing

Direct: IR 2 bytes (opcode + Addy). More space, but more flexibility

Inherent: IR 1 byte (op code), CPU already knows target Data Reg. More efficient, faster, less flexible.

Lecture 6

System Types

- Distributed: Slow, large
- Mission Critical: Reliable, $
- Signal processing: Fast

ECE
P04

Project lifecycle phases.

1. Conception stage: goals, requirements for interface, mem, interrupts, systems, CPU & Mem throughput & bottlenecks.
   Output: MOU (Memorandom of understanding).

2. Requirements phase: Customer specific goals, technical requirements,
   Output: SRS.

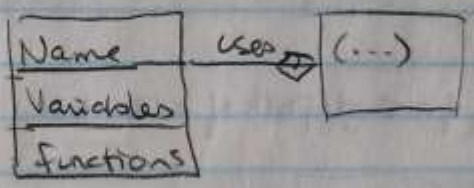
3. Design: build drivers & subsystems, ISRs, use simulation like FPGAs & VHDL
   Output: SDS (Charts & diagrams)

4. Implementation. Code, build & test
   Output: HW devices, SW Drivers & Modules

5. Integration: unit test every modules, put them together and system test
   Output: System test cases document (tests that proves that every requirements are met).

6. Release & Maintenance: Fix bugs, upkeep & upgrades.

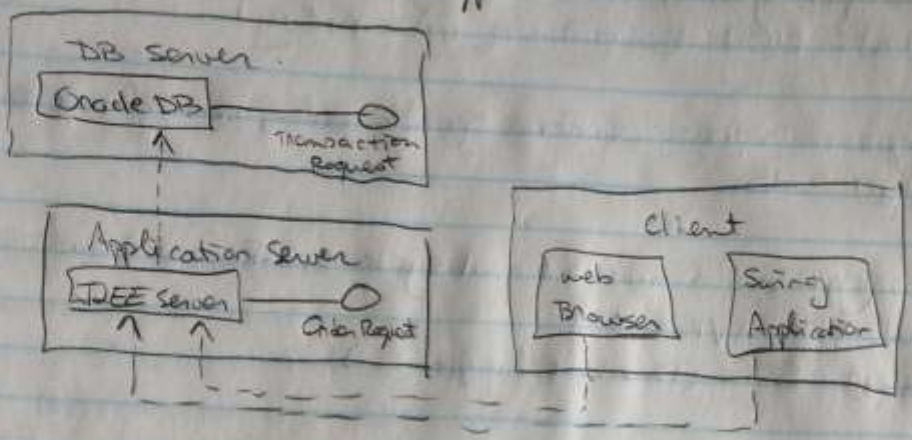
Class diagram



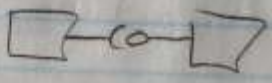
EHB

P05.

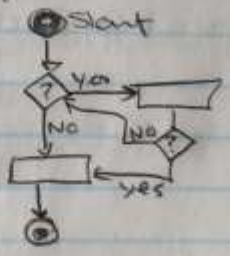
Deployment diagram: models the physical aspect of Object Oriented SW System. Models a static view of Runtime configuration distribution of components & applications.



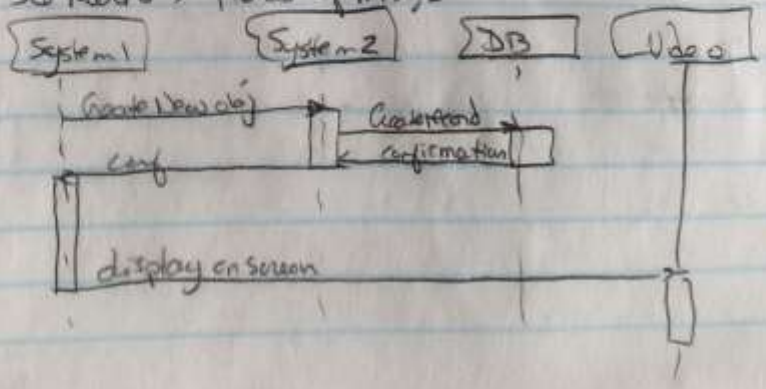
Component Diagram: Components wired together. Illustrates the service provider & consumer relationships.



Activity diagram: Detailed behavior inside a functional requirement



Msg sequence diagram: behavior of a single path of a use case Scenario → flow of msgs.



EMB
p06.

Memory Architecture

Harvard: Seperat instruction & Data mems & bus.

VonNeumann: Shared bus, interlaced Data + I in 1 mem

Memory types

- Static Rams (SRAM): Flipflops, fast & expensive (used for $)
- Dynamic Rams (DRAM): Capacitive devices, slow, scalable, cheap, regs accessed by addresses
- Synchronous SDRAMS: Clocked by system to allow CPU do something else while DMA.
- Flash: DRAM w/ much more charge, non volatile, slow write

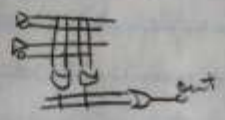
Cache: biggest source of jitter.

Av. Accesstime = $(1-TR)(\$\text{speed}) + (TR)(\$ + \text{Mem speed})$ Min.

$\varnothing(\$\text{speed}) + 100\%(\$ + \text{Mem speed})$ Max

Lecture 7

PLD:

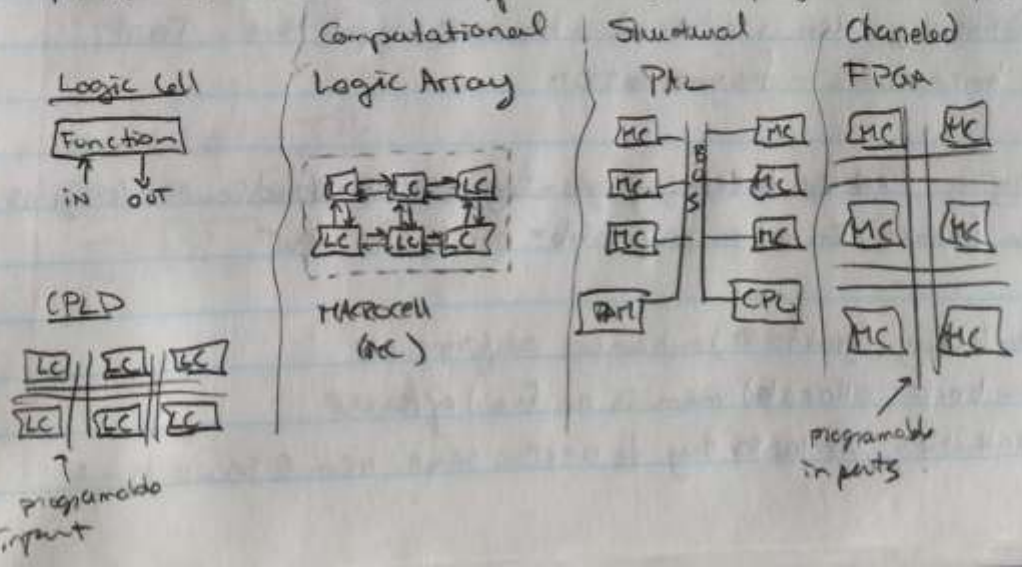


PAL: PLD w/ loopback from OR to ANDS.

PLA: PAL w/ selective loopback, more restrictive but fast, building block for CPLD

CPLD: More rigid, faster

FPGA: CPLD + CPU + RAM = Self contained, more programable, 3 kinds (see below)



EMB

P08

LECTURE 8: VHDL

```
ENTITY abc IS
    PORT(
        a,b : in STD_logic;
        c : out STD_logic);
END abc

ARCHITECTURE guts of abc IS
Component ... is
    port(...)
Signal ...
BEGIN
```

Internal Delay Model

```
c <= R NOR Q AFTER 1NS;
c <= S NOR Q AFTER 1NS;
```

only happens if hold time is respected

Transport Model

```
c <= TRANSPORT R NOR Q AFTER 1NS
c <= TRANSPORT S NOR Q AFTER 1NS
```

happens no matter what (detects rising edge)

LECTURE 9: MEMORY MGMENT

- Swapping: Program size too large. When changing contex, put process into secondary storage device. (slow).
- Overlaying: virtual memory. (slow & memory corruption)
- Partitioning: Mem is partitioned in blocks @ compile time. Fast, but results in fragmentation
- Paging: Fixed size blocks of program segment are stored in non-contiguous mem frames. Involves page table = overhead = slow
- Mem locking. (good for RT) enhances performance.
- Mem leak: allocated mem is no freed after use
- Contention: 2+ tasks try to access same mem @ same time

EMB
POS

	Stack	Heap
Allocation	Compile time	Runtime
Queue	LIFO	Garbage dump
Organization	Daisy Chained	Allocated on the fly so all over

Queuing -

- FIFO: loss free, sender & receiver isolated, no data loss
  ⇒ Mission critical: No corruption, safer
- Ring buffer - Circular FIFO, fast but can lead to corruption
  ⇒ Signal processing: FAST
- Swing buffer: Series of ring buffers in parallel, R/W sequentially
  ↳ FIFO of rings or 2+ ring buffers, last prone & non blocking
  ⇒ Distributed - bigger mem space

Lecture 10: Memory Communication

- Pipes: 1 to 1 between parent & child processes. P is frozen until pipe is cleared
- Semaphores: integer w/ 2 functions, V(s) Verhogen = increment, P(s) = prüfen = test.

Queue	Mailbox
faster	Slower
RW @ Same place	Copy msg
Less mem	more mem
Sync	Async

Socket	Pipe
Async	Sync
1 to many	threaded 1 to 1
inter-platforms	Shared memory (single platform)

Lecture 11: Jitter

Jitter - time between arrival & expected arrival.

$J_\varepsilon = \sqrt{\sum_i j_i}$

EYB

p10.

Measuring Exec time

polling	Interruption	Login Analyzer
Cheap	Cheap	Expensive
Slow	Faster	Fastest
CPU OH	CPU OH	No OH
Src of Jitter	Src of Jitter	Instant.

Clock Requirements

- Correctness: Clock-RT $\leq \varepsilon$ (is it the real time?)
- Bounded drift: $\frac{d\varepsilon}{dt} < k$ (does it drift over time)
- Monotonicity: Take 2 measures. if $c_1 > c_2$, then $t_1 > t_2$
- Chronoscopicity: Measure something same day wrt same other day

Lecture 9: Smart Cards.

wired	RFID
Cheap	expensive
Short Range	Long Range (flexibility)
less mem	more mem
Secure	Unsecure
Integrity	Error.

Transmission

Async	Sync
Separate or No CLK	Shared CLK
Common	less common
Cheaper	EXP.
Slower (Hand shaking)	Faster
Higer error	less error

EMB

PU

RFID: Smart cards where communication & power are transfered via RF
3 components: Interogator (reader), tag, host computer

Tag: μChip w/ ~2Kbit.
- passive: only power from induction from RF of Reader
- Active: Own power to amplify return signal
- Semi-passive: Power src to turn on, but not to Amp signal.

Tag is composed of an antana coil & Silicon chip (CPU, RAM, EEPROM)

Reader: Control R/W on tag. Generates RF field around antenna directed at the tag ⇒ Carrier signal ~~to~~ Tag gets power & outputs ID ⇒ backscattering signal.

Inductive magnetic coupling @ 125kHz - 13,6MHz, up to 900MHz Higher frequency = better range, metal penetration, faster, more power

Opticalsensor: Temperature sensing of fingerprint
CMOS sensor → Cells contain a photodiode that converts light to electrons by photoelectric effect
$E_e = E_{photon} - [E_{conduction} - E_v]$ Temperature is sensed by the photodiode noticing wavelength of light detected.
XY addressing

CCD: Camera store charge according to the intensity of the image being detected. # of electrons collected proportional to light intensity. Light collected over entire image simultaneously then transfered

Optical	CCD
photoelectric	photoelectric
measures wavelength of charge	measures light intensity of charge produced
cheaper	expensive
XY addressing	FIFO
faster	slower
less noise	less noise
less power	more power
	more jitter because of FIFO.

EEB
P13

L14: Reliability

RT fundamental requirement is to meet its deadlines

Sync

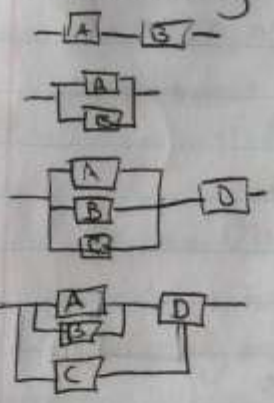
RW blocking until buffer cleared
less OH.
Shared clock
Avoids race condition
less flexible.

Async (preferred for RT)

RW done in parallel
Need A I/O control block
Independant clocks
More computation
Avoid missed deadlines *
More corruption

Layers:
Service provisional point (SPP) ⇒ Drives the SAP
Service Access points (SAP) ⇒ Services the SPP.
↕ layers.

Reliability



$R = R_A * R_B$

$R = R_A + R_B - R_A * R_B$

$R = R_D * [\overbrace{R_A + R_B + R_C}^{1^{st}} - \overbrace{R_A R_B - R_A R_C - R_B R_C}^{2^{nd}} + \overbrace{R_A R_B R_C}^{3^{rd}}]$

$R = [A+B-AB] \cdot D + C \cdot D.$

L15 Attributes of RTS.

- **Reliability**: Must meet its deadlines.
- **Predictability**: Provide range bound and worst case over times. (watchdog, clock)
- **Scheduling**: Requires knowledge of timing constraints ⇒ Makes the system reliable. Must know if deadlines are hard or soft.

Priority ∝ deadline ≠ importance. Dynamic scheduling = OH and more unstable. Unbounded operations are not permitted in RTS. Timeout mechanisms can cause dead locks.

EMB P14

- Concurrency: management of using semaphores & Mutex
  - Synchronisation
  - Communication
  - Resource Sharing

- Visibility. Allows user access to communication drivers and task priority assignments.

⇒ Priority inversion: most likely source of missed deadline
- Priority inheritance protocol: task priority raised to the level of ressource it owns (highest priority of tasks waiting on the ressource) ⇒ Not prevent deadlocks.
- Priority ceiling protocol: tasks suspended except the one owning the highest priority ressource that can run
- Priority disinheritance: shift back to normal

LIG-Kernels

1) OS top layer
   - filesystem/security
   - Task control
   - UI/shell

2) Executive
   - I/O
   - Memory managment

3) Kernel Toplayer
   - Communication drivers

4) micro Kernel
   - Scheduler
   - ISR-Interrupts.

5) Nano Kernel
   - Task dispatching
   - Task bookkeeping

APIs, DLL
FP support

task pointers
Stacks, Cash mgment
Drivers

Network mgment
Task communication tools
Task Scheduling
Clocks, timing, Interrupts

raw task managment

EMP
P15.

L17 Interrupts

External: Timers, devices (Generally HW)
Internal: divide Ø, Scheduler (Generally SW)

Software interrupts.
- Typically sync machine language op. (10, undefined op)
- Defined in CPU HW so OS independant

Hardware interrupts.
- Typically Async external events. (OS & Scheduler have no control)
- Vectored by CPU HW.
- Noise, jitter, ground loops may cause stack overflow.

Interrupt vector: Array of ptrs to ISR. (16 bit / interrupt).
Non-vectored: ctrl is transferred to one single routine that decides how to handel the interrupts.

ISR.
- CPU vector to ISR in HW (indoperdant of OS).
- Re-entrant: can call itself, can be called at the same time by many processes (use semaphores)

Non maskable interrupts (NMI)
- highest priority
- Can't be stopped (avoid if possible)
- Almost all OS SW interrupts are NMI

Simultaneous Interrupts Handling
- Masking prevents other interrupts from being serviced while one is.
- Pre-Empting/nesting: higher priority int. can int. lower priority int.
- Queuing: Serviced sequentially

ETIB p16.

L18 - Interrupt Analysis.

Interrupt sequence
1) Update Status register & make internal copy
2) Determin priority & mask lower priority interrupt lines
3) Ack interrupt priority level
4) Save PC & registers on stack
5) Set PC to head of ISR & service
6) Return from exception & restore task context

Timing interferance
$t(s)$ = time for task to run on itself.
$t'(s)$ = time for task to run in the presence of I
T_i = interrupt execution time
f = frequency of interrupt

$$t'(s) = t(s) + t'(s) \times f \times T_i \quad \Rightarrow \quad t'(s) = \frac{t(s)}{(1 - fT_i)}$$

For many interrupts: $t'(s) = \frac{t(s)}{1 - \sum_i f_i T_i}$

Ex1| Int takes 1% of CPU time ⇒ find fT_i & $t'(s)$.

$t'(s) = t(s) + t'(s) fT_i$
$1{,}01\,t(s) = t(s) + 1{,}01\,t(s) fT_i$
$\boxed{0{,}01 = fT_i}$

$t'(s) = \frac{t(s)}{1 - fT_i} = \frac{t(s)}{0{,}99} = 1{,}01\,t(s)$

Ex2| What is $t'(s)$ in terms of $t(s)$ if I_1 is 2% and I_2 is 4%

$t'(s) = t(s) + \sum t'f_i T_i$

Nesting or Queing

$t' = \frac{t^*}{1 - \sum f_i T_i}$

Masking

$t' = \frac{t}{1 - \sum f_i T_i + \underbrace{\sum f_i T_i f_k T_k}_{i \neq k}} = \frac{t}{1 - 0{,}02 - 0{,}04 + (0{,}02 \times 0{,}04) + (0{,}04 \times 0{,}02)}$

EMB
P17

- Exception Handeling: ISR of error condition
- Fault latency: time between fault occurence & error occurence
- Error latency: time btw error & its effect on system
- IL (interrupt latency): time btw hw interrupt & start of ISR
- CSL (Cortex Switch latency): time between last instruction of T1 and first inst. of T2.
- DL (Dispatcher Latency): time between end of ISR & restart of original task (kernel reschedeling, int enable & context switche)
- WISRT (worst case ISR execution handeling time): max time to execute ISR
- WIR (worst case interrupt response time): IL + WISRT
- WITR (worst case interrupt task response time): WIR + DL = IL + WISRT + DL
- TRR (Task Response Time): task time from start to finish

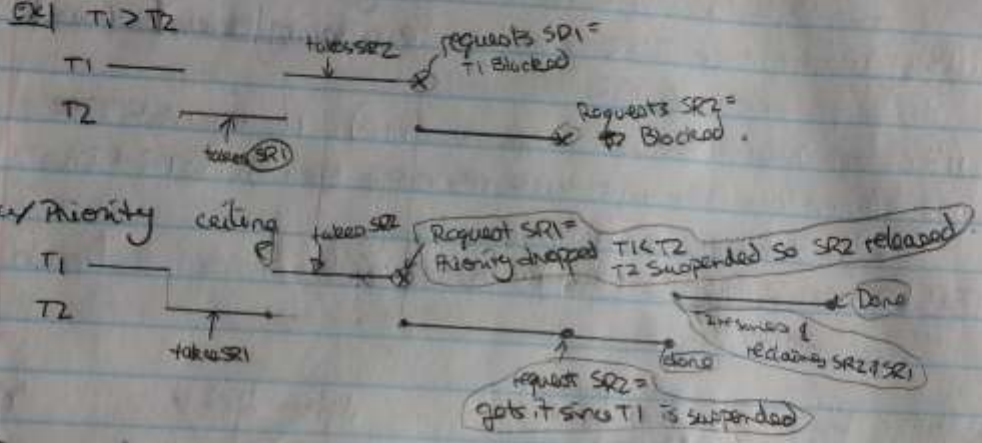
L19 - Priority.

- Pre Emption: higher priority T stealing CPU
- Priority inversion: Low priority blocking high from CPU because of a ressource it owns.
- Unbounded priority inversion: priority inversion, but low is blocked by higher priority.

P = priority
T = Task

Priority Inheritance Protocol: temporarily raise low P.t. to the level of a higher P.t. waiting on a resource the low P.t currently owns.

Priority ceiling Protocol:
1) Assign priority to SR = priority of Task that locks it
2) Suspend tasks requesting a resource except T that owns the resource with the highest p.



Period transformation:
- Fixed rate monotonic scheduling ⇒ task assigned a fixed p. depending on its paid higher frequency (f↑) ⇒ higher priority (P↑)

Round Robin: Every one gets a turn

Pre Emptive

Generic Fixed: P. automatically assigned & fixed.

Rate Monotonic (RT)
- Priority assigned by period.
- period = deadline (ignore the deadline)
- Most CPU efficient
- Total utilization $\mu_T = \Sigma e/p$.

EMB p19.

$\frac{n(2^{1/n}-1)}{}$

↑ will work ↑ may work ↑ will not work

Deadline Monotonic (DM)

- Shorter deadline has higher priority.
- Dead line utilization, $\mu_D = \sum e/d \leq 1$ will work
  $\mu_D > 1$ may work.

Least Compution Time (LCT)

- Shortest e. has higher priority

Fixed Utilization

- $\mu = e/p$ Greatest μ has highest priority

Earliest deadline first (EDF)

- At any time, T w closest deadline has highest P.
- $(d-t)_{min}$
- Optimal algorithm for uniprocessor (CPU μ_T is maxed)
- All tasks schedulable if $\Sigma e/p \leq 1$

Shortest Completion Time (SCT)

- Least exec time remaining has highest p.
- $e_{left} = (e_{tot} - e_{done})_{min}$

Least Slack Time (LST)

- Task has the least amount of time to meet deadline
- $[(d-t) - (e_{tot} - e_{left})]_{min}$

EHB
p20