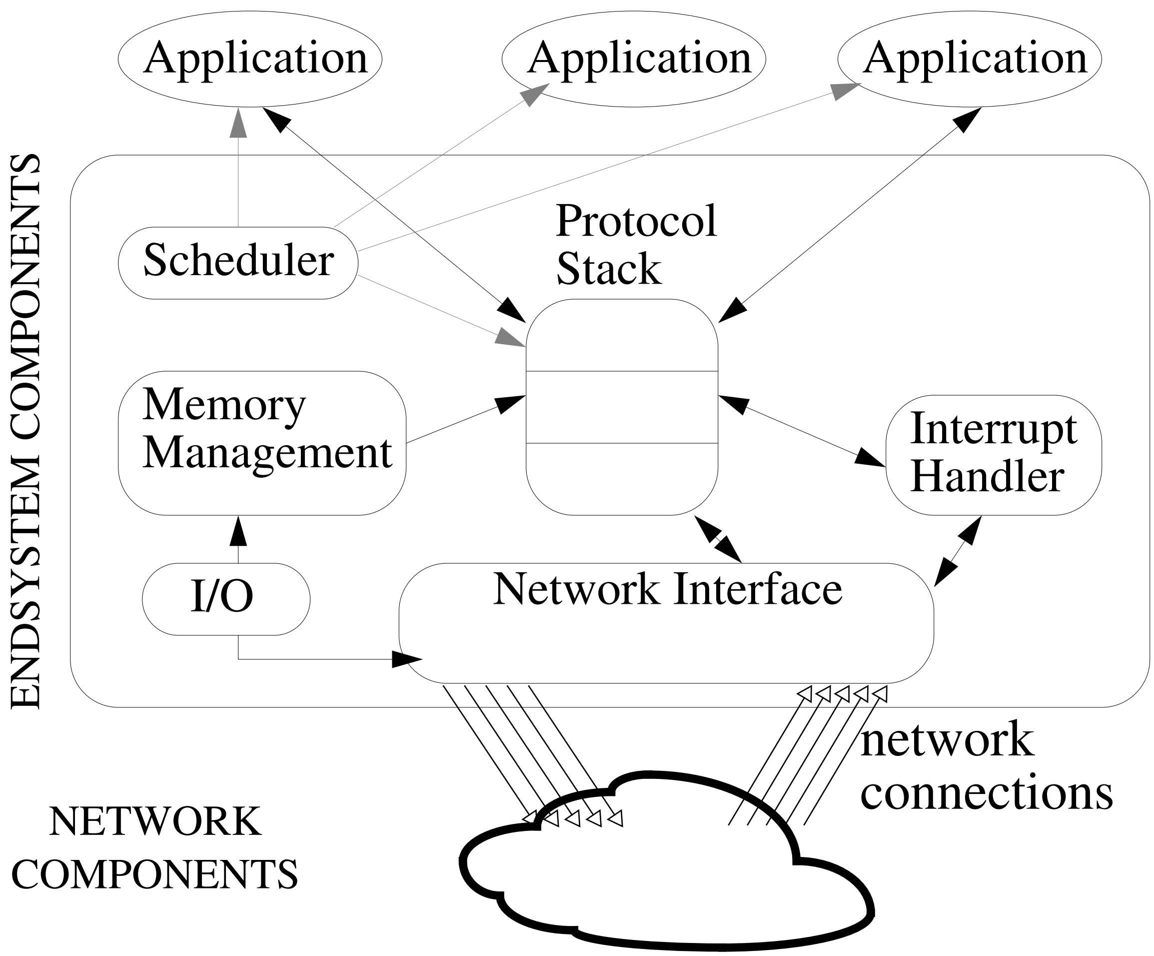
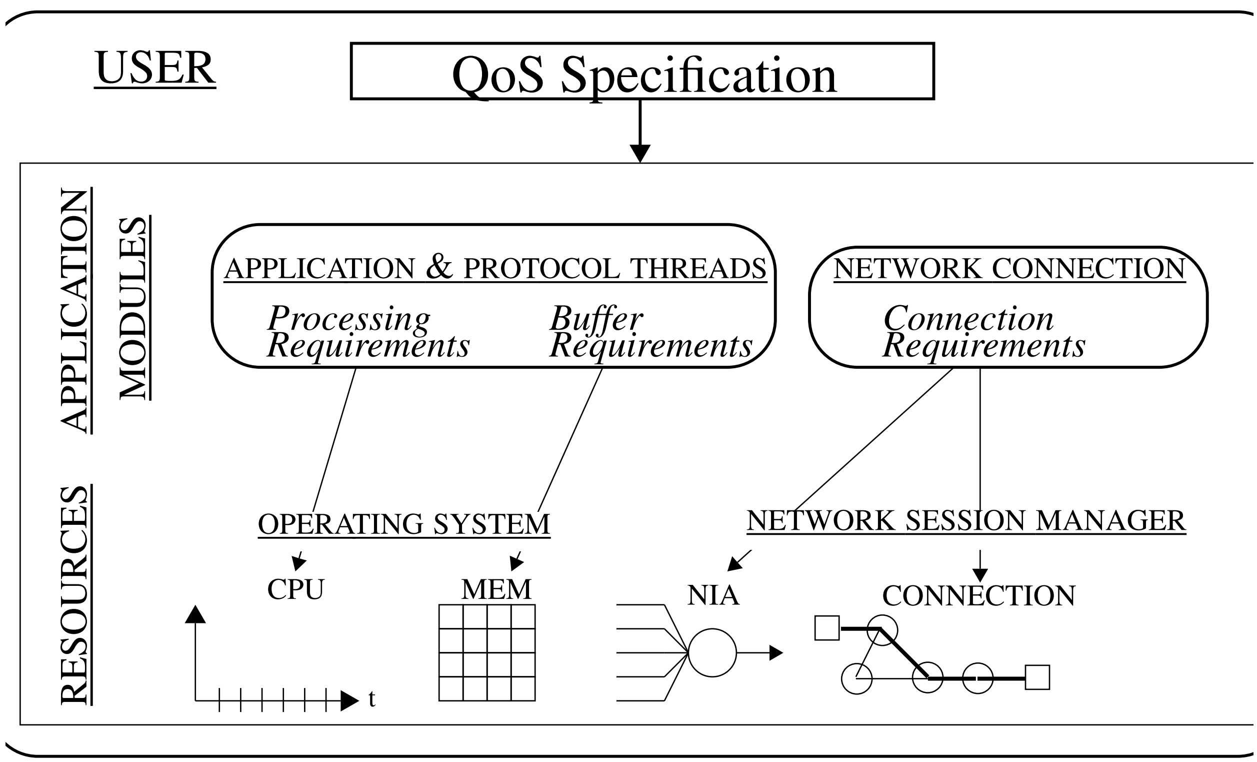
Need for QoS Support within the OS

- Network guarantees QoS parameters only upto the network boundary
- The host can violate bandwidth and delay guarantees provided by the network because–
  - →Processing load on endsystems is more
  - →Complex interaction of endsystem components



QoS Support Components

- QoS Specification
  - →Determine typical classes of applications
  - →Determine QoS parameters in each class
- QoS Mapping
  - →QoS parameters →resource requirements
  - →Components in end-to-end path are
    - ∗ CPU – Protocol processing
    - ∗ Network connection–Data transfer
    - ∗ Network interface–Pacing at sender

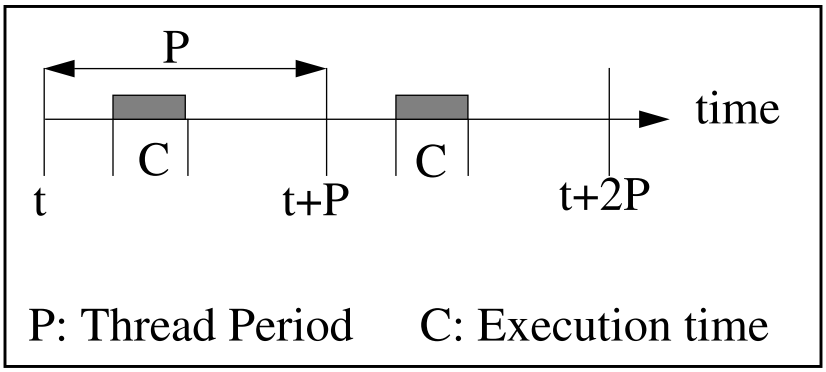


QoS Components

- QoS enforcement
  - → Enforce CPU requirements derived for each protocol session
  - → Provide admission control within the OS (schedulability test)
- Implement protocols to take advantage of QoS support

Deriving Processing Requirements

- Periodic processing model
  - → In each period P, B_p PDUs are processed
  - → Computation time C ≡ time to process B_p PDUs

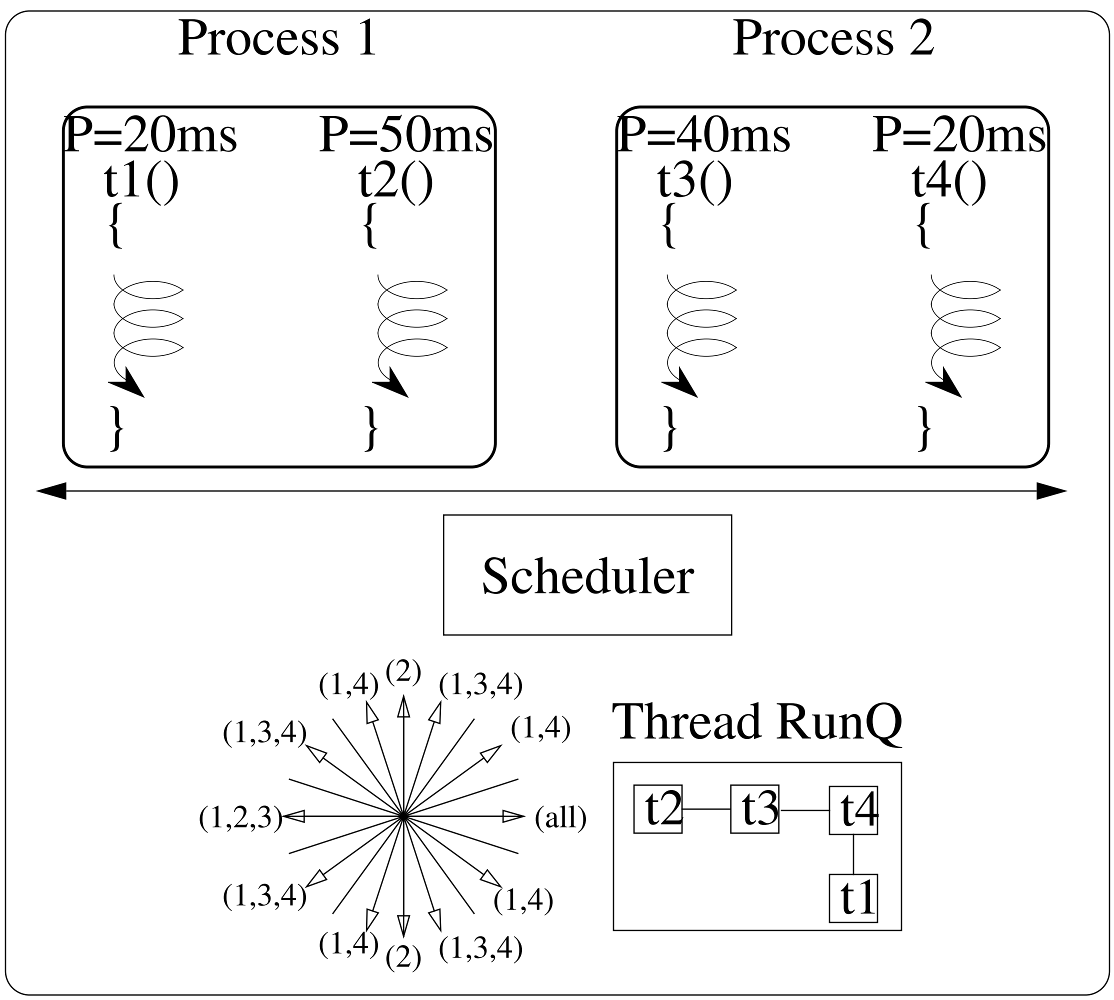


Processing Abstraction Choices

- Process Mechanism
  - → Only coarse grained control possible
  - → Cannot support multiple sessions in a process
- Real-time Thread Mechanism
  - → More complex than necessary
  - → Sharing variables between threads requires locking
- UNIX Signal Mechanism
  - → Simpler than real-time threads
  - → No real-time guarantees
- Real-time Signal Mechanism
  - → Provides real-time guarantees
  - → Optimal mechanism for protocol processing

Real-time Threads in RT-Mach

- RT-Mach implements several real-time scheduling policies
- Several threads are active in a process
- A thread has a time period 'P' and a computation time 'C'
- Higher rate threads have higher priority (Rate Monotonic)



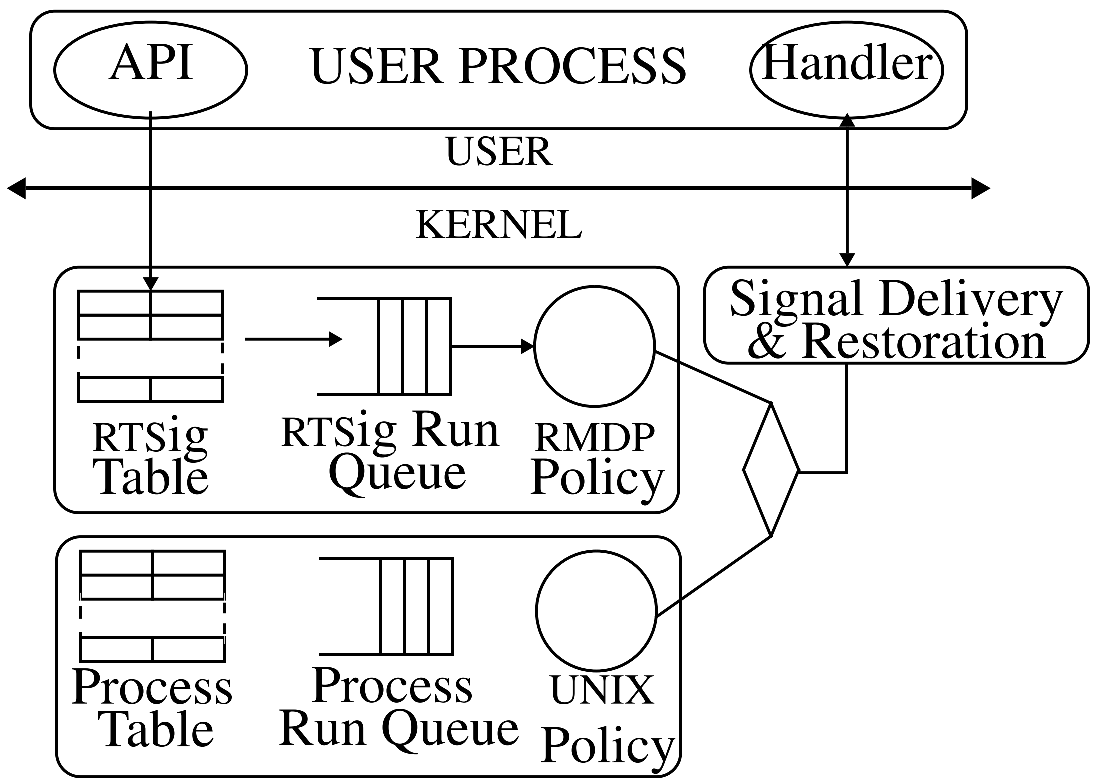
Limitations of RT-Mach

- Implementation problems
  - → Thread packages are complex to implement
  - → Not available on most UNIX systems
  - → Changing UNIX scheduler is a bad idea
- Efficiency problems
  - → Threads that share variables must lock them before access
  - → Too much context switching overhead

UNIX Signals

- What is a signal ?
  - → A signal is a software interrupt
  - → Delivered asynchronously to a process
  - → Signal handler function called on delivery
- Limitations with current signals
  - → No real-time behavior can be expected
  - → Only one periodic signal per process
  - → No performance guarantees (no admission control)

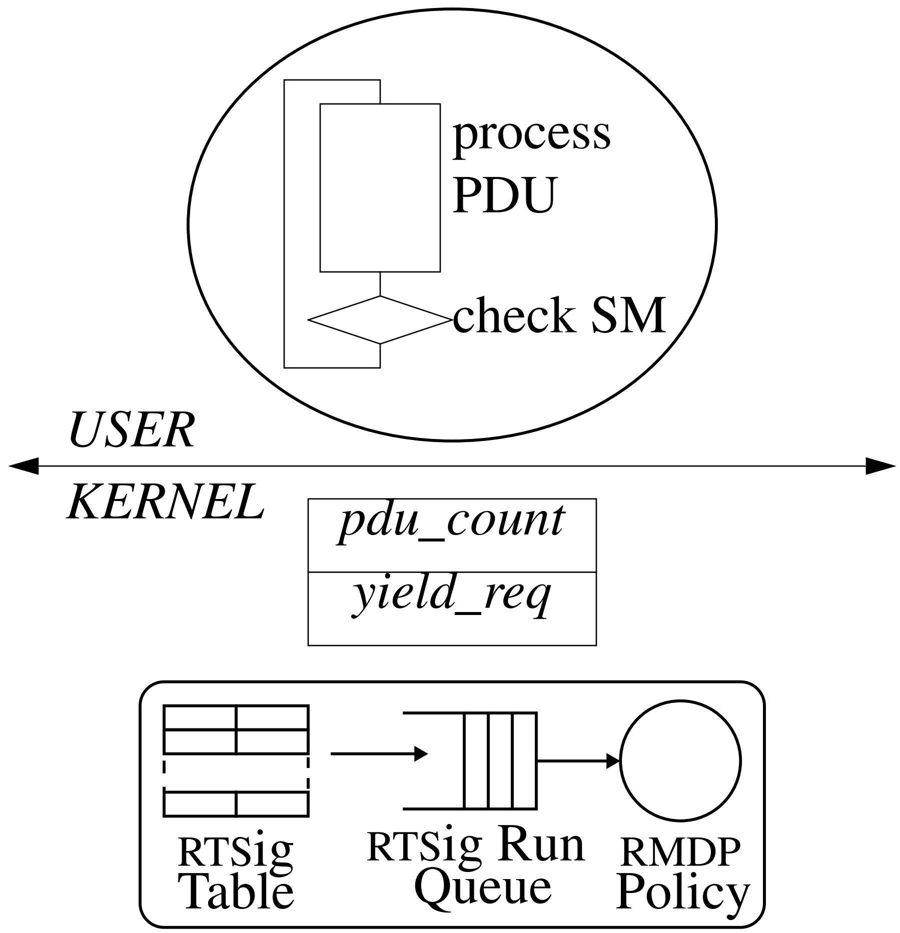
Real-time Signal (RTSig)



- RTSig setup
  - → Specify handler function and scheduling attributes
- RTSig activation
  - → Activated by the clock every period
- RTSig scheduling
  - → RMDP scheduling policy
- RTSig delivery

The RMDP Scheduling Policy

- A running RTSig handler is never pre-empted in the middle of an iteration
- The RTSig handler checks shared memory after each iteration
- Yields CPU if higher priority RTSig is ready

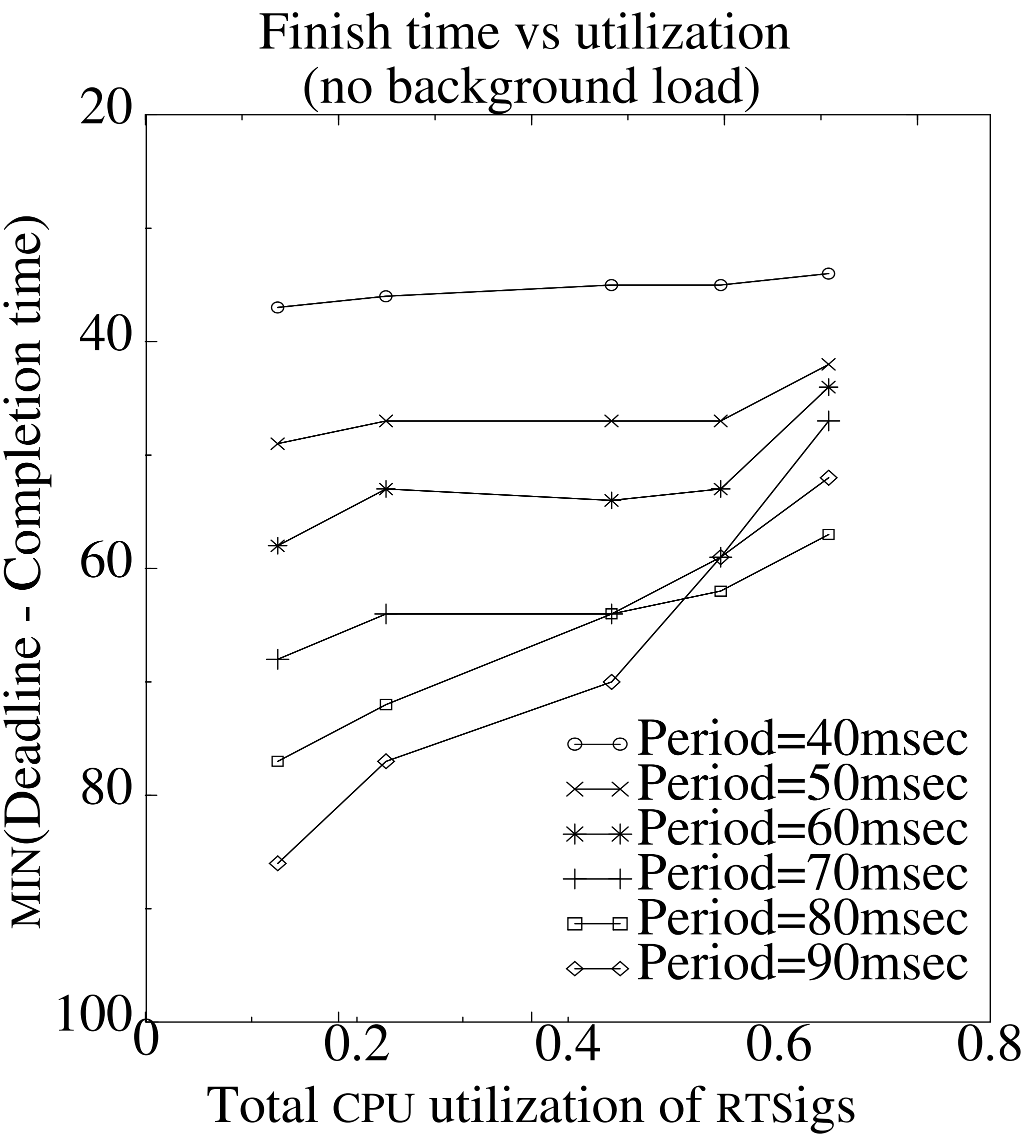


Advantages of RMDP

- No need for locking shared variables
  - → Eliminates system calls for locking
  - → Simplifies protocol code
- No need to implement real-time locking schemes in kernel
- Reduces number of context switches
  - → b_p PDUs can be processed in each run
  - → Scheduling and data movement occur in the same context switch

Experimental Results

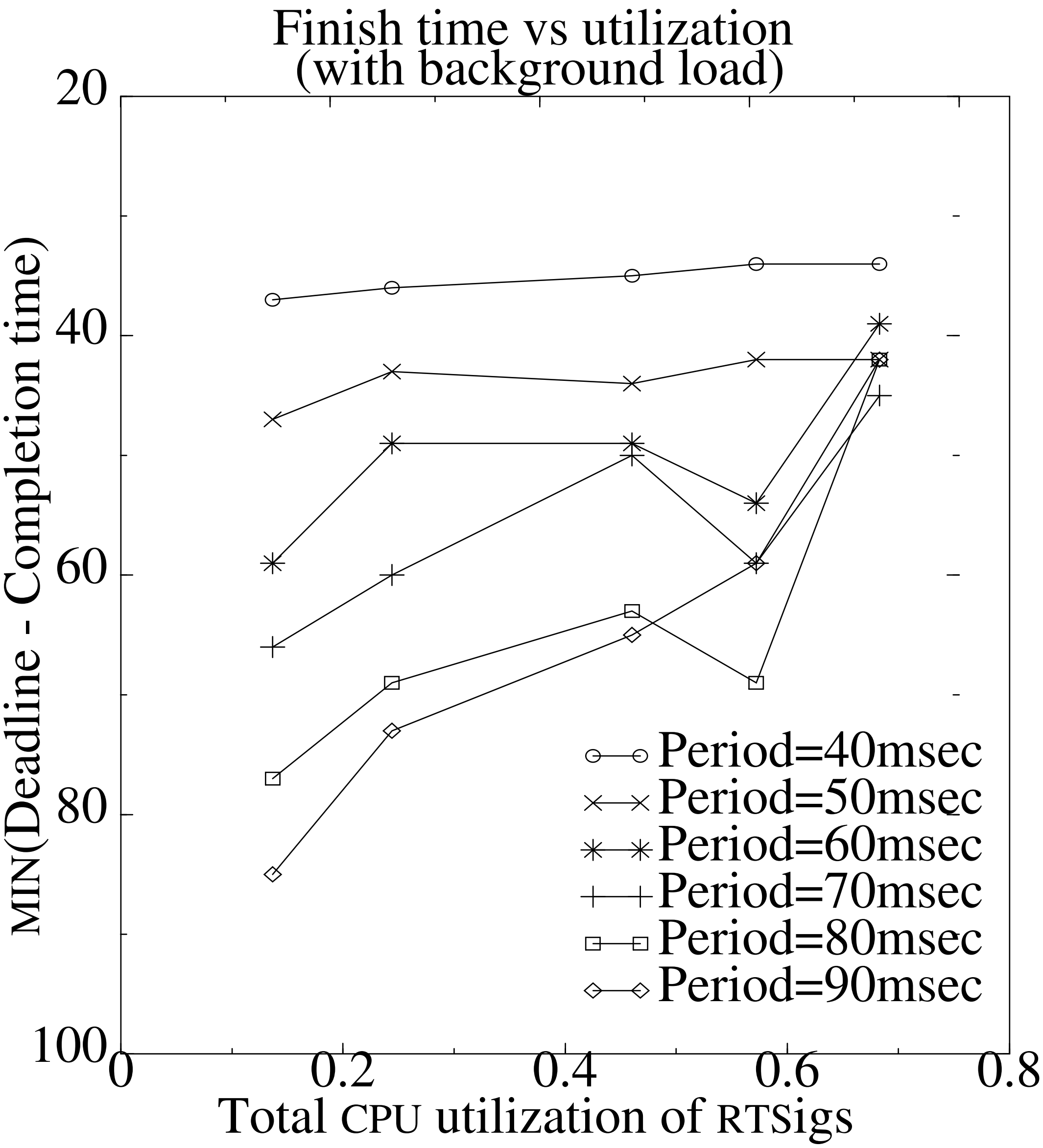
- 6 RTSigs each in 2 processes
- Utilization was varied



- All RTSigs finish before deadline
- Similar behavior with background load

Experimental Results

- Same experiment with background load



- Again all deadlines are met

Experimental Results

- Introduced pre-emption delays to check number of context switches saved