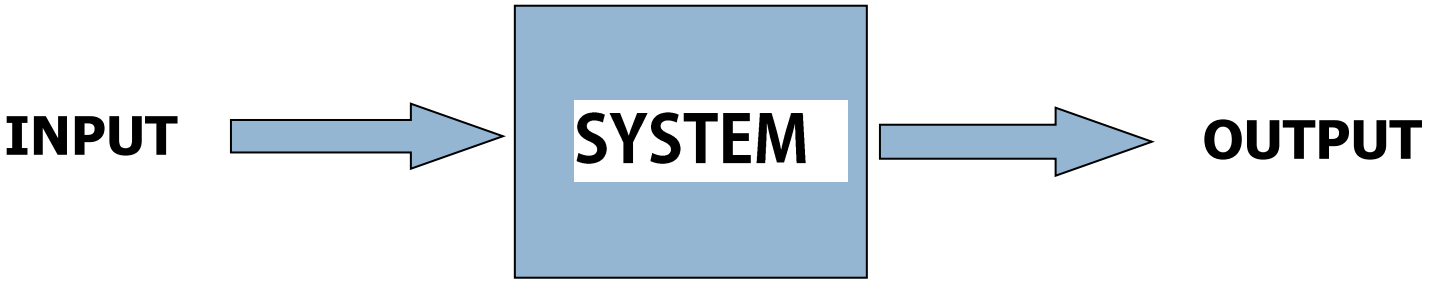
RTOS

An introduction…

What is a SYSTEM?

- An entity which takes an input and processes and gives an output.
- A system is a collection of elements or components that are organized for a common purpose.
- Eg:
  - Operating system
  - Control system
  - Embedded system


INPUT
SYSTEM
OUTPUT

Real Time System…

- Real-time systems are defined as those systems in which the correctness of the system depends not only on the **logical result** of computation, but also on the **time** at which the results are produced.
- The speed at which system functions to produce the correct result should be very high.
- Most importantly, the result should be **predictably consistent throughout the duration** of operation of the system

- A ***real-time system*** is a system in which the time at which the output is produced is significant

Some examples

- Flight control in avionics
- Process control in industrial plants
- Robotics
- Patient monitoring
- Command and control
- Telecommunication systems

Types of Real Time Systems

On basis of Timing constraints…

- **Hard Real Time Systems –** Time is a hard constraint. A late response, even if correct is unacceptable.

- **Soft Real Time Systems –** Must handle events promptly, but the constraints are not as well defined.

- **Firm Real Time Systems –** Late computation is useless, but not cause serious damage.

Hard Real Time Systems

- Failure to meet it is considered a fatal flaw.
- Systems where it is absolutely imperative that responses occur within the required deadline.
- **E.g.:**
  - Airplane Navigation Systems
  - Nuclear Reactor Cooling Systems

Soft Real Time Systems

- Late computation is undesirable, but not catastrophic.
- Systems where deadlines are important but which will still function correctly if deadlines are occasionally missed.
- **E.g.:**
  - Airport reservation systems
  - Banking and ATM transaction systems

Firm Real Time Systems

- Late computation is useless, but not catastrophic.
- Systems which are soft real-time but in which there is no benefit from late delivery of service.
- Used to decision support and value prediction.
- **E.g.:**
  - Weather forecast

RTOS…

- VxWorks - Wind River Systems
- Nucleus Plus - Accelerated Technologies
- QNX Neutrino - QNX
- RTOS32 - On Time Systems
- RT Linux - FSM Labs
- uC/OS - Micrium

RTOS and GPOS - Functional similarities

- Some level of multitasking.
- Software and hardware resource management.
- Provision of underlying OS services to applications.
- Abstracting the hardware from the software application.

RTOS and GPOS - Functional differences

- Better reliability in embedded application contexts.
- Faster performance.
- Reduced memory requirements.
- Scheduling policies tailored for real-time embedded systems.
- Support for diskless embedded systems by allowing executables to boot and run from ROM or RAM .
- Better portability to different hardware platforms.

Features of an RTOS

- Multitasking
- Constant Task switch time
- Low Interrupt Latency
- Task Priority
- Support for Preemptive Scheduling
- Scalability
- Low memory Requirement

Characteristics of an RTOS

- Determinism – predictable output
- Responsiveness – quick response
- Reliability – highly stable
- Fail Soft Operation – prevents total system failure
- Scalability – adaptable to changes

Determinacy

- Knowing how and when a system will respond to events, and knowing that the time is repeatable and guaranteed, is called ***determinacy***.

Throughput vs. Latency

- A hard real time system seeks to reduce the latency (time-gap) between external events and system response.

  E.g. : Time between detection of wind change and action taken to steady the airplane must be as short as possible.

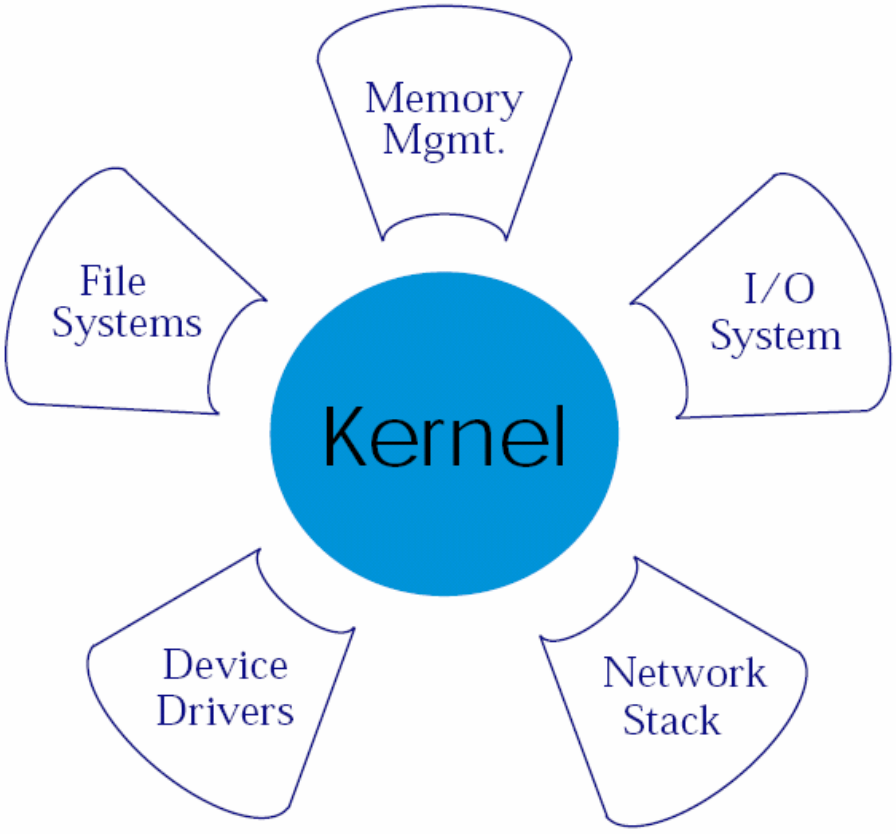
- A non-real time system on the other hand seeks to increase throughput (total information system can handle over a period of time).

  E.g. : A web server must be able to handle a large number of clients over a long period of time.

Components of RTOS

- **Real Time Kernel**
- Auxiliary components:
  - Graphics library
  - Network stack
  - High performance file system
  - Downloading component

Real Time Operating System



Real Time Kernel

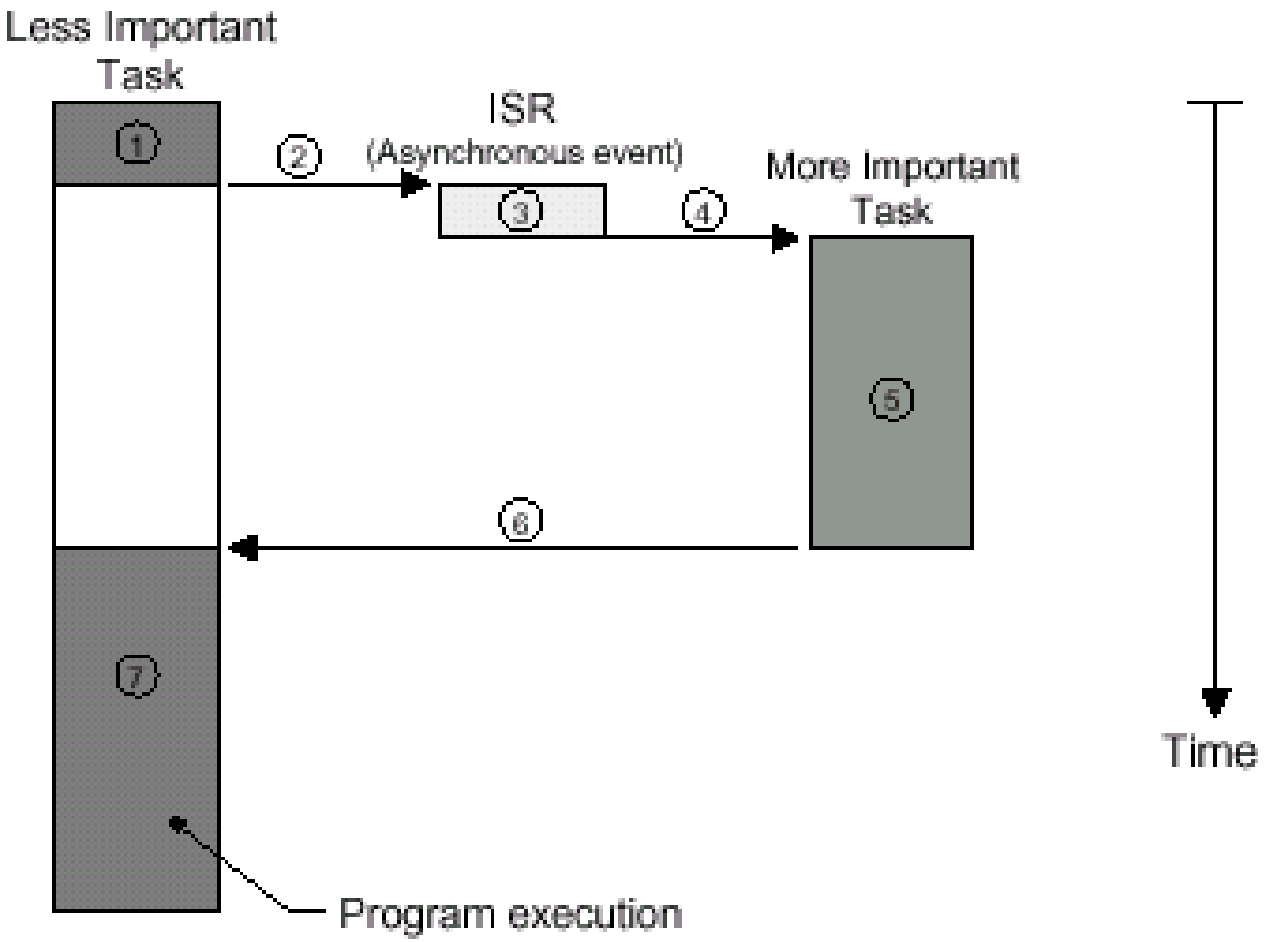
- A **kernel** is a software that manages the time of a microprocessor to ensure that all time critical events are processed as efficiently as possible.

Types of Kernel

Kernels come in two flavors:

- non-preemptive
- preemptive

Preemption


Less Important
Task
ISR
(Asynchronous event)
More Important
Task
1
2
3
4
5
6
7
Time
Program execution

Non-preemptive kernels

Non-preemptive kernels require that each task does something to explicitly give up control of the CPU (i.e. microprocessor).

Asynchronous events are handled by ISRs (Interrupt Service Routines). An ISR can make a higher-priority task ready to run but the ISR always returns to the interrupted task.

The new higher-priority task will gain control of the CPU only when the current task gives up the CPU.

Non-preemptive kernels are seldom used in real-time applications.

Preemptive kernel

A preemptive kernel is used when system responsiveness is important. With a preemptive kernel, the kernel itself ensures that the highest-priority task ready to run is always given control of the CPU.

When a task makes a higher-priority task ready to run, the current task is *preempted* (suspended) and the higher-priority task is immediately given control of the CPU.

If an ISR makes a higher-priority task ready, when the ISR completes, the interrupted task is suspended and the new higher-priority task is resumed.

Kernel Functions

- Multi Tasking and Scheduling
  - Preemptive
  - Non-preemptive
- Inter Process Communication (IPC/ITC)
  - Direct method
  - Indirect method
- Synchronization
  - Semaphores
- Interrupt Handling

uC/OS-II; an introduction

- µC/OS-II is one of the Micrium product
  - (http://www.micrium.com/)
- Highly portable, ROMable, very scalable, preemptive real-time, deterministic, multitasking kernel
- It can manage up to 64 tasks (56 application tasks available)
  - The four highest priority tasks and the four lowest priority tasks are reserved for its own use.
- It supports all type of processors from 8-bit to 64-bit

Interrupt management

- Saves all CPU registers
- Calls kernel ISR entry function: OSIntEnter()
  - Disables Global Interrupt Bit
  - Clears interrupt flags and re-enables interrupts
- Executes user code to service interrupt
- Calls kernel ISR exit function: OSIntExit()
  - Enables Global Interrupt Bit
- Restores all CPU registers

What is a Task?

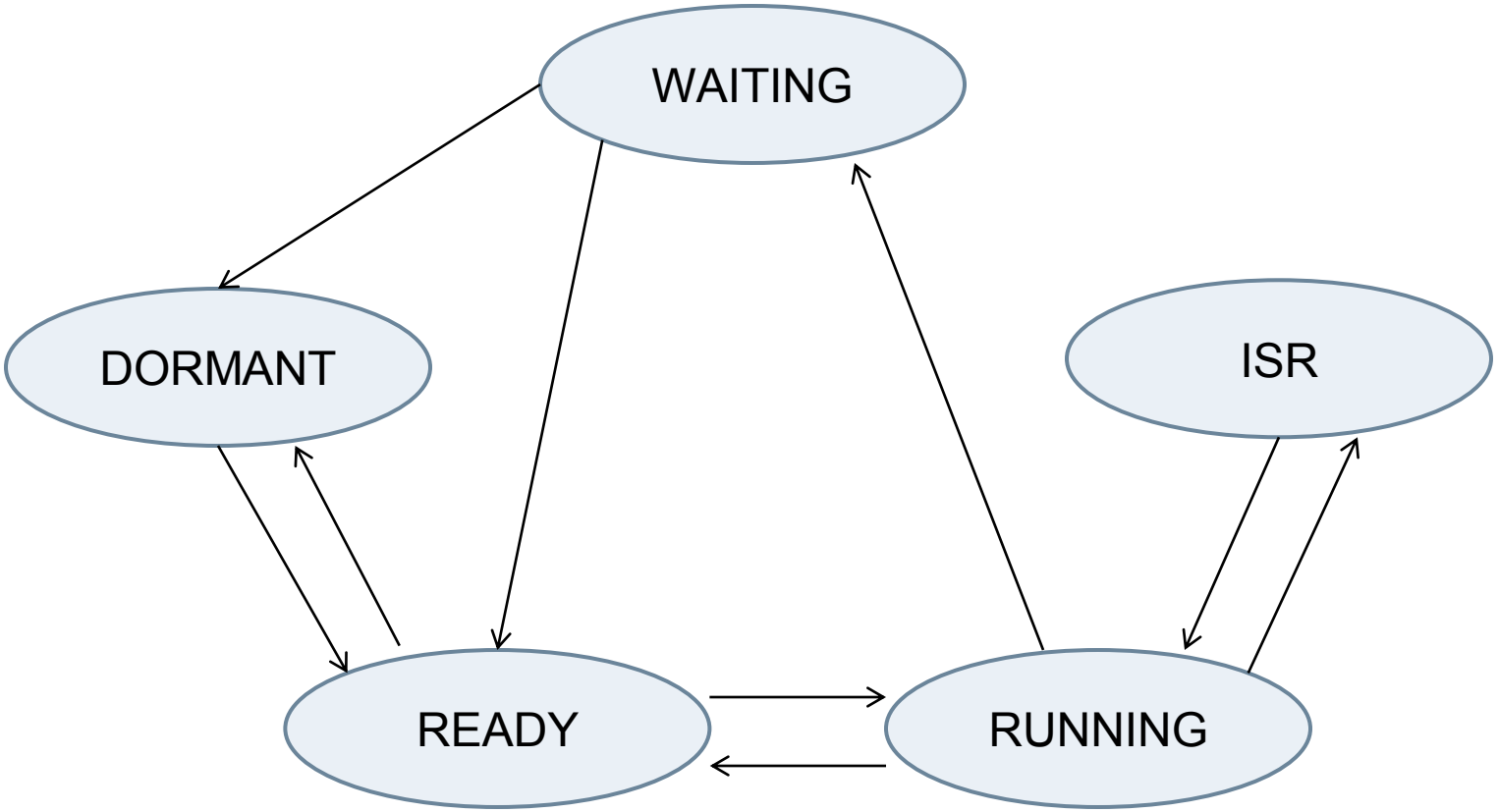
A **task** is a simple program which thinks it has the microprocessor all to itself. Each task is given a priority based on its importance.

The design process for a real-time system consists of splitting the work to be done into tasks which are responsible for a portion of the problem

Task & its data structure

- A uC/OS-II task is built with an infinite loop
  - Similar to C subroutines
  - Return type is always ***void***
  - Composed of a TCB and a STACK
  - TCB contains the task parameters
  - STACK contains:
    - Functions called by the task
    - Local variables
    - CPU registers saved during context switch

Task states

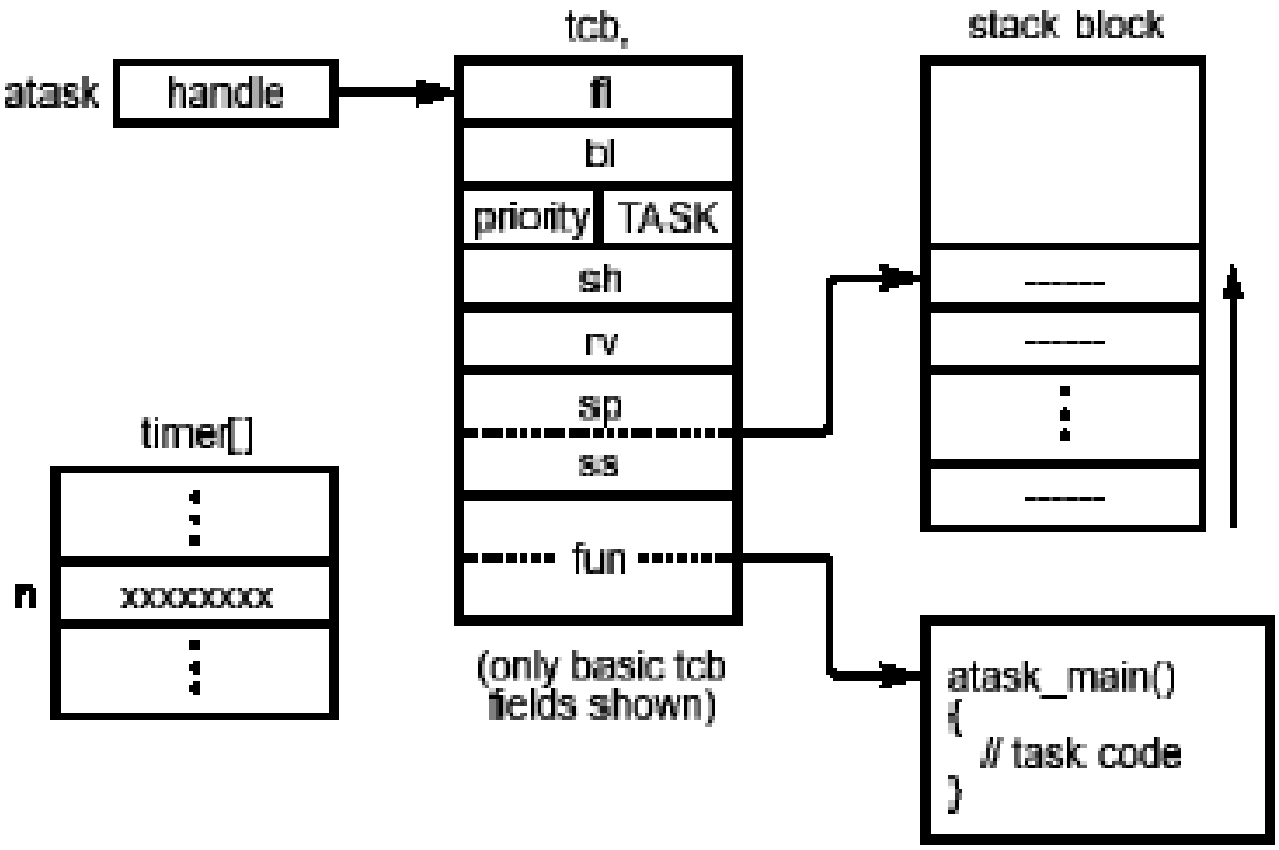


- Whenever a task is created by an RTOS system call, it returns a task handle which is pointer to the Task Control Block.

Task Control Block

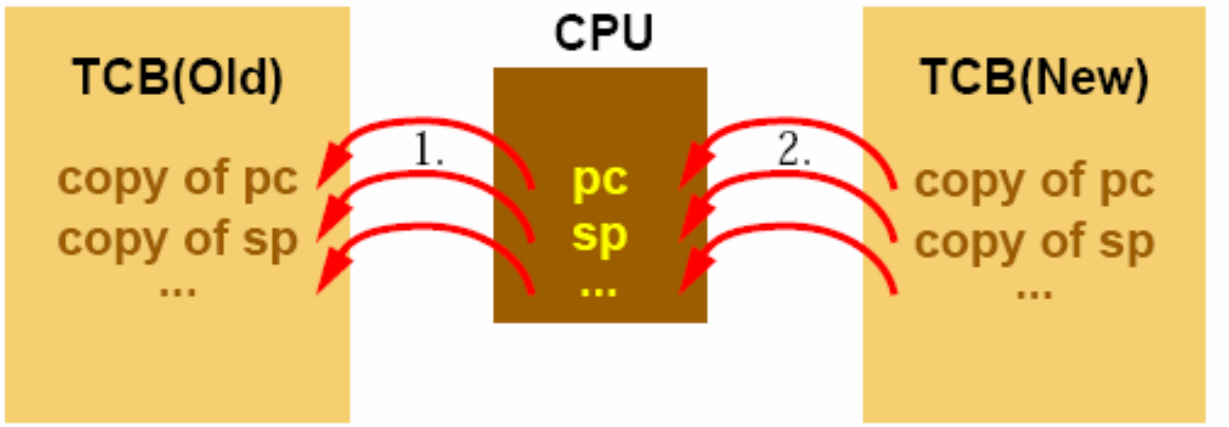
- One per task
- OS control information
  - e.g.: state, task priority, delay timer, break point list, error status etc
- CPU context information
  - e.g.: PC, SP,CPU registers, FPU registers

Task Control Block



Context Switch

- When one task stop executing and new task starts, a ***context switch*** or ***reschedule*** has occurs.



Task management

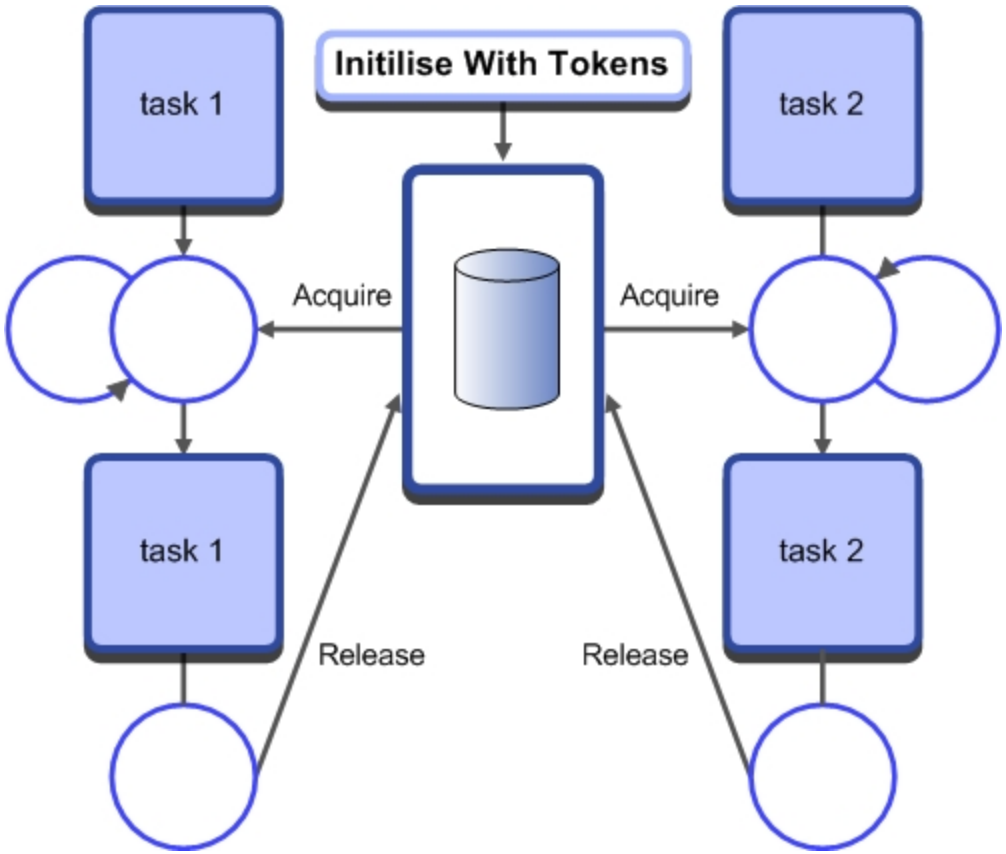
- Creating a task
  - OSTaskCreate(taskname,data,memory,priority);
- Deleting a task
- Suspending/Resuming a task
  - OSTaskSuspend(priority);
  - OSTaskResume(priority);
- Delaying a task
  - OSTimeDly(ticks);
    - Clock Tick: A clock tick is a periodic time source to keep track of time delays and time outs.
  - OSTimeDlyHMSM(hours,minutes,sec,millisec);

Task synchronization

- In a multitasking systems several tasks are executing simultaneously by making use of the shared resources.
- **Semaphores** are used as a synchronization tool to access the critical section in a protected way.
- It has to be acquired before accessing the critical section and release immediately after the job in the critical section is over

Semaphores

- Simply, a semaphore is a container that holds a number of tokens.



Semaphore functions…

- OS_EVENT *OSSemCreate(count);
- OSSemPend(OS_EVENT *, timeout, *err);
  - OS_NO_ERR: if the semaphore is available.
  - OS_TIMEOUT: if the semaphore is not signaled within the specified timeout.
  - OS_ERR_EVENT_TYPE: if pevent is not pointing to a semaphore.
  - OS_ERR_PEND_ISR: if you called this function from an ISR and µC/OS-II has to suspend it. You should not call OSSemPend() from an ISR.
  - OS_ERR_PEVENT_NULL: if pevent is a NULL pointer.
- OSSemPost(OS_EVENT *pevent);

Mutex

- “Mutual Exclusion”
- The main use of a Mutex is to control access to a chip resource such as a peripheral.

Mutex functions…

- OS_EVENT *OSMutexCreate(priority, *err);
- OSMutexPend(OS_EVENT *, timeout, *err);
- OSMutexPost(OS_EVENT *);

Inter task communication

- Direct
  - Pipes
- Indirect
  - Mailboxes
  - Queues

Mailbox

- The mailbox object supports transfer of single variable data such as byte, integer and word width data.

Mailbox functions…

- OS_EVENT *OSMboxCreate(void *msg);
- void *OSMboxPend(OS_EVENT *, timeout, *err);
- OSMboxPost(OS_EVENT *, void *msg);

Queues

- A queue consists of a buffer formatted into a series of mail slots and an array of pointers to each mail slot.

Queue functions…

- OS_EVENT *OSQCreate(void **start, size);
- void *OSQPend(OS_EVENT *, timeout, *err);
- OSQPost(OS_EVENT *, void *msg);