Non-Blocking Write Protocol NBW:

A Solution to a Real-Time Synchronization Problem

By:
Hermann Kopetz and
Johannes Reisinger

Presented By:
Jonathan Labin

March 8th 2005

Classic Mutual Exclusion Scenario

- Reader and Writer processes share some piece of memory.
- Critical sections and semaphores used
- Scheduling difficult
  - Task can be blocked at critical section
  - Task can be preempted by high priority task
- Both readers and writers can be blocked at critical sections

Problem Architecture: System

- Distributed real-time system
- Each node contains
  - CPU
  - Memory
  - Communication Controller
  - A dual-ported RAM (DPR)
  - Some nodes also have I/O interfaces

Problem Architecture: System

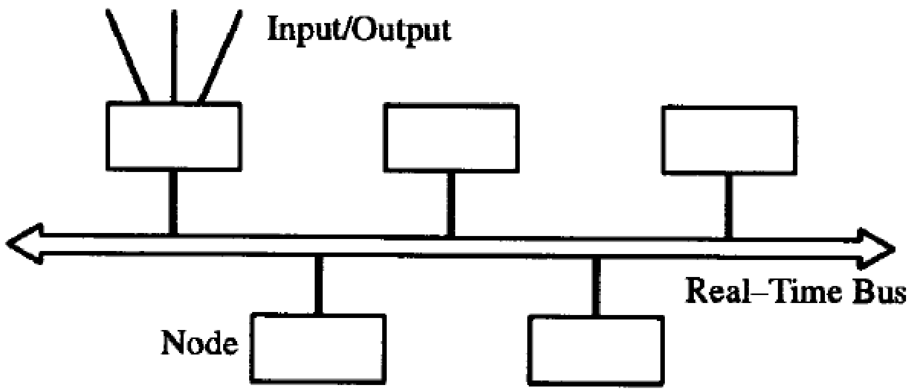
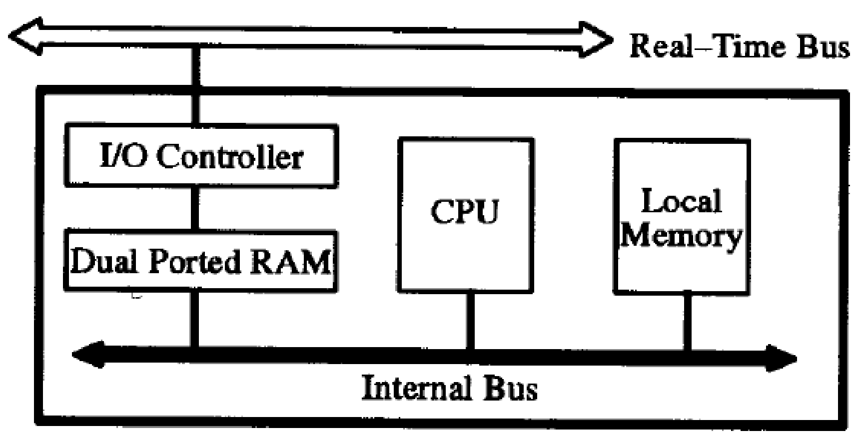


Fig. 1: Distributed Computer System



Problem Architecture: Messages

- Communication Controller serves messages to CPU through DPR
- State Messages
  - new version of message overwrites the previous
  - Similar to programming language concept of a *variable*
- Minimal interval between message instances is known

Problem Architecture: CPU Tasks

- {T} – set of concurrent tasks
- Task T_i
  - c_i = maximum execution time
  - d_i = relative deadline
  - l_i = laxity (= $d_i - c_i$)
- Tasks are preemptable

Synchronization

- Each message type is allocated a structure in DPR
- Communication Controller writes messages to allocated DPR structure each time they are received
- Real-time tasks running on CPU read messages from structure
- One writer. Many readers
- Readers can not simply block since messages are time sensitive

Desired Properties in Solution

- Safety – “If a read operation completes successfully, it must be guaranteed that it has read an uncorrupted version of the data structure.”
  - Reader does not interfere with other readers
  - Reader does not invalidate a write
  - Write corrupts a read
  - We check after a read to ensure that it was not corrupted by a write

Desired Properties in Solution

- Timeliness – “The tasks containing the read operations must complete their execution before their deadlines.”
  - This is hard real-time system.
  - Upper bounds must be known to ensure deadlines are not missed

Desired Properties in Solution

- Non-Blocking – “The writer can not be blocked by the readers.”
  - Information flow: Writer => Reader
  - Readers can be added or removed without effect on the writer
  - Communication Controller simplified: no need for buffer space

The Protocol: The Basics

- Writer free to write at any time
- Readers check after a read operation
  - If no write has occurred during the read: success
  - Otherwise, fail and try again
- To satisfy timeliness, number of read re-tries must have a known upper bound

The Protocol: Define

- Concurrency Control Field (CCF) for each message structure.
  - Size = 1 word
  - Init: CCF = 0;
  - Reading or changing CCF is atomic
  - R = max word
  - Incrementing beyond R wraps to 0

The Protocol

- Writer
  - Increment CCF
  - Perform message structure write
  - Increment CCF again.
- Reader
  - Read CCF
  - Perform message structure read
  - Check CCF for indication of writer interference

The Protocol: Pseudo-Code

Initialization:

```
        CCF_i := 0;
```

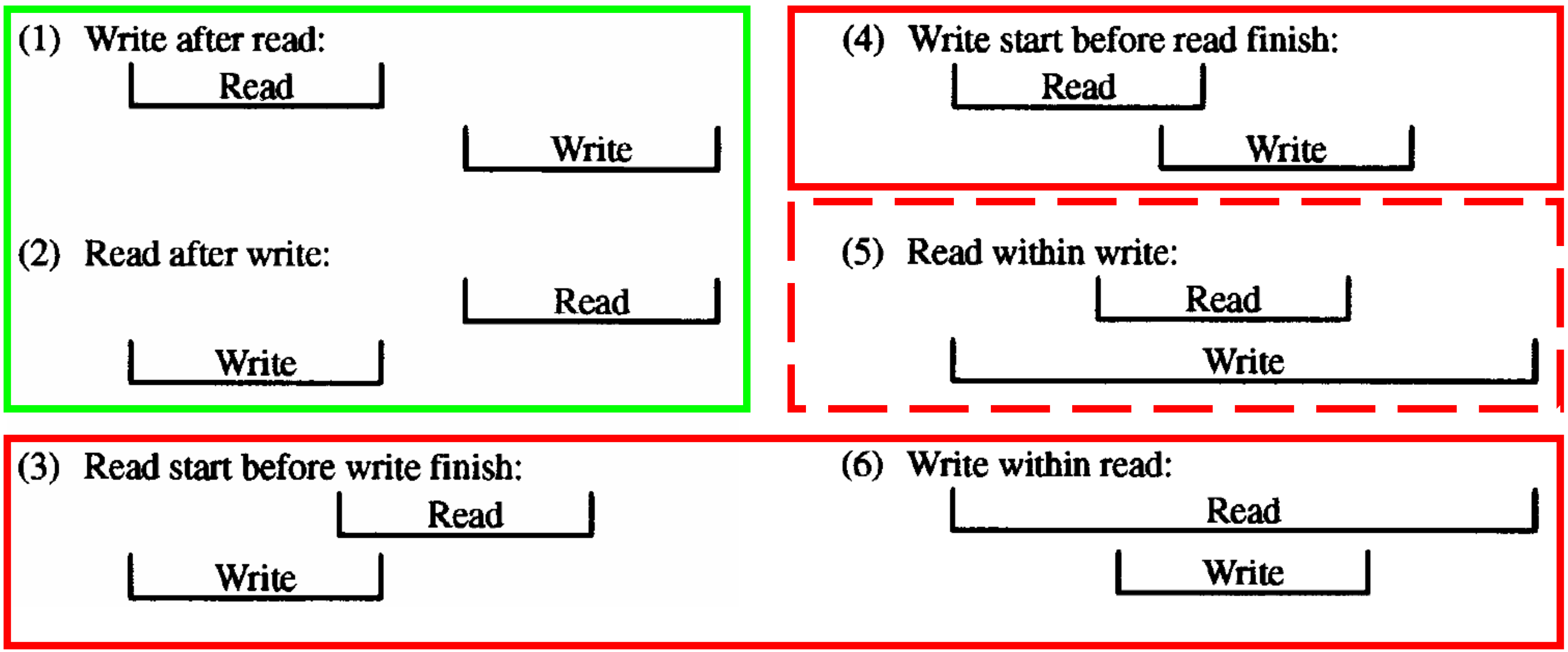
Write message i:

```
start:  CCF_old := CCF_i;
        CCF_i := CCF_old + 1;
        <write buf_i>
        CCF_i := CCF_old + 2;
```

Read message i:

```
start:  CCF_begin := CCF_i;
        <read buf_i>
        CCF_end := CCF_i;
        If CCF_end ≠ CCF_begin or
           CCF_begin = odd then goto start;
```

Correctness of Safety Property



Schedulability Analysis: Definitions

Attributes of messages:

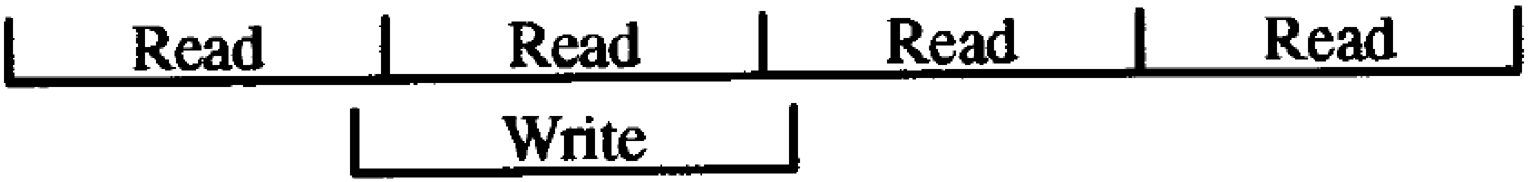
- d^r = max time of a read without retry
- d^w = max time of a write
- mint = minimum arrival interval of messages

Attributes of tasks:

- c_o = max execution without read-retries
- c_n = max execution with read-retries
- d = deadline
- l_o = min latency without read-retries ($d - c_o$)
- l_n = min latency with read-retries
- N_i = max number of interferences of read by write operations

Schedulability Analysis: Single Interference

- Assume read and write about equal:
  - $(d^r - \delta) < d^w < (d^r + \delta) \quad \text{for } \delta \ll d^r$
  - Worst case: Interference by one write => max: 3 read-retries
  - Increase execution of reading task by $3d^r$



Schedulability Analysis: Multiple Interferences

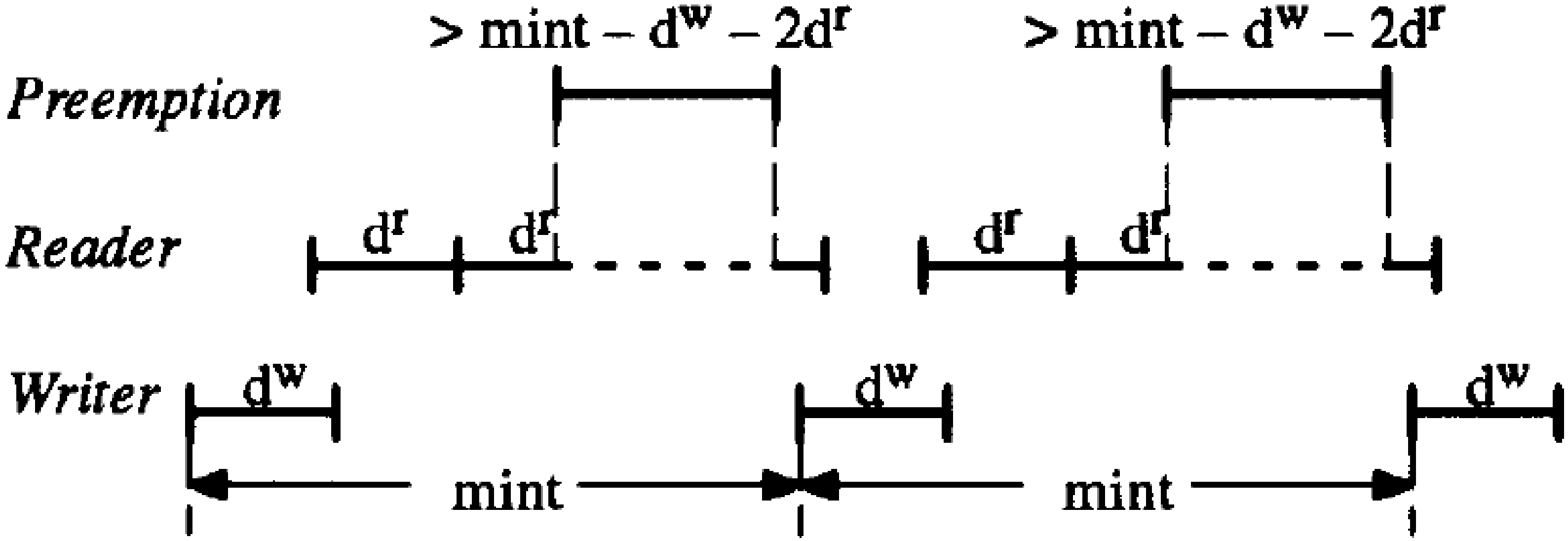
A task with a read operation shares the CPU

- Tasks with higher priority can preempt read
- Can cause more than one write to interfere
- Each write that interferes extends task by $3d^r$
- $c_n = c_o + 3N_i d^r$
- $l_n = l_o - 3N_i d^r$

Schedulability Analysis: Multiple Interferences

- Assume that chosen task scheduling algorithm guarantees all tasks complete before deadline.
- With *mint* known we can bound worst case number of interferences: N_i
- For a read operation to be interfered by a second write:
  - Preempted by an interval of: $mint - d^w - 2d^r$

Schedulability Analysis: Multiple Interferences



Schedulability Analysis: Multiple Interferences

- With this we can bound N_i:

$$N_i = \left\lfloor \frac{l_n}{mint - d^w - 2d^r} \right\rfloor + 1 \leq \left\lfloor \frac{l_o + mint - 3d^{rw}}{mint} \right\rfloor$$

- And therefore execution time bounded:

$$c_n = c_o + 3d^{rw} \left\lfloor \frac{l_o + mint - 3d^{rw}}{mint} \right\rfloor$$

- Use this execution time for each task when testing Schedulability

Example

Message size:	12 bytes (6 words with a size of 16 bits)
read/write time d^{rw}:	10 μsec
Execution time c_o:	3 msec
Deadline d:	10 msec
Laxity l_o:	7 msec
mint:	2 msec

Execution time extension:

$$ete = 3d^{rw} \left\lfloor \frac{l_o + mint - 3d^{rw}}{mint} \right\rfloor =$$

$$= 3 \times 10 \times \left\lfloor \frac{7000 + 2000 - 30}{2000} \right\rfloor =$$

$$= 30 \times 4 = 120 \ \mu sec$$

Execution time extended by .12 msec (4% more than original execution time)

Poor performance

- Same example
- Change d^{rw} *from 10 μsec to 200 μsec*
- Execution extension grows

  from *120 μsec* (4% increase)

  to *2400 μsec* (80% increase)

- Not so good…

Problems with current Protocol

- Not possible to handle tasks with low laxity
- Protocol becomes inefficient when read times become non-negligible compared to total execution time.

Extension to Protocol: Define

- Allocate more than one buffer per message
- Buffers written to cyclically
- Guarantees that reader reads most recent version of message (as of start of read)
- CCF_i used to determine which buffer to use
- Define $bcnt_i$ to be # buffers for message *i*
- Range of CCF_i must be a multiple of $2*bcnt_i$

Extension to Protocol: Read/Write

- Write – write to buffers cyclically
  - Each write increments CCF_i by 2
  - Buffer to write to = $floor(CCF_i / 2) \; mod \; bcnt_i$
- Read – use latest available message
  - $Floor(CCF_i / 2) \; mod \; bcnt_i$ gives current/next write
  - $[Floor(CCF_i / 2)$**-1**$] \; mod \; bcnt_i$ gives last write (that is not currently being written)

Extension to Protocol: Idea

- Cyclic buffer writing => message not overwritten for $bcnt_i$ more messages
- Interference will be much more rare
- Each write causes *CCF* + *2*
- If *CCF* has been incremented $bcnt_i$ *2 times since this message has been written

Extension to the Protocol: Code

Initialization:

```
        CCFᵢ := 0;
```

Write message i:

```
start: CCF_old := CCFᵢ;
       CCFᵢ := CCF_old + 1;
       <write bufᵢ [ ⌊CCF_old/2⌋ mod bcntᵢ ]>
       CCFᵢ := CCF_old + 2;
```

Read message i:

```
start: CCF_begin := CCFᵢ;
       <read bufᵢ [ (⌊CCF_begin/2⌋ - 1)
        mod bcntᵢ ]>
       CCF_end:= CCFᵢ;
*      if CCF_end < CCF_begin
          then CCF_end = CCF_end + Rᵢ;
       If CCF_end - ⌊CCF_begin/2⌋ * 2 >
          bcntᵢ * 2 - 2
          then goto start;
```

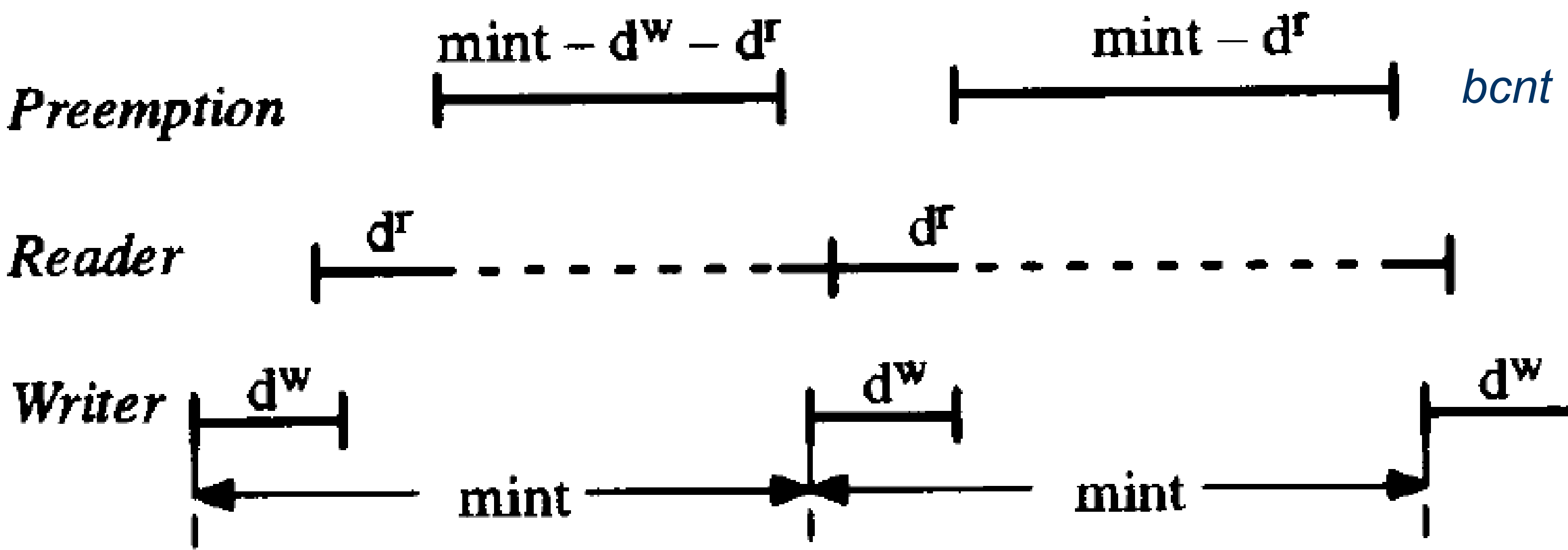
The line marked with a * is needed because of the limited range of R_i.

Extension to the Protocol: Schedulability Analysis

- As long as *bnct* > *1*:
  - $c_n = c_o + 1 N_i d^r$
  - $l_n = l_o - 1 N_i d^r$
- Interference: (*bcnt* = *2*)
  1. Write almost finished to buffer 2
  2. Read starts from buffer 1 (most recent to read)
  3. Write completes to buffer 2
  4. Read is delayed
  5. Write to buffer 1 begins (wrapping to beginning)
  6. Read attempt of buffer 1 ends (corrupted)
- If *bcnt* > 2 then *bcnt-2* extra writes at #4

Extension to the Protocol: Schedulability Analysis: read delay

- To cause interference, read:
  - Must last: $(bcnt - 1) * mint - d^w$
  - Preemption must last: $(bcnt - 1) * mint - d^w - d^r$
  - retry must be preempted by: $(bcnt - 1)*mint - d^r$



bcnt =2

Schedulability Analysis: Multiple Interferences

- With this we can bound:

$$N_i \leq \left\lfloor \frac{l_o + d^w}{(bcnt - 1)mint} \right\rfloor$$

- And therefore execution time bounded:

$$c_n = c_o + d^r \left\lfloor \frac{l_o + d^w}{(bcnt - 1)mint} \right\rfloor$$

- Use this execution time for each task when testing Schedulability
- Also check *2*bcnt* N_i < *R* (range of CCF)

Example

- Same example except $d^{rw} = 200$
  - First protocol: process extension = *2400 µsec* (80% increase in execution time)

But now $$ete = d^r \left\lfloor \frac{l_o + d^w}{(bcnt - 1)mint} \right\rfloor = 200 \times \left\lfloor \frac{7000 + 200}{1 \times 2000} \right\rfloor$$

$$= 200 \times 3 = 600 \ \mu\text{sec}$$

for $bcnt = 2$ and to

$$ete = d^r \left\lfloor \frac{l_o + d^w}{(bcnt - 1)mint} \right\rfloor = 200 \times \left\lfloor \frac{7000 + 200}{4 \times 2000} \right\rfloor$$

$$= 200 \times 0 = 0 \ \mu\text{sec}$$

for $bcnt = 5$.

Questions?