



*Using Reservation  
Technology in Linux*

***January, 2002***

# Embedded Applications

- **Embedded applications always have performance constraints**
  - **Throughput**
    - **Good average response. Priorities reflect throughput levels, sometimes modified for fairness**
  - **Soft real-time**
    - **Priority-driven, but rare transient overloads are ok**
  - **Hard real-time**
    - **Priority-driven. Must prevent overloads from affecting critical events**
- **Full RTOS capabilities are critical:**
  - **Kernel preemptibility**
  - **Good priority management,**
  - **Control over interrupt priorities**
  - **Priority inheritance**
  - **High resolution clocks and timers**

# Priorities

- **Both General-purpose and Real-Time OSs – Priorities are the ONLY mechanism for controlling performance**
  - **First difference: How precisely priorities are handled**
  - **Fixed vs. dynamic priorities**
  - **Priority Inheritance**
- **Priorities work well for static load requirements**
  - **E.g., Rate Monotonic, Deadline Monotonic priority assignments**
- **Priority is often a difficult mechanism to work with**
  - **No inherent connection to any performance metric**
  - **Dynamic priorities generally handled by unpredictable heuristic**
  - **Dynamic load means priorities poorly reflect performance requirements**
  - **Under overload, predictability suffers greatly**

# Improving on Priorities

- **Ideal OS Performance Interface**
  - Give OS actual requirements in application terms, e.g.,
    - Generate video frame every 33.3 ms.
    - Handle event within 150  $\mu$ s.
    - Make 7 ms. periodic thread
- None of these have anything to do with priority
- Knowledge of priorities don't directly help
- Priority choices in such systems are intimidating
  - Rate Monotonic Analysis
  - Deadline Monotonic Scheduling
  - Deadline Scheduling
  - Imprecise Computations
  - Etc.

# Non-Priority Performance Control

- **There are several approaches to improving on priorities, e.g.,**
  - **Utility / value function scheduling**
  - **Deadline scheduling**
  - **Reservations**
- **Reservations**
  - **Mechanism closely tied to performance**
  - **Parameters are directly in time units (nanoseconds)**
  - **Admission control uses Deadline Monotonic Analysis**
    - **Priorities automatically generated and managed**
    - **Programmer need not be aware of DMA**
  - **Enforcement uses priorities and precise timer management**

# CPU Reservations

- **Goal: Guarantee either:**
  - Access to the CPU by a critical thread (or set of threads)
  - Protection of a set of threads from a poorly-behaved thread (or set of threads)
- **Reservation Set (RS) is a set of threads sharing a CPU and/or a network reservation**
- **RS requests:**
  - A given amount (C) of computation time
  - A given period (T) within which C must be guaranteed available
  - A given deadline within each period by which C must be done
  - A reservation depletion behavior (Hard, Soft)

# Temporal Reservation Parameters

- Replenishment Interval: “T”
- Execution Budget: “C”
- Deadline: “D”
- Depletion Behavior:
  - “Hard” or “Soft”

Admission control guarantees maximum of “C” units of resource within “D” units in each replenishment interval.



Soft reserves can compete at background priority after reserve is depleted.

time

# CPU Reservation Example

- **Reservation:**
  - **Computation time  $C = 6$  ms.**
  - **Period  $T = 33$  ms.**
  - **Deadline = 33 ms.**
  - **Depletion behavior = Hard**
- **Hard – When  $C$  is exhausted, RS blocked until  $T$  expires**
- **Soft – When  $C$  is exhausted, RS continues at low priority**
- **Result:**
  - **RS guaranteed to run for 6 ms. every 33 ms. (18.2%)**
  - **Other threads can have all the remaining time (81.8%)**
  - **Priorities handled automatically by OS**
  - **Think of this as a “temporal firewall”**

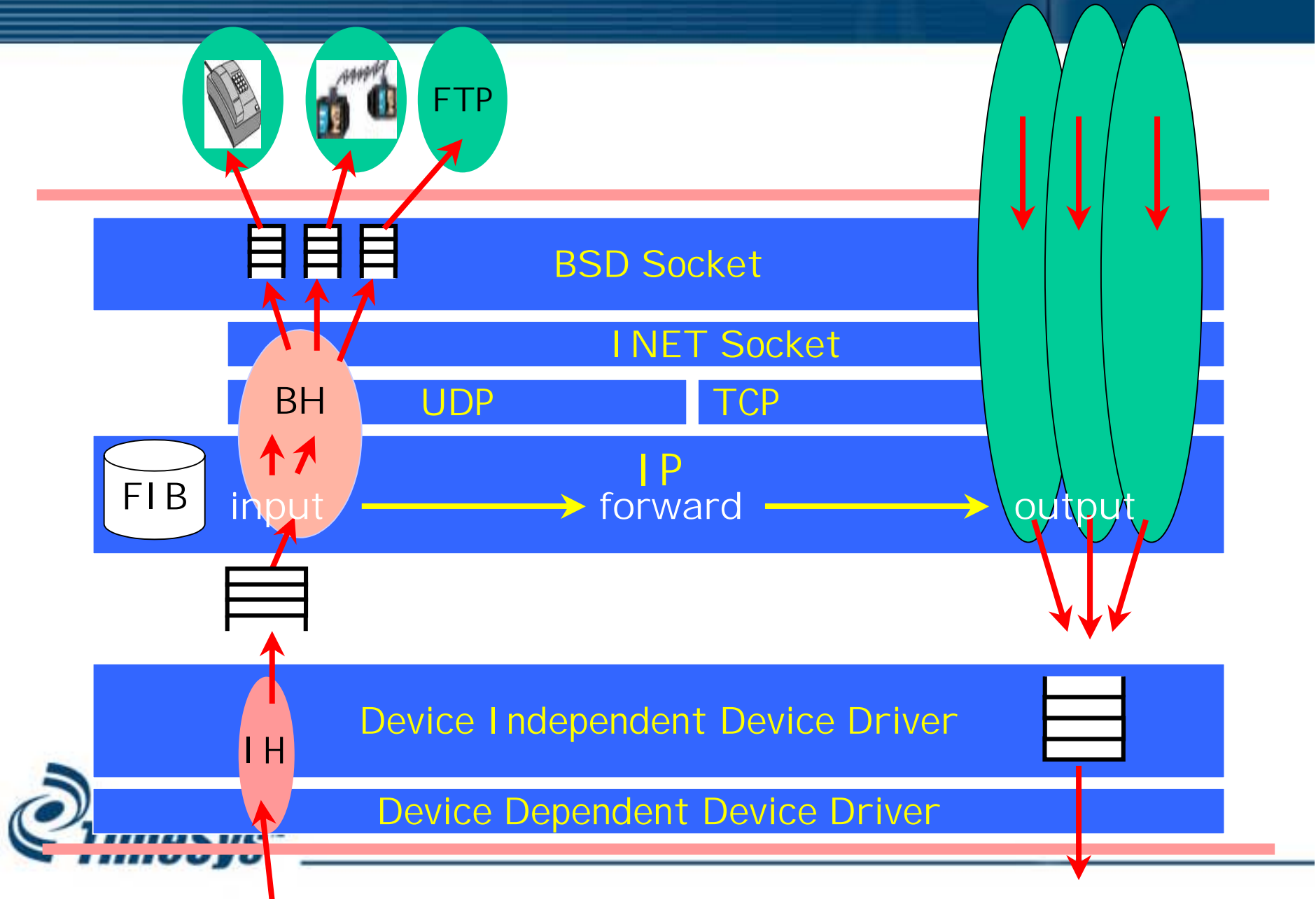
# Applications for CPU Reservations

- **Overload management (Real-Time or not)**
  - Give reservations to critical tasks so they aren't affected by overload
- **Support for Hard Real-Time**
  - Predictable Applications can expect hard real-time behavior, regardless of other applications
- **Support for Soft Real-Time**
  - The QoS of an application can be changed dynamically, by changing the amount of resources they get
  - Applications can be written with adaptive behavior
- **Mixed Real-Time & Non-Real-Time**
- **Unbounded components mixed with Real-Time components**
- **Faulty components cannot hog CPU indefinitely**
- **Any application with a reserve is guaranteed to make progress**

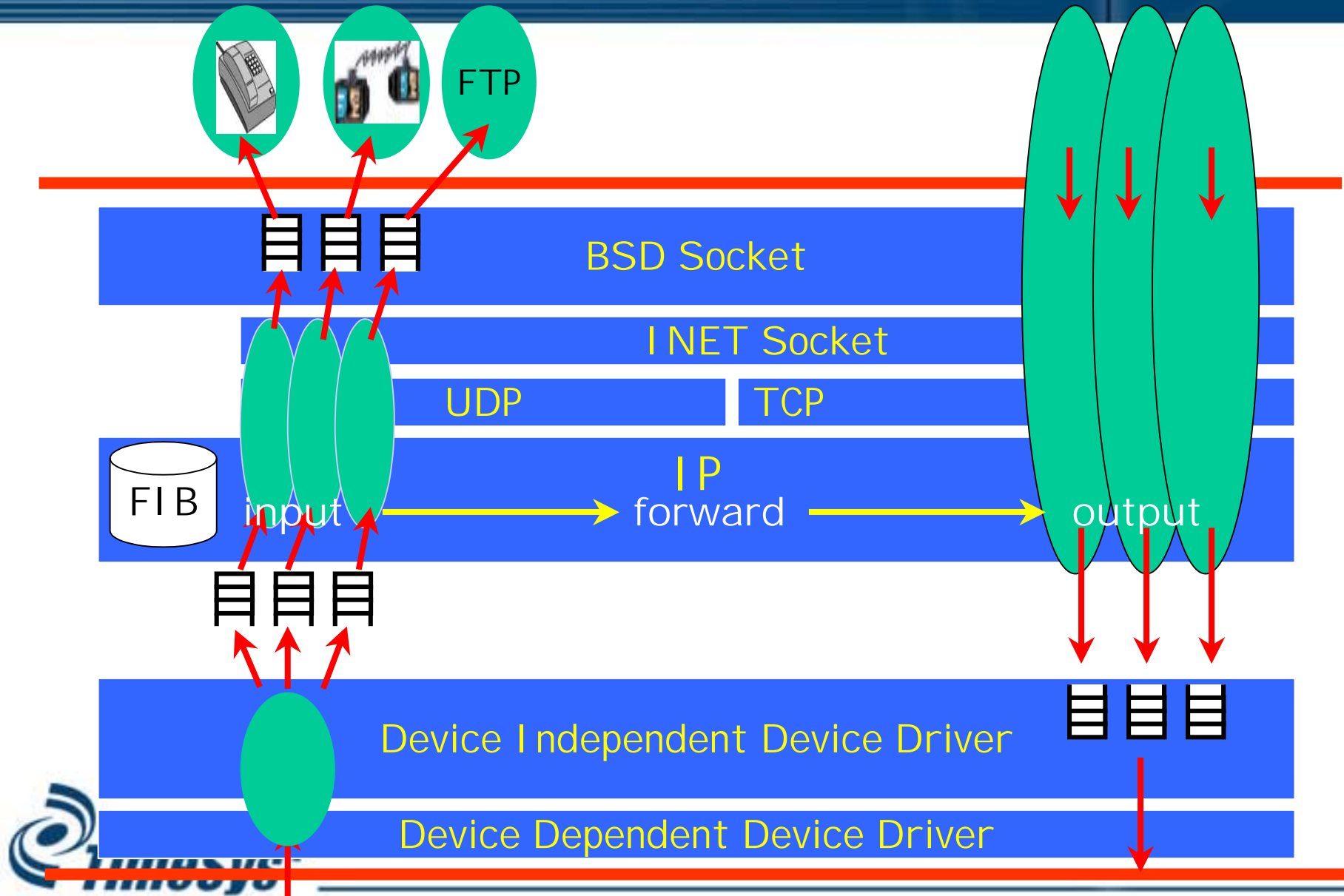
# Network Reservations

- **Guarantees incoming and outgoing packet bandwidth capability**
  - **Outgoing packets guarantee number of bytes per period on a socket to network interface**
  - **Incoming packets guarantee number of bytes per period to a socket from the network interface**
  - **Incoming buffer allocation also guaranteed**
- **Separate IP queues used for each reservation**

# Standard Linux Networking Architecture



# Network Reservation Architecture



# Potential Applications

- **A few application possibilities**
  - **Air traffic control – Reserving radar & confliction detection resources**
  - **C<sup>3</sup>I – Protecting message handling and sensor management from X-windows & fusion algorithms**
  - **Telecom – Protecting call management from failure handling**
  - **Car navigation – Protecting satellite data handling from display and entertainment management**
  - **Process control – Protecting sensor/actuator management from user interactions**

# Summary

- **Reservations provide a step beyond priorities**
- **Can guarantee response in overload**
- **Can guarantee resource availability even against unbounded applications**
- **Do not necessarily require any application changes**
- **Enables an entirely new way to think about embedded application architecture**
  - **Overloads in any application**
  - **Soft real-time / QoS resource management**
  - **Hard real-time resource management in mixed environments**

