Embedded Systems Architecture

Session #8

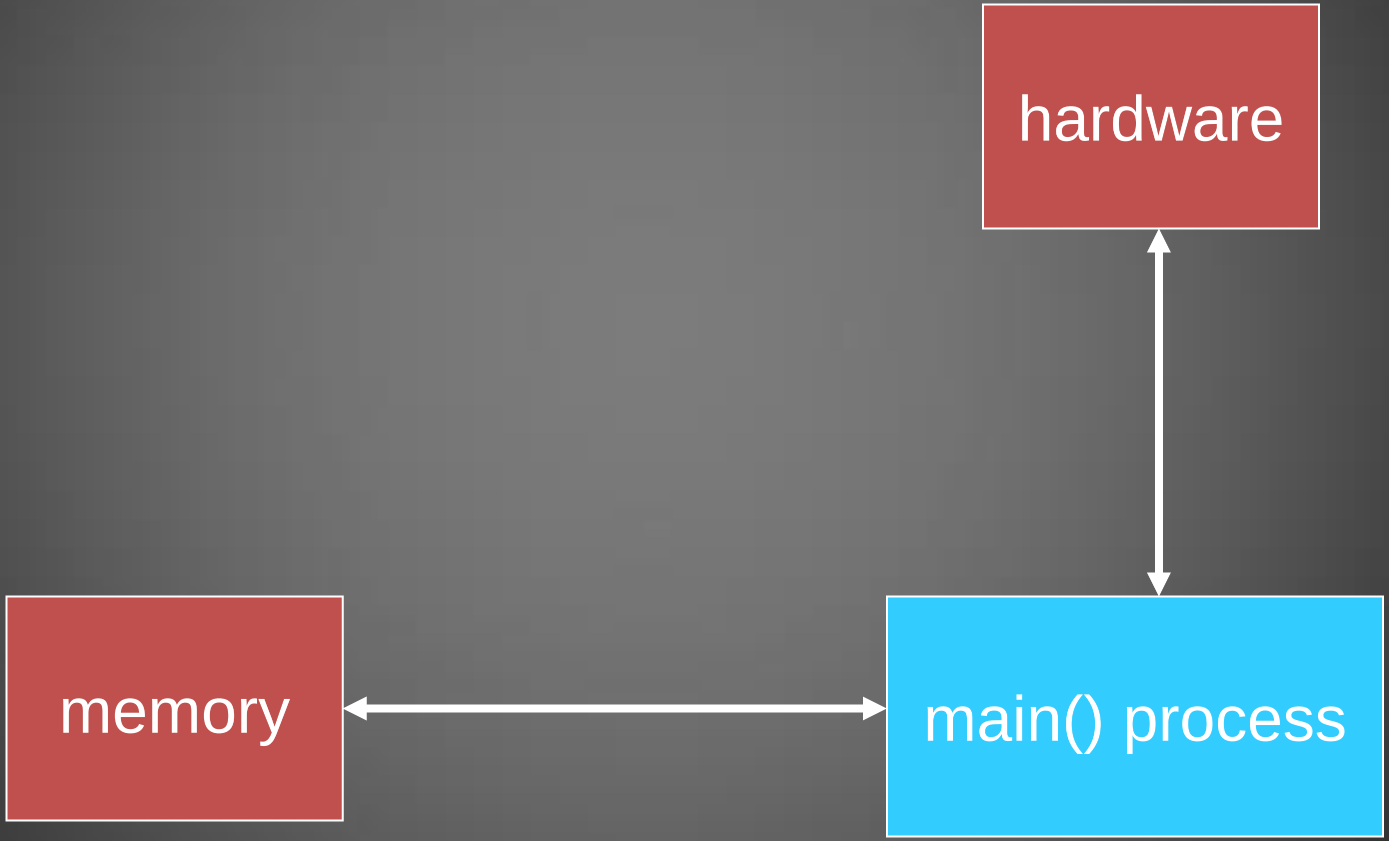
Processes and Real Time Operating Systems (RTOS)

- Why Multi Process?
- Processes
- Context Switching
- The Tick
- Scheduling Policies
- RTOS List

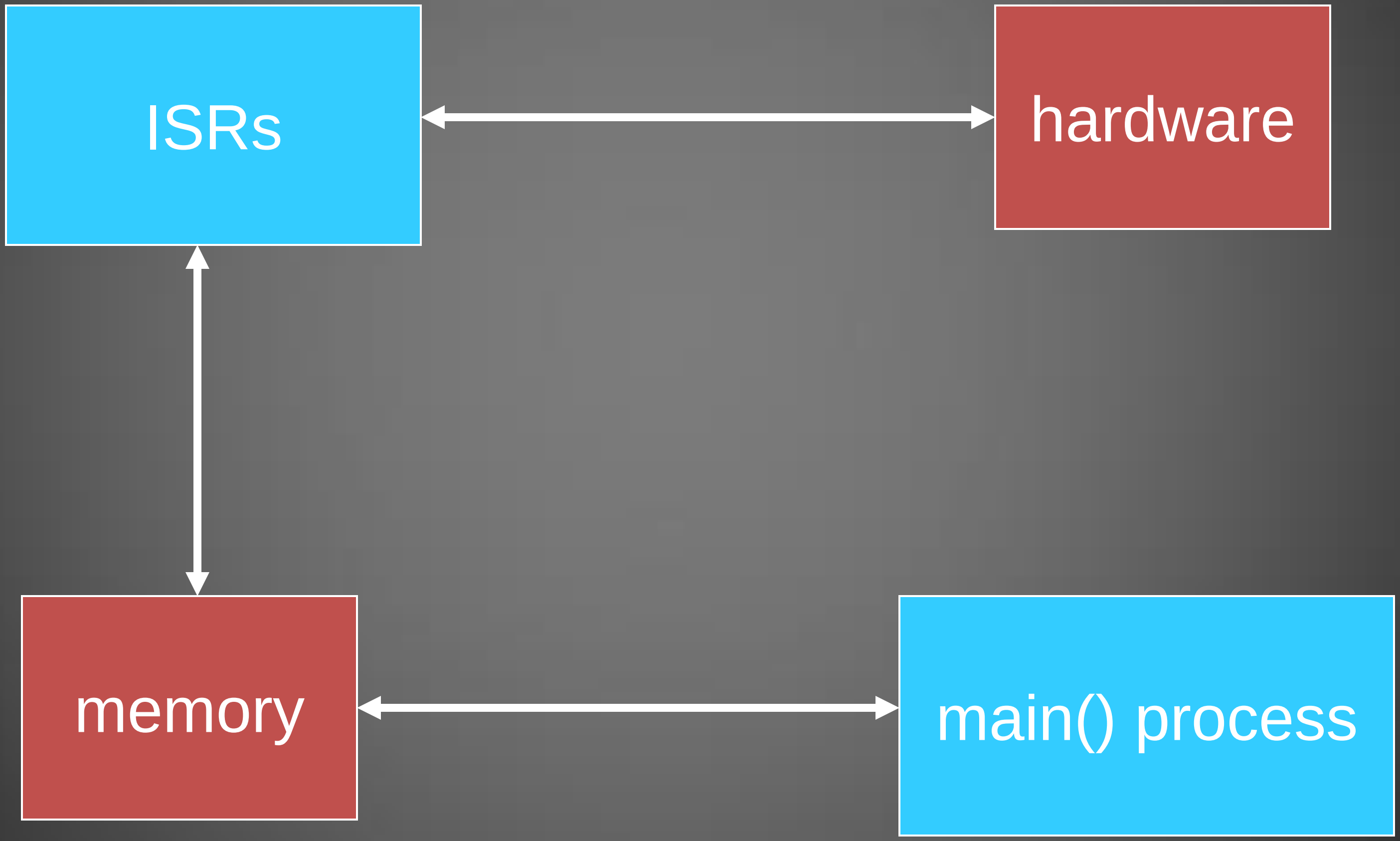
Process / Task / Thread: What's in a name?

- The terms *Process* & *Task* are considered interchangeable.
- Threads compared with processes
  - Processes have separate memory spaces.
  - Threads share the memory space of their parent process.

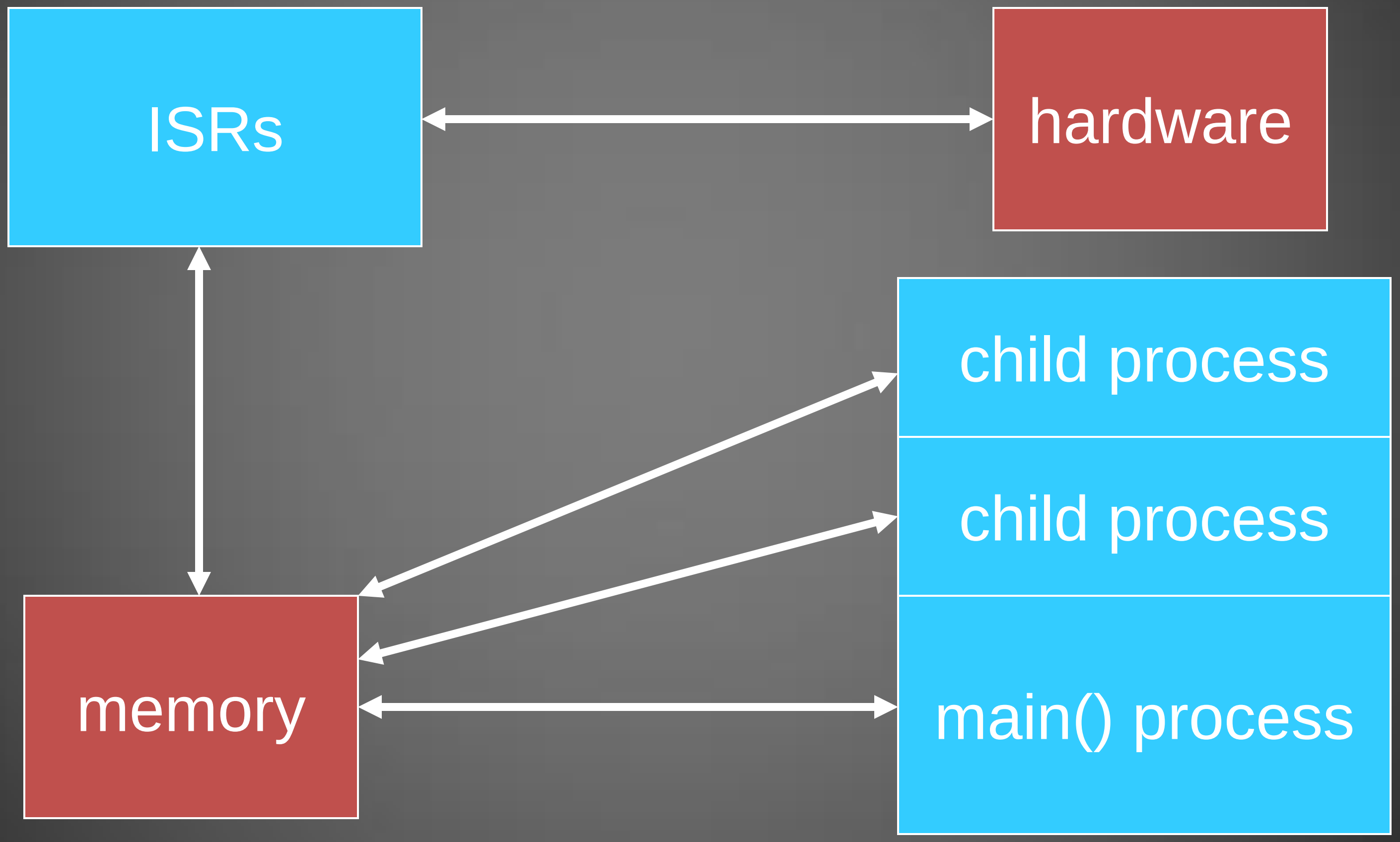
Single Process



Single Process with Interrupts



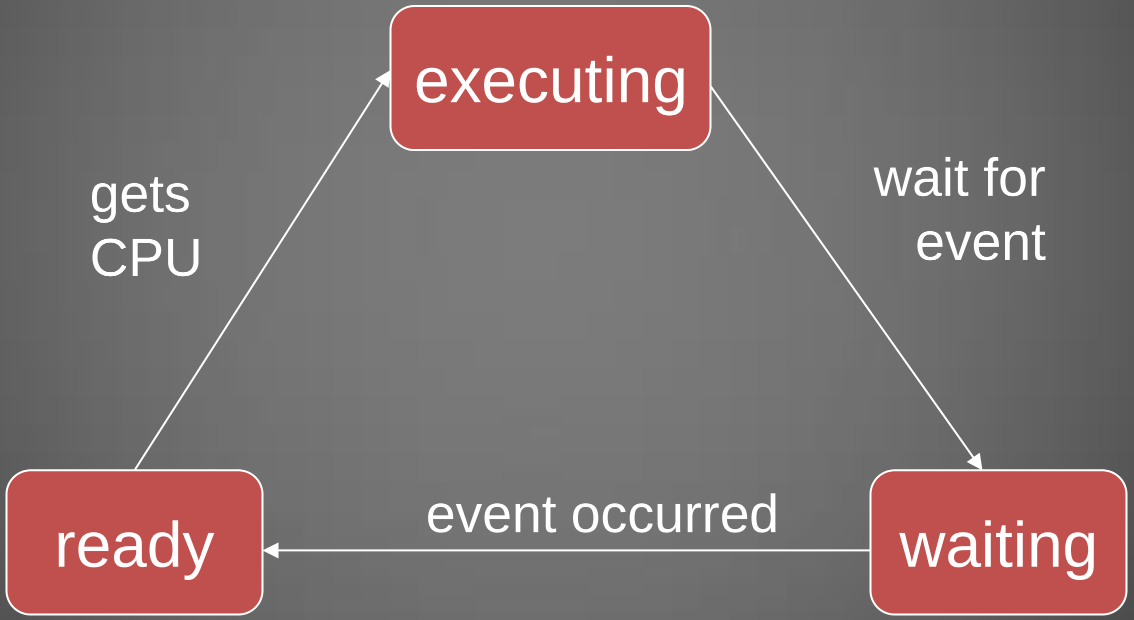
Multiple Processes with Interrupts



Processes

- Process States
- Process Control Block
- Process Priorities
- Process Operations

Process States



Process Control Block

- State
  - *Executing*
  - *Ready*
  - *Waiting*
- Context
  - Registers
  - Resources
- Priority

Process Priorities

- Priority is relative to that of other processes.
- Used to determine which process currently in the *ready* state is dispatched to the executing state.
- Priority can be fixed or dynamic.

Process Operations

- Create
- Delete
- Priority Change
- Suspend
- Resume

Context Switching

- Cooperative
- Preemptive

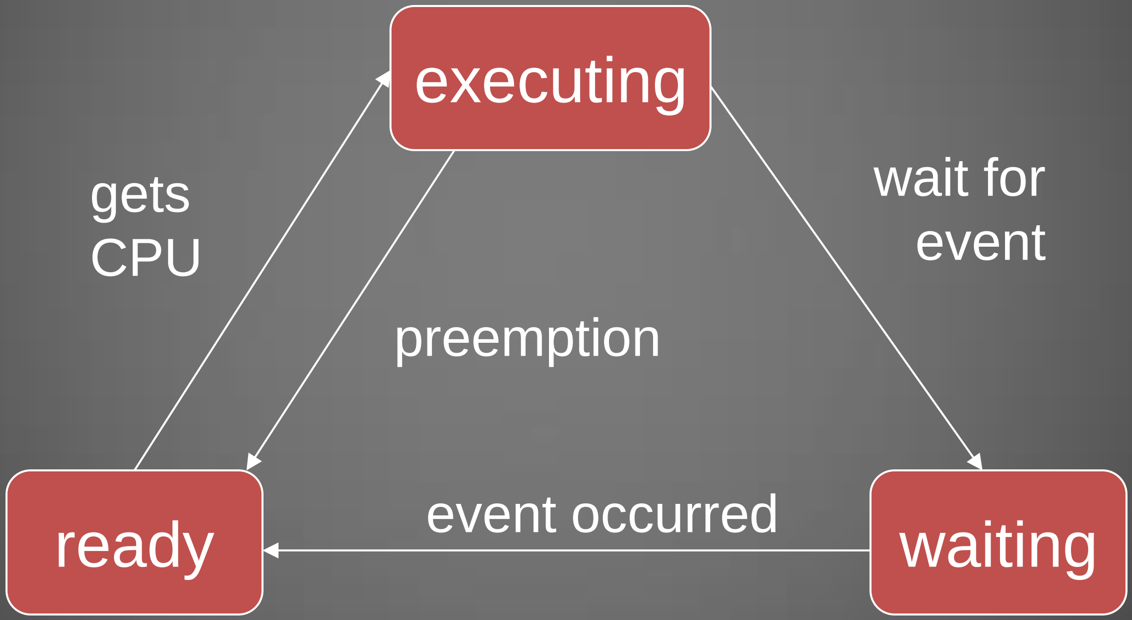
Cooperative

- The executing process *stays* the executing process until it releases the processor.
- If the process *never* releases the processor than other processes can never execute.
- MicroSoft Windows 3.1x used this approach.

Preemptive

- The executing process stays the executing process until it releases the processor *or until it is preempted*.
- Preemption can occur because a higher priority process (or one of equal priority, if we are *round robinning)* is now ready to be executed.
- *But how can we regain control if a process doesn’t release it?*

Process States



The Tick

- A hardware interval timer interrupt that the RTOS uses to facilitate preemption.
- The timer interrupt is also used for timing purposes.
- A tick time interval is sometimes called a quantum or time-slice.

Scheduling Policies

- Fixed Priority
- Dynamic Priority

RTOS List

- Open source
  - eCos
  - FreeRTOS
  - RTLinux
- Proprietary
  - QNX
  - MicroC/OS-II
  - VRTX
  - VxWorks

InterProcess Communications

- Semaphores
- Queues

Semaphores

- Used to guarantee mutual exclusion on a critical section of code.
- Types:
  - Counting Semaphores
  - Binary Semaphores

Counting Semaphores

- Used to allocate shared resources of equivalent type for mutually exclusive use.
  - Semaphore is initialized to the number of shared resources.
  - A *post* (or *give* or *V()*) from a process or ISR will increment the value of the semaphore.
  - A *pend* (or *take* or *signal* or *P()*) from a process will decrement the value of the semaphore, if it is greater than zero.
  - If the value of the semaphore is zero, then the process is placed in the *wait* state.

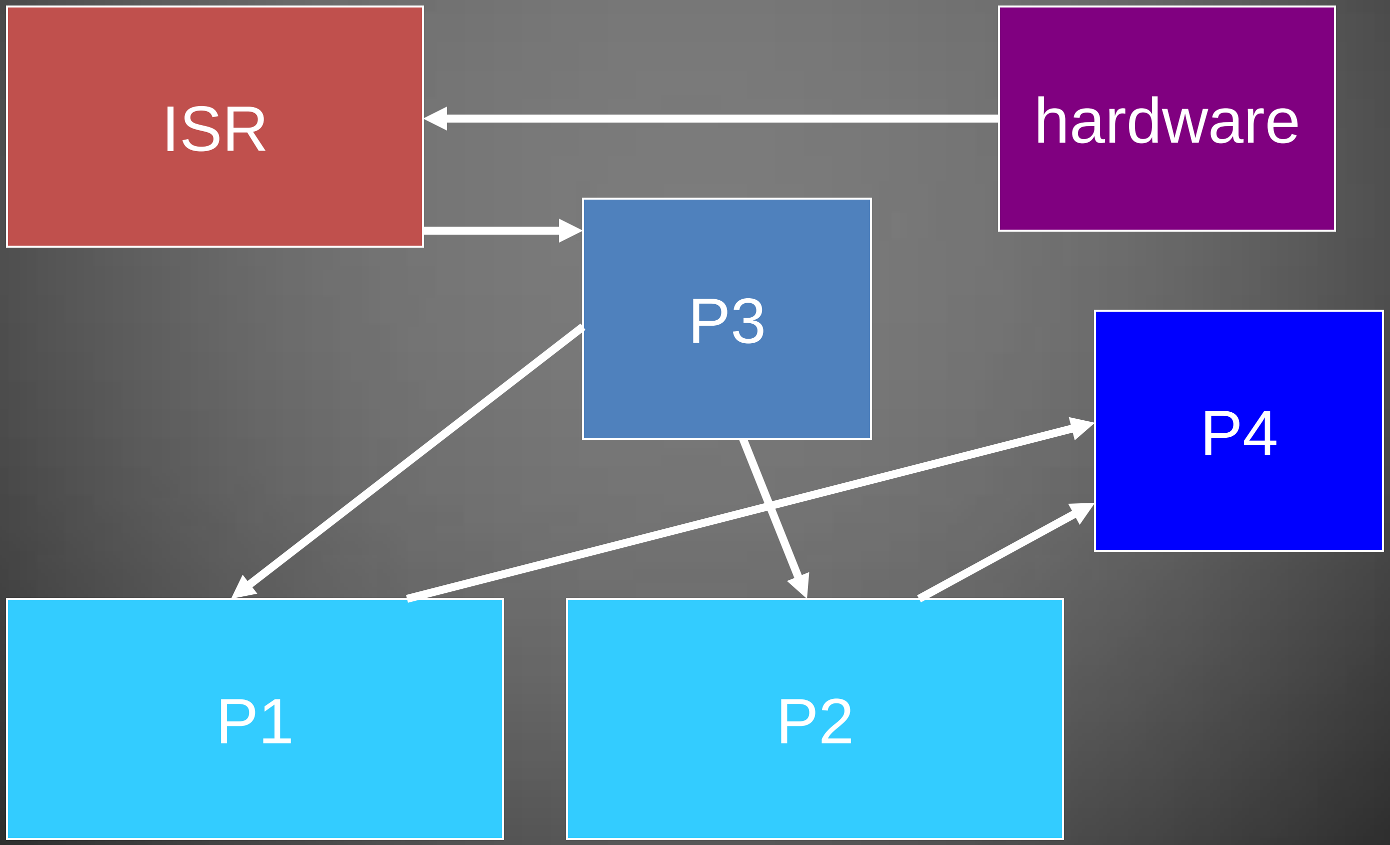
Binary Semaphores

- This is a special case of the counting semaphore where the number of resources is one.
  - Semaphore is initialized to one.
  - A *post* from a process or ISR will set the value of the semaphore to one.
  - A *pend* from a process will set value of the semaphore to zero if it is currently one.
  - If the semaphore is already zero, then the process is placed in the *wait* state.

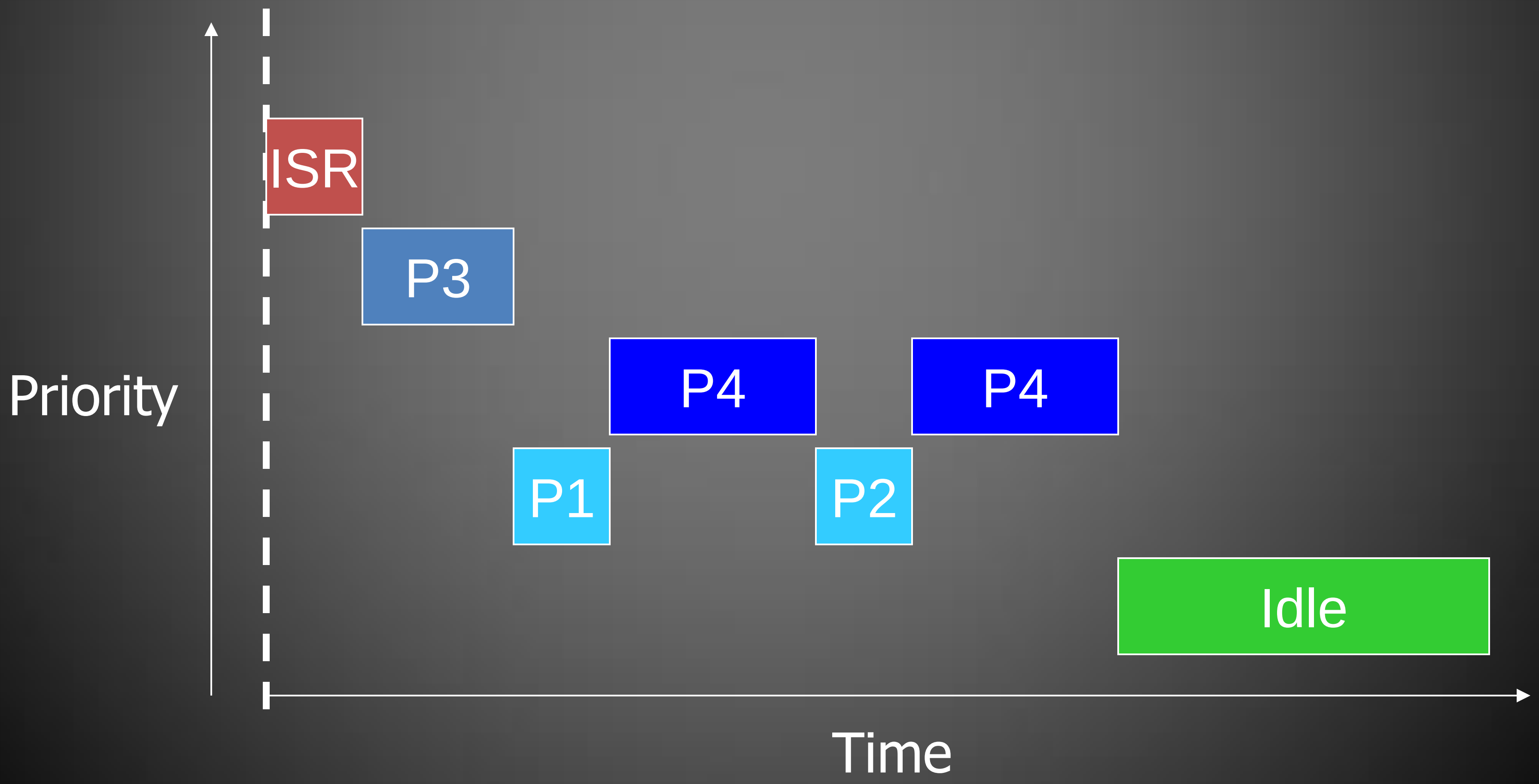
Queues

- A process or ISR can *enqueue* data to a queue.
  - If the queue is full, then the process is placed in the *wait* state.
- A process can *dequeue* data from a queue.
  - If the queue is empty, then the process is placed in the *wait* state.

MultiProcess Example



MultiProcess Example



Mutual Exclusion

- Mutual exclusion (mutex) is the mechanism by which critical sections of code are protected. (I.e.: limited to one process executing it at a time)
- Critical sections of code typically access shared data or hardware resources.
- Binary semaphores can be used to guarantee mutual exclusion.

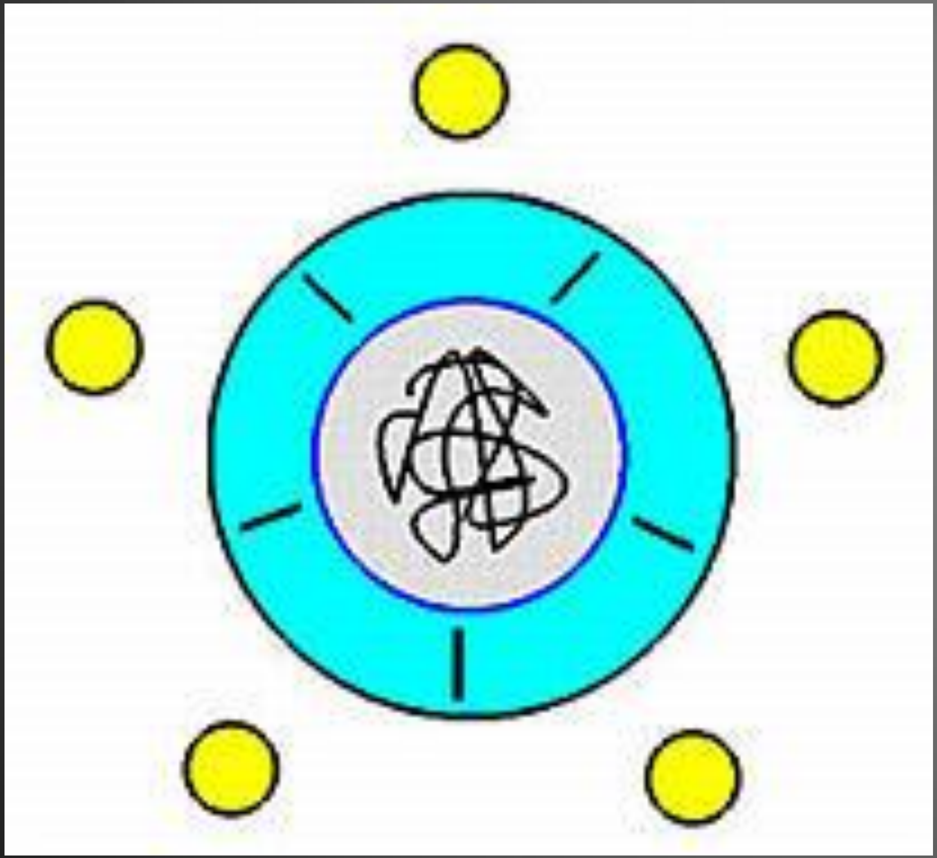
Mutual Exclusion Example

- Do unprotected processing.
- *P()* on a binary semaphore.
- Do protected processing (critical section of code).
- *V()* the binary semaphore.

Deadlock

- A process is deadlocked if it is waiting for an event that will not occur.
- Four conditions for deadlock:
  - mutual exclusion
  - wait for
  - no preemption
  - circular wait

Classic Deadlock Example



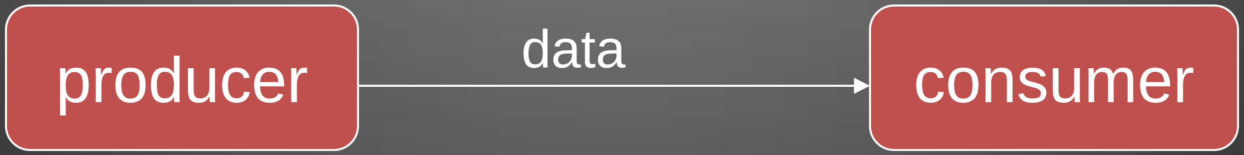
- The dining philosophers problem is invented by E. W. Dijkstra. Imagine that five philosophers who spend their lives just thinking and feasting. In the middle of the dining room is a circular table with five chairs. The table has a big plate of spaghetti. However, there are only five chopsticks available, as shown in the figure. Each philosopher thinks. When he gets hungry, he sits down and picks up the two chopsticks that are closest to him. If a philosopher can pick up both chopsticks, he eats for a while. After a philosopher finishes eating, he puts down the chopsticks and starts to think.

Reentrancy

- Code is said to be reentrant if it can be reentered by one process after another process has entered but hasn’t exited yet and still operate properly.
- Code that is not reentrant *and* is accessible by multiple processes must be protected by a binary semaphore.

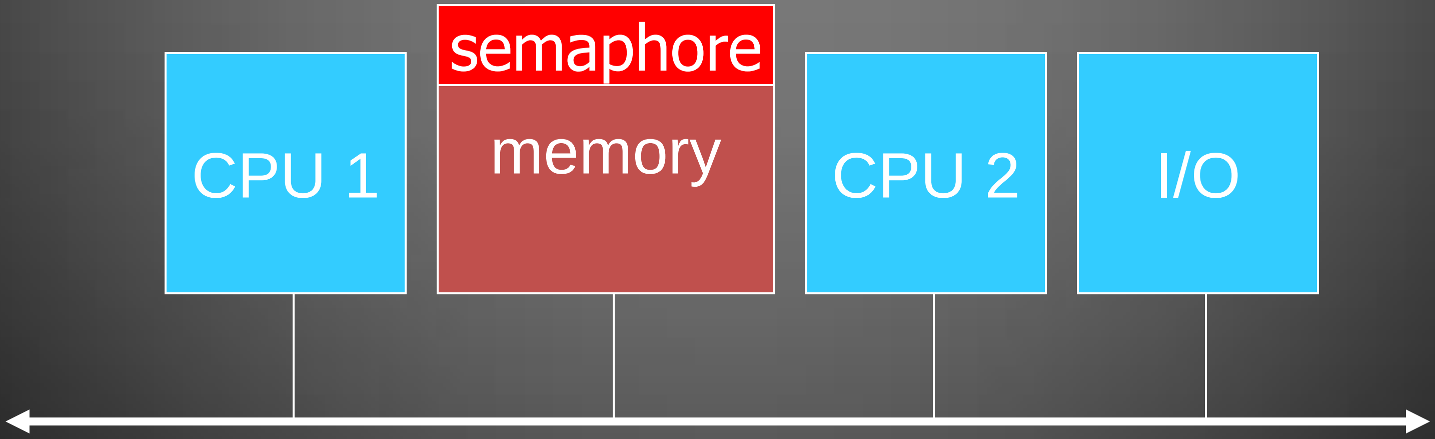
Producer / Consumer

- A producer process places data in the queue.
- A consumer process takes data from the queue.
- The consumer must take the data faster than the producer can place the data in the queue.

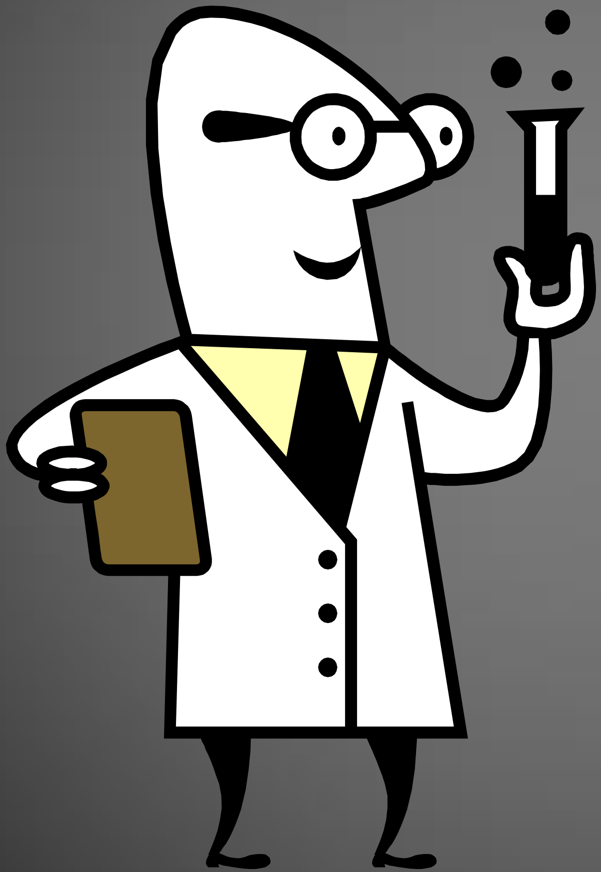


Multiprocessor / Multicore

- Multiple processors (or processor cores) can access the same semaphore simultaneously only if an atomic read-modify-write memory cycle is supported.



Lab Session #8



- Architecture Model Partitioning Considerations

Architecture Module Specification (AMS)

- Identifies the requirements allocated to this architecture module.

Architecture Interconnect Specification

- Establishes the characteristics of the communication channels on which information travels between modules.
- It describes the transmission medium and information formats.

Timing Requirements

- Each architectural module has an associated timing requirement.
- The aggregate system timing must meet or exceed the timing specification from the requirements model.

Architecture Dictionary

- Identifies all control and data flows between architecture modules.

Mapping requirements to architecture

- Each requirement element from the requirements model is assigned possible architectural attributes.
- Requirement elements of compatible architectural attributes are then assigned possible groupings.

Architectural attributes

- Electrical or mechanical?
- High or low power?
- Analog or digital?
- Small signal or large signal?
- Complex or simple control?
- Fast or slow timing requirement?

- Modules might contain multiple attributes.
- Allocation of processes to modules is an exercise in balancing.