Implementation of µITRON Embedded Operating System Specification on top of L4 Microkernel

Anton Burtsev, antonb@cse.unsw.edu.au
Supervisor: Ihor Kuz, ikuz@cse.unsw.edu.au

November 16, 2004

Outline

- µITRON overview
- Implementation on top of L4
- Protected extensions and µITRON future

µITRON overview

Execution Environment

- Single unprotected address space
- Tasks

  entry point + thread + stack + task activation counter
- System call – just a C function call
- Uniprocessor machine

Basic µITRON Objects

- Synchronization Primitives
  - Semaphores
  - Event flags
- Communication Primitives
  - Data queues
  - Mailboxes
- Memory Management
  - Fixed-size memory pool

Wait Queues

Each µITRON object has a wait queue either in a:

- FIFO or
- Priority order

Preemption Control

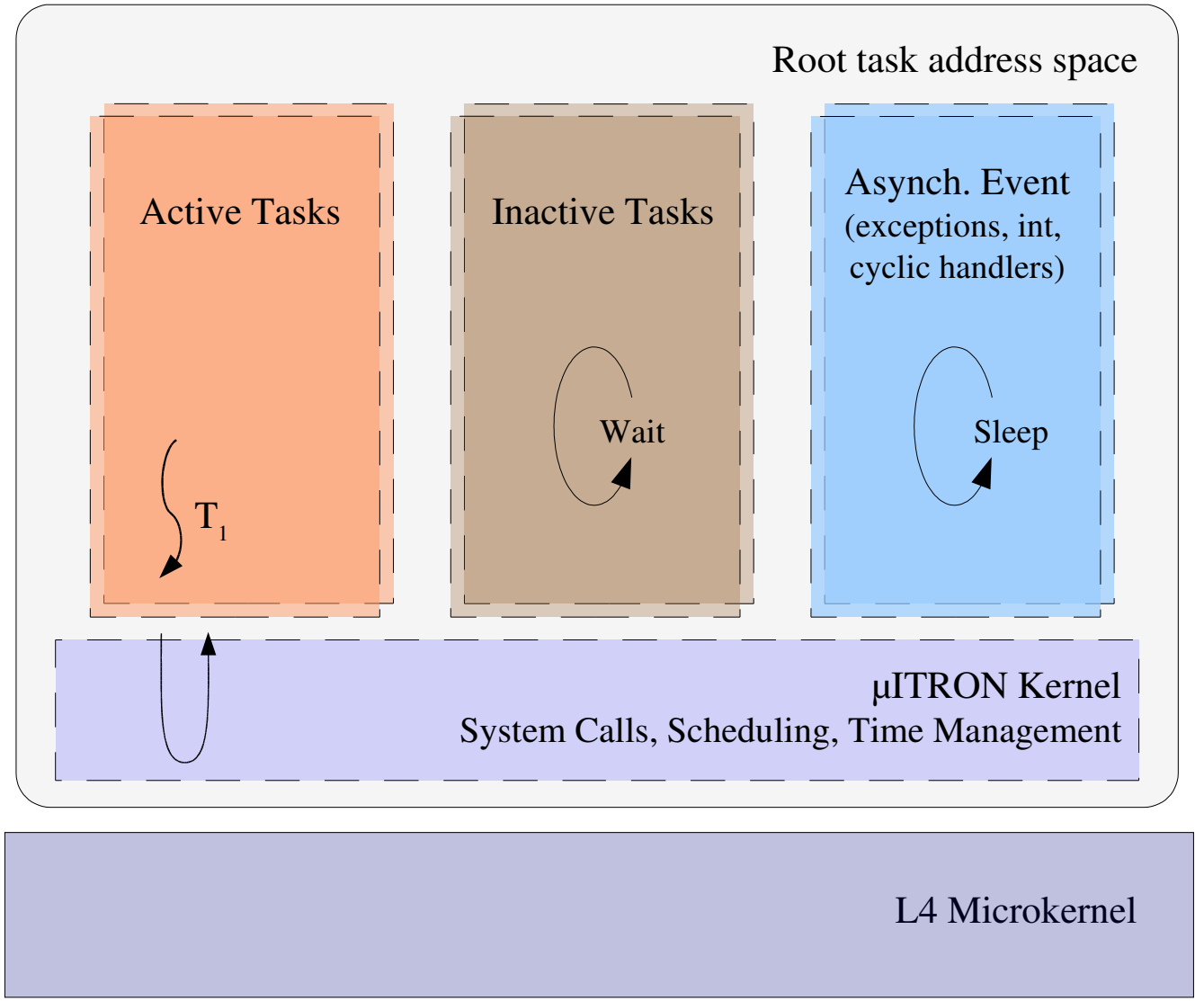
- CPU locking
- Dispatching control

Kernel Deployment

- Kernel as a static library
- Set of C include files
- Applications are linked with the kernel into bootable image

Implementation

Architecture



Synchronization

Problem:

- Provide wait-free execution
- Avoid priority inversion

Possible solutions:

- Priority inheritance
- Priority ceiling
- Delayed preemption
- Lock-free algorithms

Priority Inheritance

Pros:

- Good average performance (when contentions are rare)

Contras:

- Poor worst-case performance in case of nested locks

Priority Ceiling

Immediate Priority Ceiling Protocol

- Pros:
  - Good worst case performance (high priority task is blocked at most once by all lower priority tasks)
  - Can handle nested locks
- Contras:
  - Poor average performance caused by two priority changes (context switches + scheduler queues operations)

Protocol implementation uses separate thread for serialization of priority change requests.

Delayed Preemption

Pros:

- Good worst case performance
- Good average performance
- Can handle nested locks

Contras:

- Requires careful programming

Lock-Free Algorithms

Pros:

- Best worst-case performance
- Good average performance
- Work for multiprocessor machine

Contras:

- Hard to implement for complex data structures

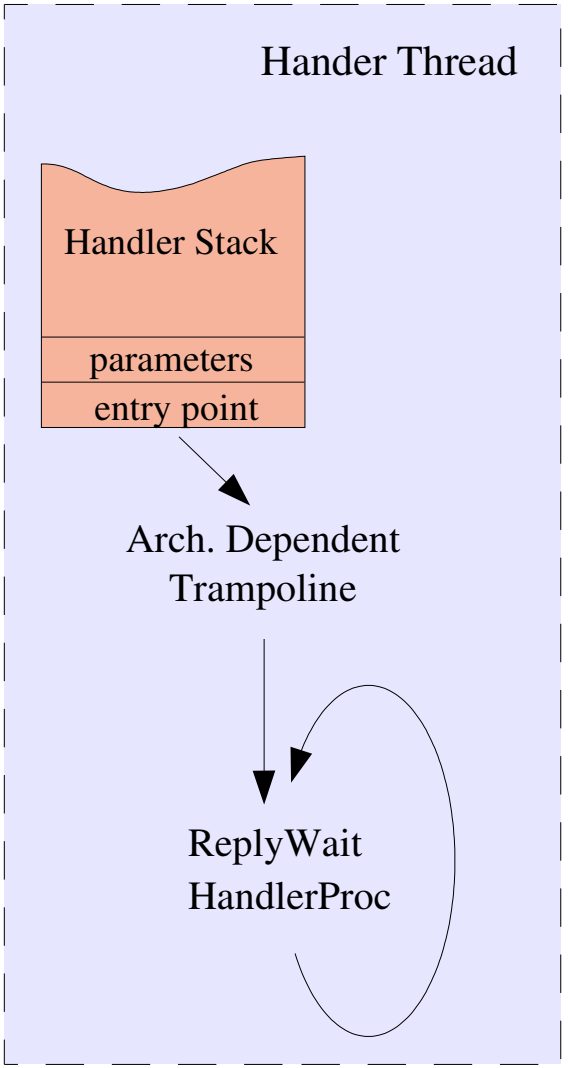
Conclusion:

Lock-free kernel - promising future direction

Non-task Context Emulation

- Interrupt handlers
- CPU exception handlers
- Task exception handling routines
- Time event handlers (cyclic time event handler)

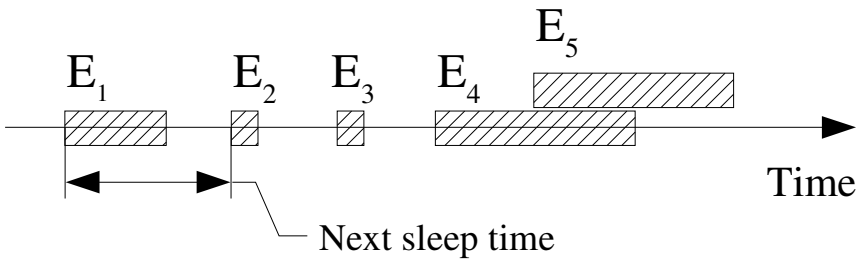
Non-task Context Emulation

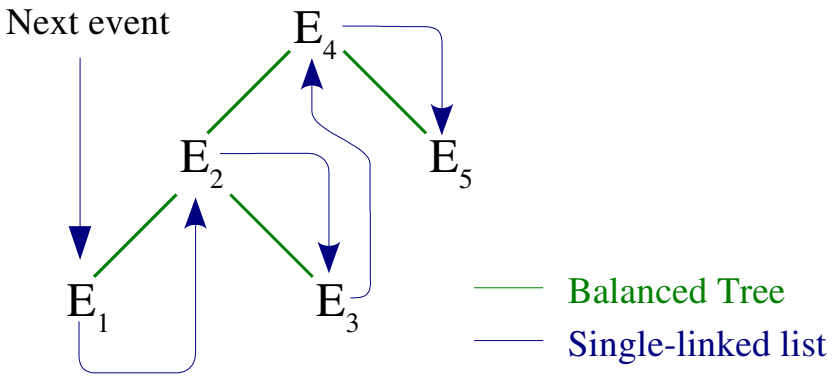


Emulated in a traditional L4 way:

- High priority thread in a ReplyWait cycle
- Startup parameters are passed through the stack
- Activated either by:
  - Timeouts, or
  - Incoming messages

Time Event Management



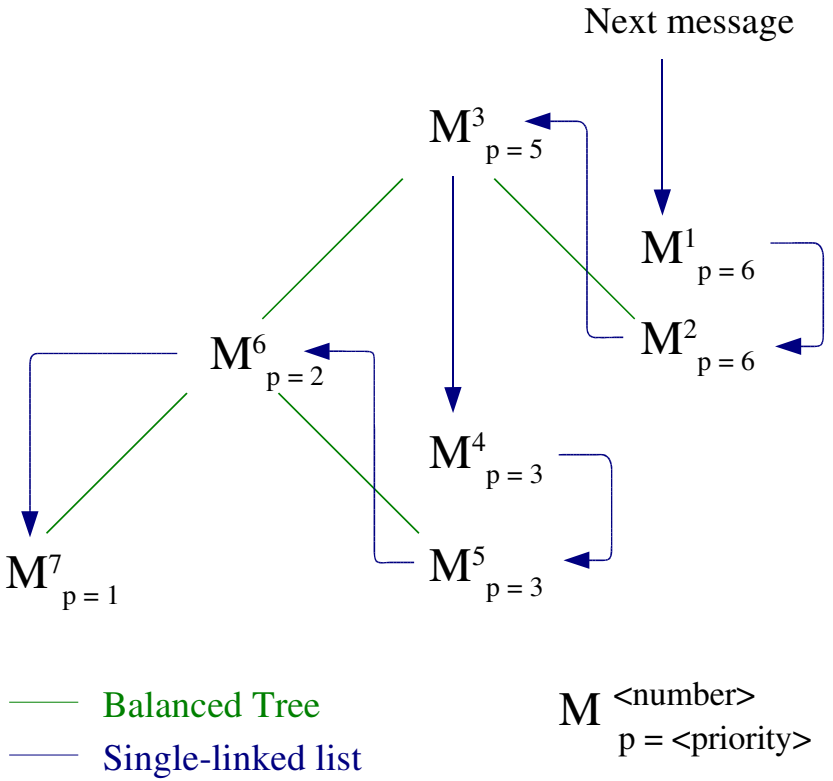


- High priority thread in a Wait cycle schedules events and starts event handlers
- Events are stored in a balanced tree ordered by time and in a single-linked list:
  - O(1) retrieve next event
  - O(log n) add or remove event

Problems:

- Overlapping and long running events
- Should a new thread be created for each dispatched event?

Fine-Grained Priority Message Queue



Messages are stored in a modified balanced tree ordered by priority and in a single-linked list:

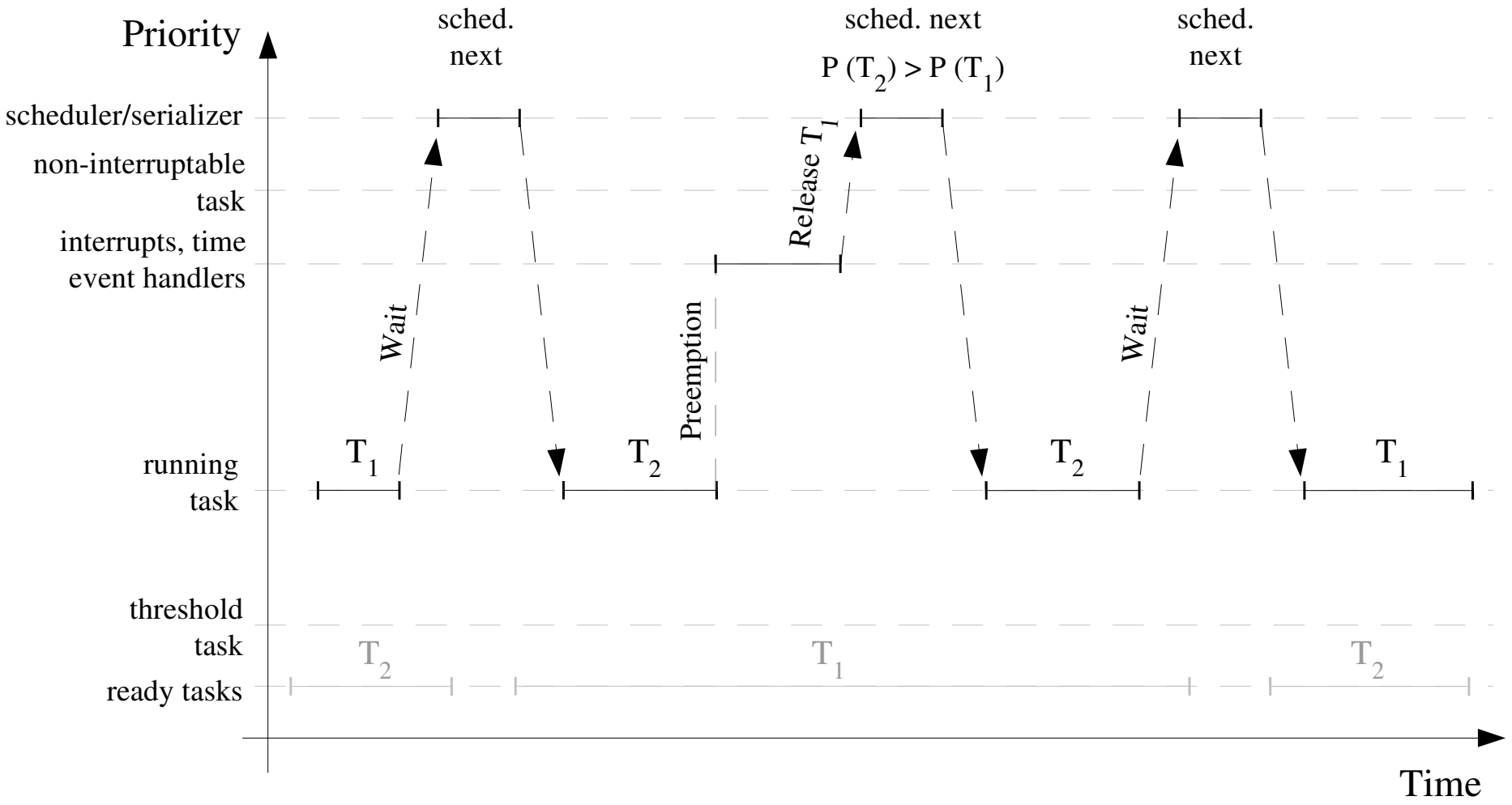
- O(1) retrieve next message
- O(log n) add or remove message
- FIFO ordering between messages with equal priorities

User-Level Scheduling

µITRON API requirements:

- Knowledge of current/next executing task
- Ability to modify queue of ready tasks

User-Level Scheduling



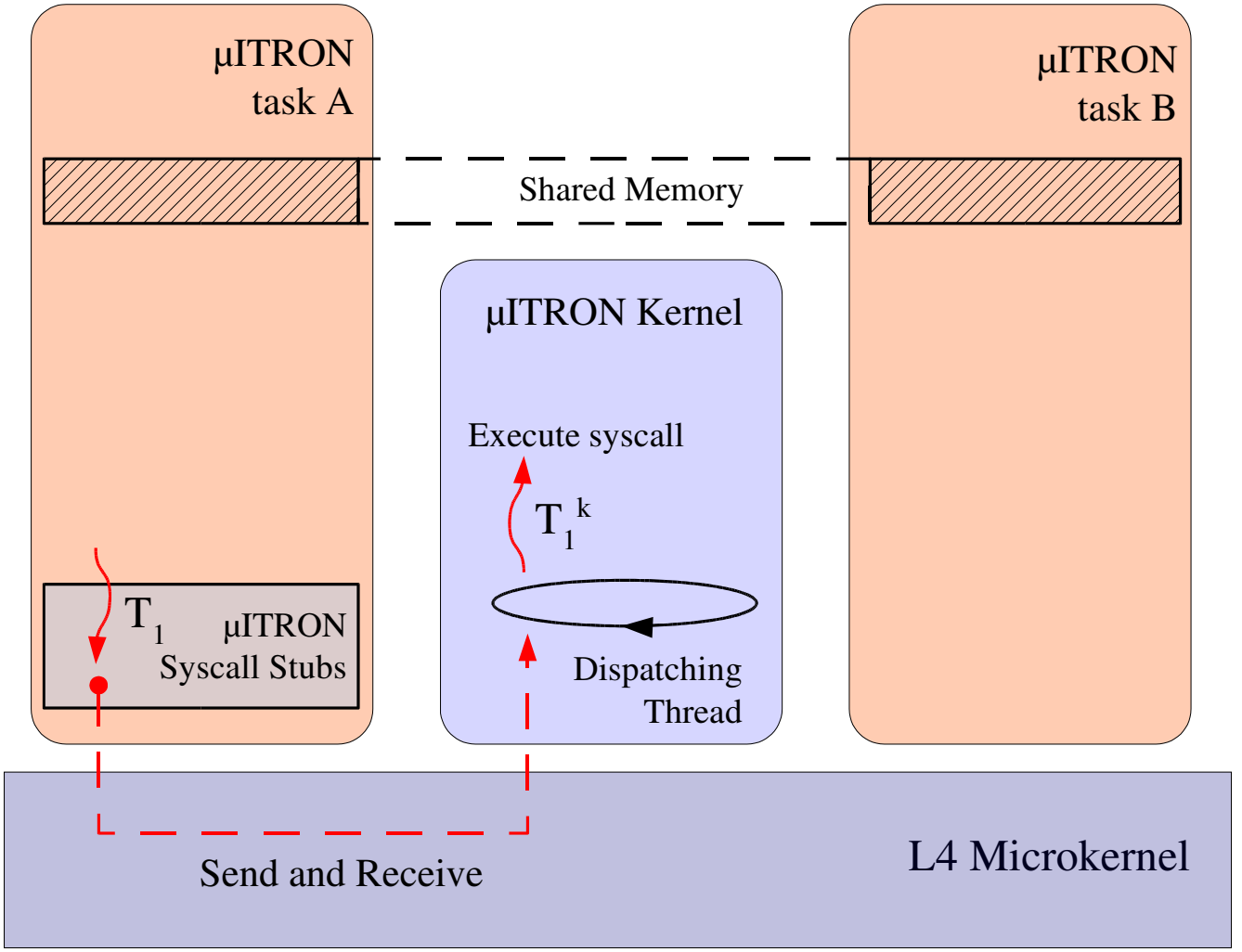
Protection Extensions

Protection Extensions

Protection Extensions is a separate µITRON specification extending base specification with memory protection and access control mechanisms.

- No memory translation
- Protection domains
- µITRON objects reside within protection domains
- New type of object – memory object
- Access control vectors define rights to perform operations on objects

Implementation of Protection Extensions



µITRON Future

Protected Extensions vs T-Kernel

Thank You