Priority Inheritance Protocol Proved Correct

Xingyuan Zhang
PLA University of Science and Technology
Nanjing, China

joint work with
Christian Urban
Kings College, University of London, U.K.
Chunhan Wu
My Ph.D. student now working for Christian

Prioirty Inheritance Protocol (PIP)

- Widely used in Real-Time OSs

Prioirty Inheritance Protocol (PIP)

- Widely used in Real-Time OSs
- One solution of 'Priority Inversion' problem

Prioirty Inheritance Protocol (PIP)

- Widely used in Real-Time OSs
- One solution of 'Priority Inversion' problem
- A flawed manual correctness proof (1990)

Prioirty Inheritance Protocol (PIP)

- Widely used in Real-Time OSs
- One solution of ‘Priority Inversion’ problem
- A flawed manual correctness proof (1990)
  - Notations with no precise definition
  - Resorts to intuitions

Prioirty Inheritance Protocol (PIP)

- Widely used in Real-Time OSs
- One solution of 'Priority Inversion' problem
- A flawed manual correctness proof (1990)
  - Notations with no precise definition
  - Resorts to intuitions
- Formal treatments using model-checking

Prioirty Inheritance Protocol (PIP)

- Widely used in Real-Time OSs
- One solution of ‘Priority Inversion’ problem
- A flawed manual correctness proof (1990)
  - Notations with no precise definition
  - Resorts to intuitions
- Formal treatments using model-checking
  - Applicable to small size system models
  - Unhelpful for human understanding

Prioirty Inheritance Protocol (PIP)

- Widely used in Real-Time OSs
- One solution of 'Priority Inversion' problem
- A flawed manual correctness proof (1990)
  - Notations with no precise definition
  - Resorts to intuitions
- Formal treatments using model-checking
  - Applicable to small size system models
  - Unhelpful for human understanding
- Verification of PCP in PVS (2000)

Prioirty Inheritance Protocol (PIP)

- Widely used in Real-Time OSs
- One solution of 'Priority Inversion' problem
- A flawed manual correctness proof (1990)
  - Notations with no precise definition
  - Resorts to intuitions
- Formal treatments using model-checking
  - Applicable to small size system models
  - Unhelpful for human understanding
- Verification of PCP in PVS (2000)
  - A related protocol
  - Priority Ceiling Protocol

Our Motivation

- Undergraduate OS course in our university

Our Motivation

- Undergraduate OS course in our university
  - Experiments using intrutional OSs
  - PINTOS (Stanford) is choosen
  - Core project: Implementing PIP in it

Our Motivation

- Undergraduate OS course in our university
  - Experiments using intrutional OSs
  - PINTOS (Stanford) is choosen
  - Core project: Implementing PIP in it
- Understanding is crucial to implemention

Our Motivation

- Undergraduate OS course in our university
  - Experiments using intrutional OSs
  - PINTOS (Stanford) is choosen
  - Core project: Implementing PIP in it
- Understanding is crucial to implemention
- Little help was found in the literature

Our Motivation

- Undergraduate OS course in our university
  - Experiments using intrutional OSs
  - PINTOS (Stanford) is choosen
  - Core project: Implementing PIP in it
- Understanding is crucial to implemention
- Little help was found in the literature
- Some mentioning the complication

Some excerpts

"Priority inheritance is neither efficient nor reliable. Implementations are either incomplete (and unreliable) or surprisingly complex and intrusive."

Some excerpts

"Priority inheritance is neither efficient nor reliable. Implementations are either incomplete (and unreliable) or surprisingly complex and intrusive."

"I observed in the kernel code (to my disgust), the Linux PIP implementation is a nightmare: extremely heavy weight, involving maintenance of a full wait-for graph, and requiring updates for a range of events, including priority changes and interruptions of wait operations."

Our Aims

- Formal specification at appropriate abstract level, convenient for:
  - Constructing interactive proofs
  - Clarifying the underlying ideas

Our Aims

- Formal specification at appropriate abstract level, convenient for:
  - Constructing interactive proofs
  - Clarifying the underlying ideas
- Theorems usable to guide implementation, critical point:
  - Understanding the relationship with real OS code

Our Aims

- Formal specification at appropriate abstract level, convenient for:
  - Constructing interactive proofs
  - Clarifying the underlying ideas
- Theorems usable to guide implementation, critical point:
  - Understanding the relationship with real OS code
  - Not yet formalized

Real-Time OSes

- Purpose: gurantee the most urgent task be processed in time
- Processes have priorities
- Resources can be locked and unlocked

Problem

High-priority process

Low-priority process

Problem

High-priority process

Medium-priority process

Low-priority process

Problem

High-priority process

Medium-priority process

Low-priority process

- Priority Inversion $\stackrel{\text{def}}{=}$ H < L

Problem

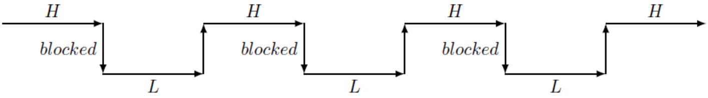
High-priority process

Medium-priority process

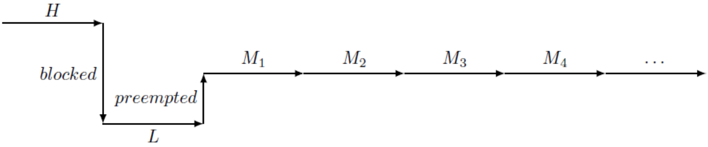
Low-priority process

- Priority Inversion $\stackrel{\text{def}}{=}$ H < L
- avoid indefinite priority inversion

Priority Inversion

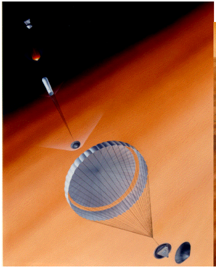
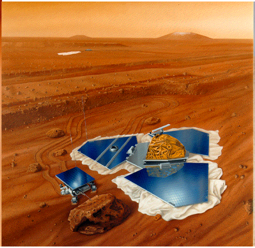
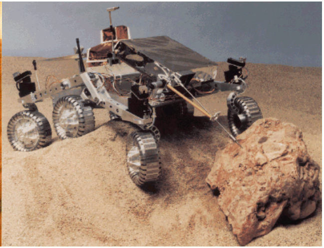


Prioirty Inversion



Indefinite Priority Inversion

Mars Pathfinder Mission 1997



Solution

Priority Inheritance Protocol (PIP):

High-priority process

Medium-priority process

Low-priority process

(temporarily raise its priority)

A Correctness "Proof" in 1990

- a paper⋆ in 1990 "proved" the correctness of an algorithm for PIP

  ... after the thread (whose priority has been raised) completes its critical section and releases the lock, it "returns to its original priority level".

⋆ in IEEE Transactions on Computers

High-priority process 1

High-priority process 2

Low-priority process

High-priority process 1

High-priority process 2

Low-priority process

- Solution:
  Return to highest remaining priority

Event Abstraction

- Use Inductive Approch of L. Paulson

Event Abstraction

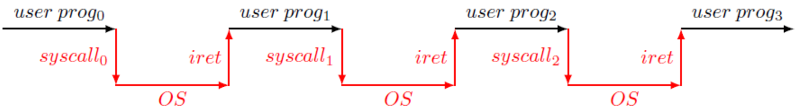
- Use Inductive Approch of L. Paulson
- System is event-driven

Event Abstraction

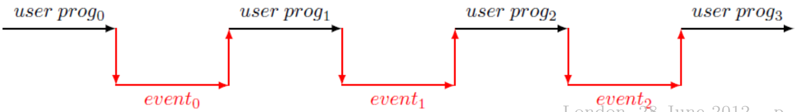
- Use Inductive Approch of L. Paulson
- System is event-driven
- A state is a list of events

Event Abstraction

- Use Inductive Approch of L. Paulson
- System is event-driven
- A state is a list of events



Execution of OS



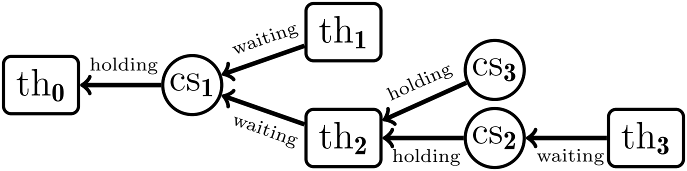
Events

Create thread priority
Exit thread
Set thread priority
Lock thread cs
Unlock thread cs

Precedences

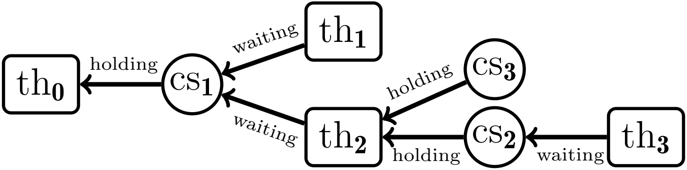
$$\text{prec th s} \stackrel{\text{def}}{=} (\text{priority th s}, \text{last\_set th s})$$

RAGs



RAG wq $\stackrel{\text{def}}{=}$ {(T th, C cs) | waits wq th cs} ∪ {(C cs, T th) | holds wq th cs}

RAGs



$$\text{RAG wq} \stackrel{\text{def}}{=} \{(\text{T th, C cs}) \mid \text{waits wq th cs}\} \cup \{(\text{C cs, T th}) \mid \text{holds wq th cs}\}$$

Good Next Events

$$\frac{\text{th} \notin \text{threads s}}{\text{step s (Create th prio)}}$$

$$\frac{\text{th} \in \text{running s} \quad \text{resources s th} = \varnothing}{\text{step s (Exit th)}}$$

$$\frac{\text{th} \in \text{running s}}{\text{step s (Set th prio)}}$$

Good Next Events

$$\frac{\text{th} \in \text{running s} \qquad (\text{C cs, T th}) \notin (\text{RAG s})^+}{\text{step s (P th cs)}}$$

$$\frac{\text{th} \in \text{running s} \qquad \text{holds s th cs}}{\text{step s (V th cs)}}$$

Theorem: “No indefinite priority inversion”

Theorem: "No indefinite priority inversion"

Theorem ⋆: If th is the thread with the highest precedence in state s:

Theorem: “No indefinite priority inversion”

Theorem ⋆: If th is the thread with the highest precedence in state s:

$$\text{prec th s} = \text{Max (cprec s ` threads s))}$$

Theorem: "No indefinite priority inversion"

Theorem ⋆: If th is the thread with the highest precedence in state s:

$$\text{prec th s} = \text{Max (cprec s ` threads s))}$$

and th is blocked by a thread th' in a future state s' (with s' = t@s):

Theorem: “No indefinite priority inversion”

Theorem *: If th is the thread with the highest precedence in state s:

$$\text{prec th s} = \text{Max (cprec s ` threads s))}$$

and th is blocked by a thread th’ in a future state s’ (with s’ = t@s):

$$\text{th' } \in \text{ running (t@s) and th' } \neq \text{ th}$$

Theorem: “No indefinite priority inversion”

Theorem $\star$: If th is the thread with the highest precedence in state s:

$$\text{prec th s} = \text{Max (cprec s ` threads s))}$$

and th is blocked by a thread th’ in a future state s’ (with s’ = t@s):

$$\text{th' } \in \text{ running (t@s) and th' } \neq \text{ th}$$

Theorem: “No indefinite priority inversion”

Theorem *: If th is the thread with the highest precedence in state s:

$$\text{prec th s} = \text{Max (cprec s ` threads s))}$$

and th is blocked by a thread th’ in a future state s’ (with s’ = t@s):

$$\text{th'} \in \text{running (t@s) and th'} \neq \text{th}$$

- th’ did not hold or wait for a resource in s:

$$\neg\text{detached s th'}$$

Theorem: "No indefinite priority inversion"

Theorem *: If th is the thread with the highest precedence in state s:

$$\text{prec th s} = \text{Max (cprec s ` threads s))}$$

and th is blocked by a thread th' in a future state s' (with s' = t@s):

$$\text{th'} \in \text{running (t@s) and th'} \neq \text{th}$$

- th' did not hold or wait for a resource in s:

  $$\neg\text{detached s th'}$$

- th' is running with the precedence of th:

  $$\text{cp (t@s) th'} = \text{preced th s}$$

Theorem: "No indefinite priority inversion"

Theorem ⋆: If th is the thread with the highest precedence in state s:

$$\text{prec th s} = \text{Max (cprec s ` threads s))}$$

and th is blocked by a thread th' in a future state s' (with s' = t@s):

$$\text{th'} \in \text{running (t@s) and th'} \neq \text{th}$$

- th' did not hold or wait for a resource in s:

  $$\neg\text{detached s th'}$$

- th' is running with the precedence of th:

  $$\text{cp (t@s) th'} = \text{preced th s}$$

⋆ modulo some further assumptions

Theorem: "No indefinite priority inversion"

Theorem ⋆: If th is the thread with the highest precedence in state s:

$$\text{prec th s} = \text{Max (cprec s ` threads s))}$$

and th is blocked by a thread th' in a future state s' (with s' = t@s):

$$\text{th'} \in \text{running (t@s) and th'} \neq \text{th}$$

- th' did not hold or wait for a resource in s:

  $$\neg\text{detached s th'}$$

- th' is running with the precedence of th:

  $$\text{cp (t@s) th'} = \text{preced th s}$$

⋆ modulo some further assumptions

It does not matter which process gets a released lock.

Implementation

s = current state; s' = next state = e#s

When e = Create th prio, Exit th

- RAG s' = RAG s
- No precedence needs to recalculate

Implementation

s = current state; s' = next state = e#s

When e = Set th prio

- RAG s' = RAG s
- No precedence needs to recalculate

Implementation

s = current state; s' = next state = e#s

When e = Unlock th cs where there is a thread to take over

- RAG s' = RAG s - {(C cs, T th), (T th', C cs)} $\cup$ {(C cs, T th')}
- we have to recalculate the precedence of the direct descendants

Implementation

s = current state; s' = next state = e#s

When e = Unlock th cs where there is a thread to take over

- RAG s' = RAG s - {(C cs, T th), (T th', C cs)} ∪ {(C cs, T th')}
- we have to recalculate the precedence of the direct descendants

When e = Unlock th cs where no thread takes over

- RAG s' = RAG s - {(C cs, T th)}
- no recalculation of precedences

Implementation

s = current state; s' = next state = e#s

When e = Lock th cs where cs is not locked

- RAG s' = RAG s $\cup$ {(C cs, T th')}
- no recalculation of precedences

Implementation

s = current state; s' = next state = e#s

When e = Lock th cs where cs is not locked

- RAG s' = RAG s ∪ {(C cs, T th')}
- no recalculation of precedences

When e = Lock th cs where cs is locked

- RAG s' = RAG s - {(T th, C cs)}
- we have to recalculate the precedence of the descendants

Conclusion

- Aims fulfilled

Conclusion

- Aims fulfilled
- Alternative way

Conclusion

- Aims fulfilled
- Alternative way
  - using Isabelle/HOL in OS code development
  - through the Inductive Approach

Conclusion

- Aims fulfilled
- Alternative way
  - using Isabelle/HOL in OS code development
  - through the Inductive Approach
- Future researches

Conclusion

- Aims fulfilled
- Alternative way
  - using Isabelle/HOL in OS code development
  - through the Inductive Approach
- Future researches
  - scheduler in RT-Linux
  - multiprocessor case
  - other “nails” ? (networks, ...)

Conclusion

- Aims fulfilled
- Alternative way
  - using Isabelle/HOL in OS code development
  - through the Inductive Approach
- Future researches
  - scheduler in RT-Linux
  - multiprocessor case
  - other “nails” ? (networks, . . .)
  - Refinement to real code and relation between implemenations