Chapter 1: Introduction

Resource Allocator: manages and allocate resources
Control Program: controls the execution of user programs and operations of I/O devices
Kernel: the one program always running (all else being application programs)
RTOS: OS designed to deal with real-time issues where 1. applications are time-critical, so schedulers guarantee upper-bound of execution time (deadline), 2. strong emphasis on reliability and 3. Emphasis on reading sensors and operating actuators
Response Time: Time between input to a system and realization of required behaviour (output)
Real-Time System: Computer system that must satisfy bounded response time constraints or risk severe consequences OR system whose logical correctness is based on both correctness of output and their timeliness
Periodic Events: Events that occur at predictable times
Aperiodic Events: Events that do not occur at regular periods

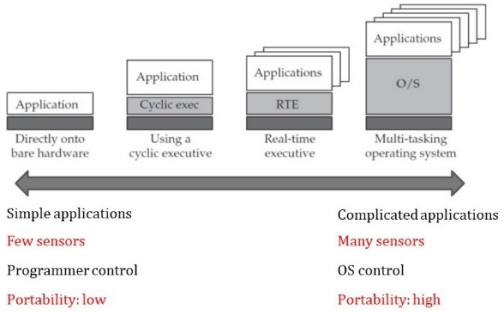
Desirable Characteristics of RT System
Timeliness, Predictability, Survival Under Peak Load, Fault Tolerance, Maintainability

Types of RT System: Tolerance of Missing Deadline
Hard RTS: Failure to meet a single deadline may lead to complete or catastrophic system failure
Soft RTS: Failure to meet deadlines may degrade performance but not destroy the system
Firm RTS: Failure to meet a few deadlines will not lead to total failure, but missing more than a few may lead to complete or catastrophic system failure

Types of RT System: Multitasking Support
Single Tasking System: Run another task only when current task is completed; simple memory layout
Multitasking System: Several tasks are kept in the main memory simultaneously; CPU multiplexed among them
Multitasking on Multiprocessor: Tasks run in parallel on different processor

Advantage of Multitasking:
1.System able to handle several unpredictable input/output activities with variable demands
2.Decomposing problems into multiple semi-independent tasks for concurrent processing offers easier methods for dealing with complexity
3.Improves system throughput/bandwidth & efficiency

Types of RT System: System Architecture



Chapter 2: Interrupts and DMA

Trap: Software-generated interrupt caused either by an error or a user request.
Polling: Synchronization mechanism between processor and device where processor repeatedly reads device status register that indicates whether an expected event has taken place.
Synchronous: Exceptions raised by internal events
Asynchronous: Exceptions raised by external events not related to execution of processor instructions
Programmable Interrupt Controller: Prioritize multiple interrupt sources such that the higher priority interrupt is presented to the processor and take care of processing required to determine interrupt's source
Nested Interrupts: Ability of higher priority interrupt source to pre-empt the processing of lower priority one
Interrupt Latency: Interval between the time when interrupt is raised and the time when ISR execute

Disadvantages of Polling:
1.Processor cannot perform any useful operation as it is always busy with polling
2.Energy wastage as processor cannot go into sleep mode at all

Interrupt:
1.Each device controller in-charged of a device type has a local buffer.
2.Device controller moves data to/from local buffer from/to CPU memory (cache)
3.Device controller informs CPU that it has finished its operations by causing an interrupt
Non-Overlap: I/O devices and CPU execute individually
Overlap: I/O devices and CPU execute concurrently

Common Functions of Interrupts
1.When an I/O completes, device raises interrupt request to processor
2.Interrupt transfers control to interrupt service routine through interrupt vector which contains addresses of all the service routines
Interrupt Process:
Interrupt raise -> Processor gets interrupt number -> Use interrupt number to calculate ISR address from interrupt vector -> execute ISR
Note: OS is interrupt driven i.e. OS idles when no interrupt

Interrupt Handling
1.OS store registers and program counter to preserve the current state of CPU
2.OS determines which type of interrupt has occurred
3.Separate segments of code determine what action should be taken for each type of interrupt
4.OS restore processor context after ISR completes and continue execution of other tasks

Interrupt Response Time
- Interrupt response time (T_D) = interrupt latency + processing time
- Real-time requirement: T_D < Time between interrupt
- Reduce latency by keeping ISR code short
- Processing time of higher priority interrupt also contributes to latency of lower priority interrupts

Note: Most architecture treats interrupts as higher priority than running task.

Split Interrupt Processing
- Perform minimum amount of work within ISR and let dedicated task complete main processing
- ISR informs scheduler to invoke dedicated task
- Dedicated task's execution depends on the priority of other tasks in the system

Direct Memory Access
- Device controller transfers blocks of data from buffer storage to main memory (RAM) without CPU
- Only one interrupt is generated per block rather than per byte (as in the case of interrupt)

DMA Process:
Process initiate READ -> Process block -> OS transfers application information to DMA controller to start data transfer -> OS switches to another process while DMA controller transfer data -> DMA interrupts CPU once transfer complete -> Blocked application now ready

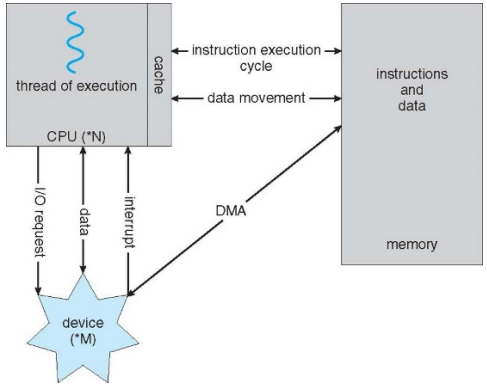
Chapter 3: Real-Time Software Architecture

Simple Polled Loop:
- Repetitive testing of a flag that indicate if event has occurred
- Polling continues until the event has occurred
- Delay can be added between initial triggering of the event and event processing to deal contact bounce
- Pros: Fast response to single device and when no overlapping can occur
- Cons: Fail when event burst occurs (number of events > time to process the events), Waste CPU time

Round Robin:
- Set of self-contained tasks in a continuous loop
- Cycle rate is the same for each task
- Different cycle rate can be achieved by repeating task
- Pros: Fast response to multiple short tasks
- Cons: Fails for device that requires attention in less time than it takes for CPU to go around the loop; Fragile as adding device may violate deadline; no priority as all tasks are processed equally

Round Robin with Interrupt:
- Add interrupt to the polling loop
- Separate each task into ISR for time-critical processing and remaining task code for longer-running code
- Pros: Devices get attended to immediately; put more time-critical processing in the ISRs
- Cons: May still take a while for data to be processed; still does not support priority handling

Function Queue Scheduling:
- Interrupt handler enqueue a pointer to the function that process the data into a priority queue
- Main loop dequeue function from the priority queue and execute the function
- Pros: Easy to enforce priorities
- Cons: Complexity of implementation; Does not support inter-task communication



Chapter 4: Task Management

Process/Task: A program in execution, process execution must progress in sequential fashion
Program: Static entity; executable file
Process: Dynamic entity; notion of program execution
Context Switch: CPU switches to other processes, saving the context of old process and loading the saved context of new process (Note: Context-switch time is overhead as the system does no useful work while switching.)
Multithreading: Multi flows of control/threads within a single process

Advantages of Process Model
- Simplifies design/implementation of concurrent system
- Clean and easy to understand
- Low-level mechanism hidden by RTOS

Process in Memory:
1. Stack: stores parameters and local variables in a function
2. Heap: stores dynamically allocated variable (eg. malloc)
3. Data: stores global parameters
4. Text: Stores code section, program

Process Control Block:
1. Pointer: Locate next PCB in queue
2. Process State: Maintain state of process
3. Process ID: Identifier for each process
4. CPU Registers: Stores temporary data in CPU cache
5. Program Counter: Pointer to the next instruction
6. Memory Management Information: Starting and ending point in the memory structure (memory pointer)
7. Information of Open Files

In multi-programmed system, PCBs are stored in main memory for security.

Process State: Created
- Immediately after creation with valid PCB
- OS has initialized the PCB of the process
- Process can only be created if there exist minimum resources to support it
- Does not compete CPU with other processes for execution yet

Process State: Ready
- Process is willing and competes CPU with other processes to execute
- No processor available to execute it yet
- Admission control of OS enables transition from Created to Ready
- In RTOS, OS must ensure new process actively competing for execution cannot affect timing of whole system

Process State: Running
- Process is being actively executed by a processor
- As processor can only have one running process at any moment, Scheduler of OS which is transparent to the process enables transition from Ready to Running

Running -> Ready State
- Pre-emption (involuntary, any point in code): scheduler forces running process to relinquish processor even if it is still willing to execute
- Yield (voluntary, specific locations in code): process itself asks the OS to reconsider the scheduling decision and possibly hand over the processor

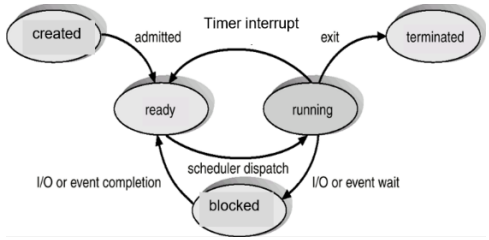
Process State: Block
- Process is waiting for an event, such as completion of I/O operation, synchronization with other process, communication with other process
- Process does not compete for execution in this state

Process State: Block -> Ready State
- Happens when an event occurs
- For I/O event, ISR moves the processes to ready state
- For synchronization/communication, OS moves the process to ready state

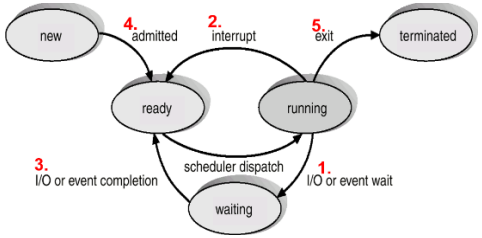
Process State: Terminated
- Process has completed execution and can no longer be executed
- PCB is still available for other processes to retrieve and examine information contained in PCB (eg. retrieval of exception handling)
- Process and PCB are eventually removed from system

Process State: Running -> Terminated
Voluntary: Process completed execution and is no longer needed in the system
Involuntary: Process encounters unrecoverable error

Process State Transition:
Voluntary Action by Process Per Se: Running -> Ready (Yield), Running -> Blocked, Running -> Terminated (Complete)
OS Decision: Created, Created -> Ready, Ready -> Running, Running -> Ready (Pre-emption)
Event Triggered by Hardware or Other Process: Blocked -> Ready. Running -> Terminated (Error)



CPU Scheduler
- OS component that decide which process will be executed and how long the process should execute
- Can have constraints on which processes are available for scheduling at any given time (eg. P uses values of Q, scheduler cannot execute P before Q completes)
- Scheduling decision occurs when process changes state
- Nonpreemptive: CPU has been allocated to a process which keeps the PCU until it releases the CPU by termination or requesting IO/event wait (1, 5)
- Preemptive: CPU can be taken away from running process any time by the OS (2, 3, 4)



Thread
- Thread/Lightweight process is a basic unit of CPU utilization consisting of thread ID, program counter, register set and stack space
- Thread shares with its peer threads in the same process its code section, data section, OS resources
- Threads are like process (Process state vs Thread state, PCB vs TCB, context switch in TCB and PCB)

Chapter 5: Real-Time Scheduling

Worst-Case Execution Time (WCET), C: Maximum amount of time required to complete the execution of any instance of task without any interference from other activities
Deadline, D: Maximum amount of time allowed between a task being released and its completion time.
Worst-Case Response Time (WCRT), R: Maximum elapsed time between the release of a task instance and its completion
Pre-emptive Priority-Driven Scheduling: Lower priority task that is current executing gets context switched out once a higher priority task becomes ready
Hyper-Period: Least Common Multiple of the task periods
Utilization: Fraction of processor time spent in the execution of the task set, $U = \sum_i \frac{c_i}{p_i}$

Basic Real-Time Task Model
1. Tasks are periodic with known periods and:
   a. Periods do not change with time
   b. Task is an infinite sequence of instance
   c. One instance is released at the beginning of each period
2. Tasks are completely independent of each other
3. System overhead for scheduling and context switch time is negligible
4. All tasks have hard deadline
5. Assume D = P, to meet deadline, C ≤ R, R ≤ D

Rate-Monotonic Scheduling for Fixed Priority
1. Priority is inversely proportional to task period
2. Scheduler repeats itself every hyper-period, i.e. task set is schedulable if no deadline is missed in one hyper-period
3. If a task set is schedulable by any arbitrary but fixed-priority assignment, then it is schedulable by RMS
4. U ≤ 1 is a necessary but not sufficient condition for any task set to have a feasible schedule i.e. even if U ≤ 1, task set may not have a feasible schedule
5. Utilization Bound $U \leq n(2^{\frac{1}{n}} - 1)$ is a sufficient but not necessary condition for any task set to have a feasible schedule i.e. even if the condition is not satisfied, task set may have a feasible schedule