FIU
FLORIDA INTERNATIONAL UNIVERSITY
Miami's public research university

Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism

Thomas E. Anderson, Brian N. Bershad, Edward D. Lazowska, Henry M. Levy

Presenters: Yexi Jiang, Jorge Arenas, Hao Jiang

FIU | FLORIDA INTERNATIONAL UNIVERSITY

The Authors




- Thomas E. Anderson . Robert E. Dinning Professor.
  - constructing robust, secure, and efficient computer systems
- Brian N. Bershad. Professor.
  - computer operating systems, distributed systems and architecture
- Edward D. Lazowska. Bill & Melinda Gates Chair in Computer Science & Engineering.
  - the design, implementation, and analysis of high performance computing and communication systems
- Henry M. Levy. Chairman and Wissner-Slivka Chair
  - operating systems, distributed systems, security, the world-wide web, and computer architecture

FIU FLORIDA INTERNATIONAL UNIVERSITY

Outline

- Background
- Problem Statement
- The Approach
- Related Works
- User-level Thread
- Kernel-level Thread
- Scheduler Activation
- Implementation
- Experimental Evaluation
- Summary

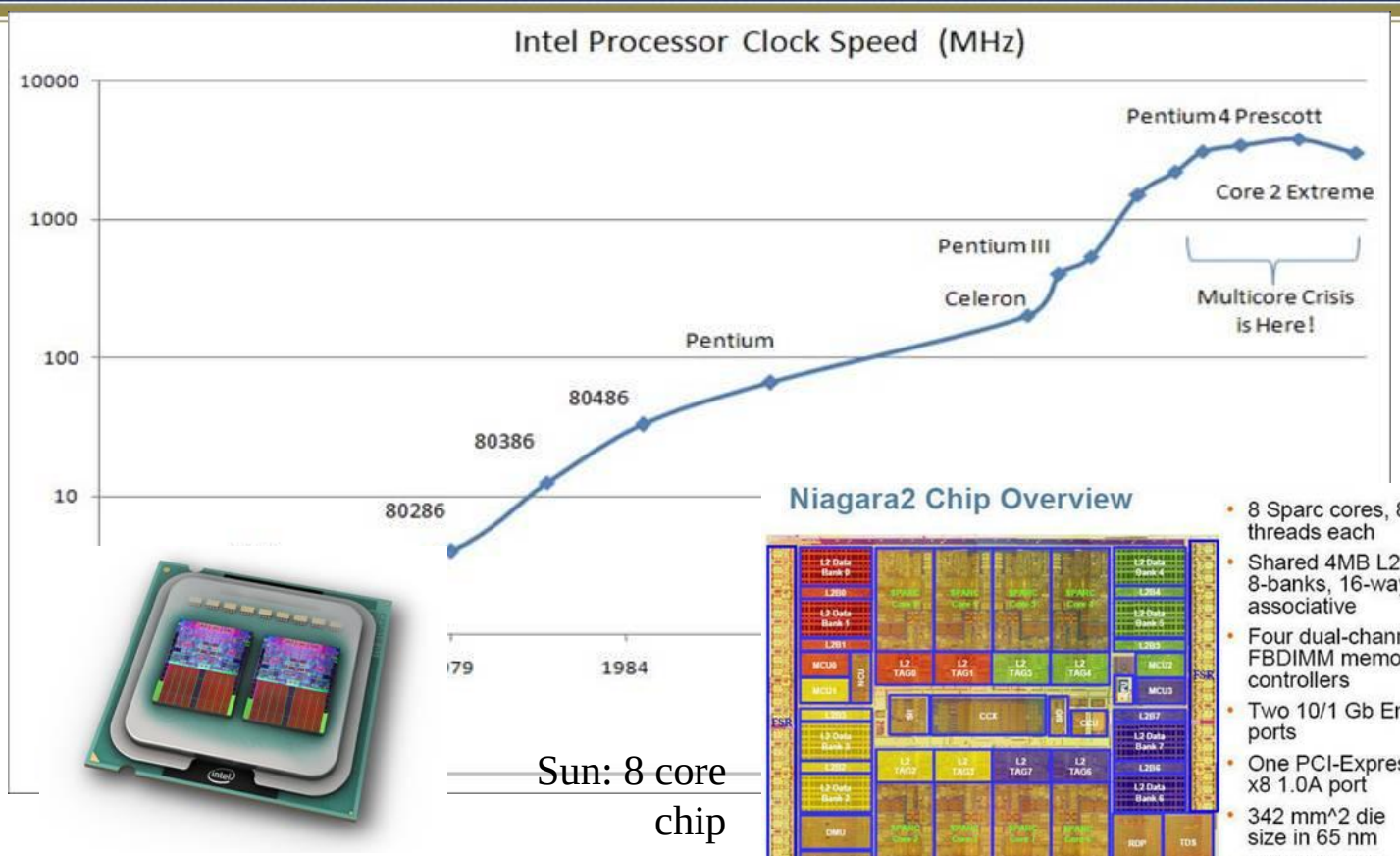
FIU | FLORIDA INTERNATIONAL UNIVERSITY

Background

- Technologies Trend.
- Process vs. Thread, and Why Thread?
- User Mode vs. Kernel Mode
- User-Level Thread vs. Kernel-Level Thread

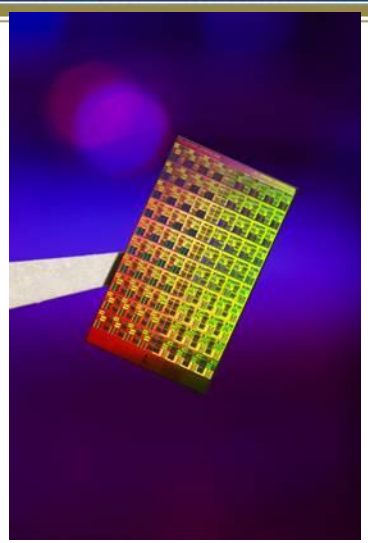
FIU | FLORIDA INTERNATIONAL UNIVERSITY

Technologies Driving Multi-Programming



Intel: Quad Core

Sun: 8 core chip



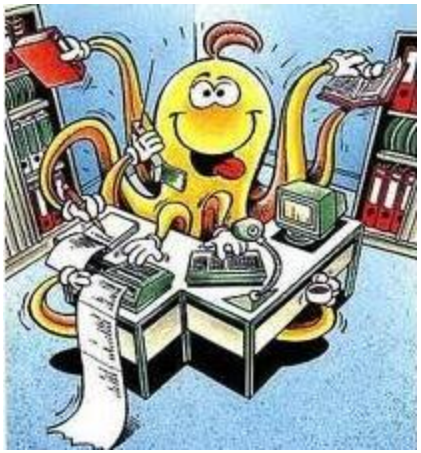
Intel: 80 core experimental system

FIU FLORIDA INTERNATIONAL UNIVERSITY

Reasons for Multi-Programming

performance

multitasking



parallelism

coordination

FIU FLORIDA INTERNATIONAL UNIVERSITY

Process vs. Thread

- Process is a program in execution, thread is a light-weight process.

	Process	Thread
Data Structure	PCB	TCB
Communication among processes/threads	Message, Shared Memory	Inherent Shared Memory
Time overhead	High	Low
Protection among processes/threads	Yes	No

Why Thread? An Example

- How to develop an editor that allow typo checking, document printing, and user input simultaneously?

Process based	Thread based
1 process for typo checking, 1 process for printing, 1 process for handling user input. **Explicitly leverage inter-process communication. Complicate!**	1 thread for typo checking, 1 thread for printing, 1 thread for handling user input. **Use shared data structure.**

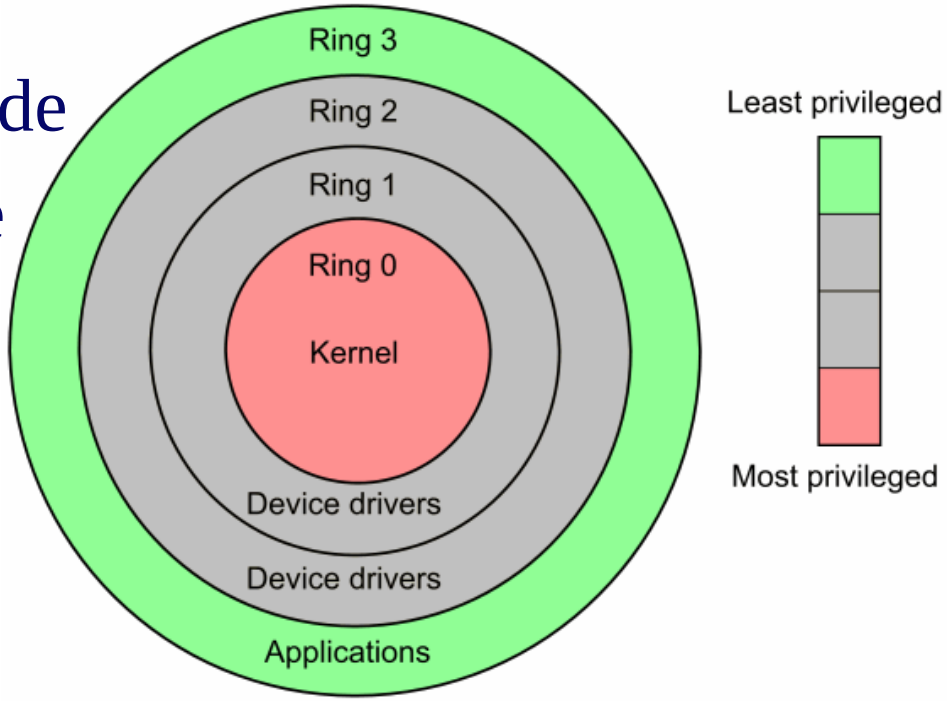
User Mode vs. Kernel Mode

- **Kernel Mode**
  - Unrestricted access to hardware.
  - Execute trusted functions.
  - Crash is severe.
- **User Mode**
  - No ability to directly access hardware.
  - Crash is OK.

FIU | FLORIDA INTERNATIONAL UNIVERSITY

User Mode vs. Kernel Mode

- X86 architecture
  - Ring 0: kernel mode
  - Ring 3: user mode
  - Ring 1,2: driver?



FIU | FLORIDA INTERNATIONAL UNIVERSITY

User-Level Threads vs. Kernel-Level Threads

- **User-level threads:** Build without any modifications to the underlying OS. Running on top of the system in user mode.
- **Kernel-level threads:** Build in kernel. Running directly in kernel mode.

FIU | FLORIDA INTERNATIONAL UNIVERSITY

User-Level Threads vs. Kernel-Level Threads

	User-level threads	Kernel-level threads
Pros	1. No modification for kernel 2. Low overhead	Full-knowledge of all threads.
Cons	1. Threads in a process is invisible to the OS. 2. Block one thread would block whole process. 3. Access to multiple processors is not guaranteed.	1. High overhead. 2. Synchronization is more expensive.

FIU | FLORIDA INTERNATIONAL UNIVERSITY

User-Level Threads vs. Kernel-Level Threads

- This paper is one of the pioneers that takes advantages from both user-level and kernel-level threads.

- Models: One-to-one, many-to-one, many-to-many, two-level. (See details later)

FIU FLORIDA INTERNATIONAL UNIVERSITY

Outline

- Background
- Problem Statement
- The Approach
- Related Works
- User-level Thread
- Kernel-level Thread
- Scheduler Activation
- Implementation
- Experimental Evaluation
- Summary

Problem Statement

- Existing implementations
  - User-level choices: *PCR*, *FastThreads*
  - Kernel-level choices: threads support in *Mach*, *Topaz*, *V*

Dilemma of programmer

- Choose user-level threads
  - High performance.
  - Exhibit poor performance or incorrect behavior in presence of I/O, page faults, preemption.
- Choose kernel-level threads
  - No system integration problem. Threads are directly scheduled by kernel.
  - Low performance.

Neither is good enough!

FIU | FLORIDA INTERNATIONAL UNIVERSITY

What is an ideal design?

- Functionality
  - No idled process when ready threads exist.
  - No high-priority threads wait for low-priority threads.
  - Block a thread would not block whole process
- Performance
  - As fast as user-level threads
- Flexibility
  - User-level part enables application-specific customization.

FIU FLORIDA INTERNATIONAL UNIVERSITY

Outline

- Background
- Problem Statement
- The Approach
- Related Works
- User-level Thread
- Kernel-level Thread
- Scheduler Activation
- Implementation
- Experimental Evaluation
- Summary

FIU | FLORIDA INTERNATIONAL UNIVERSITY

The Approach

- A hybrid threads design.
  - Threads are implemented in both user-level and kernel-level . Only transmit necessary information between user-level and kernel-level.
- Each application:
  - Provided with a virtual multiprocessor.
  - Has full knowledge about the allocated processors.
  - Can control which threads run on which processor.

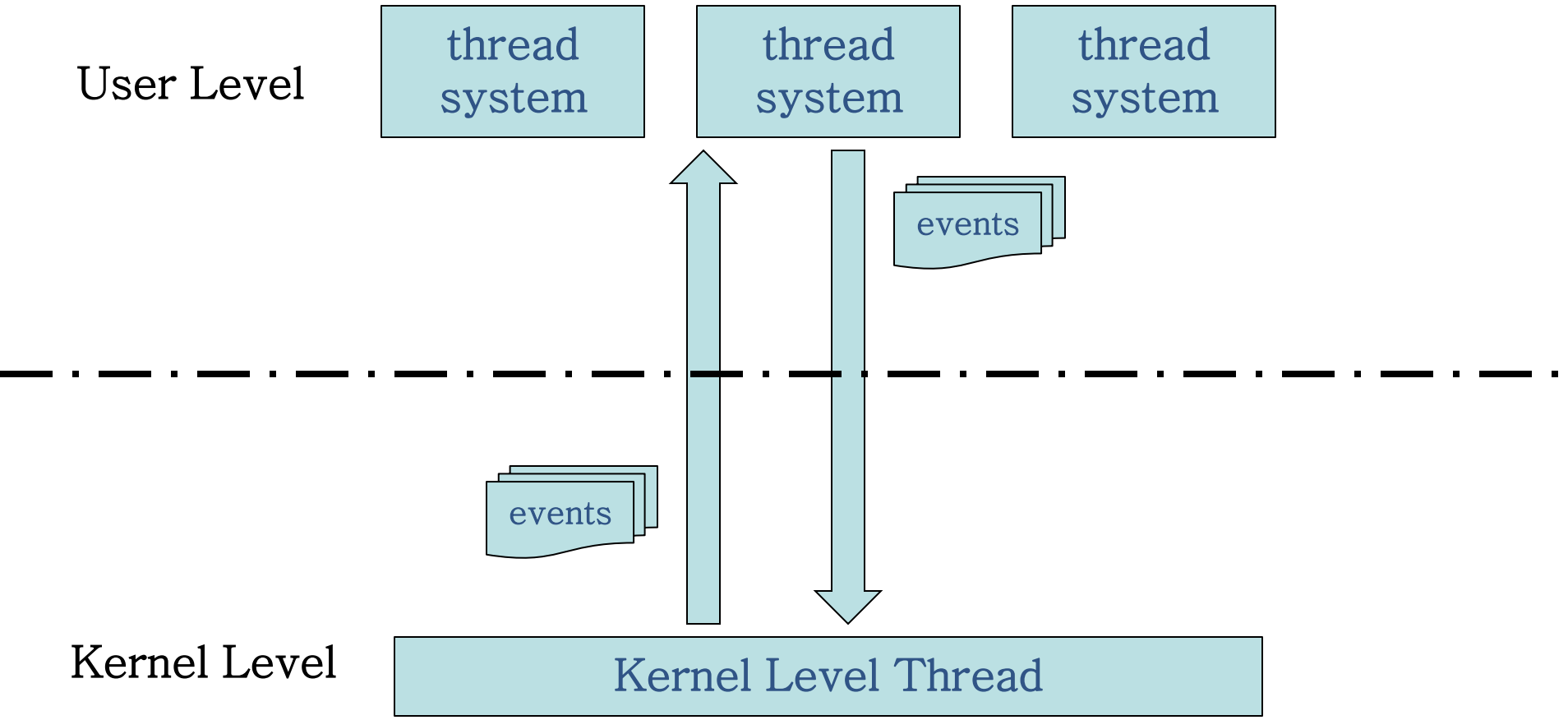
The Approach

- Communicating upward
  - Improve functionality
- Communicating downward
  - Preserve performance

Scheduler activations: The execution context for an event vectored from the kernel to an address space.

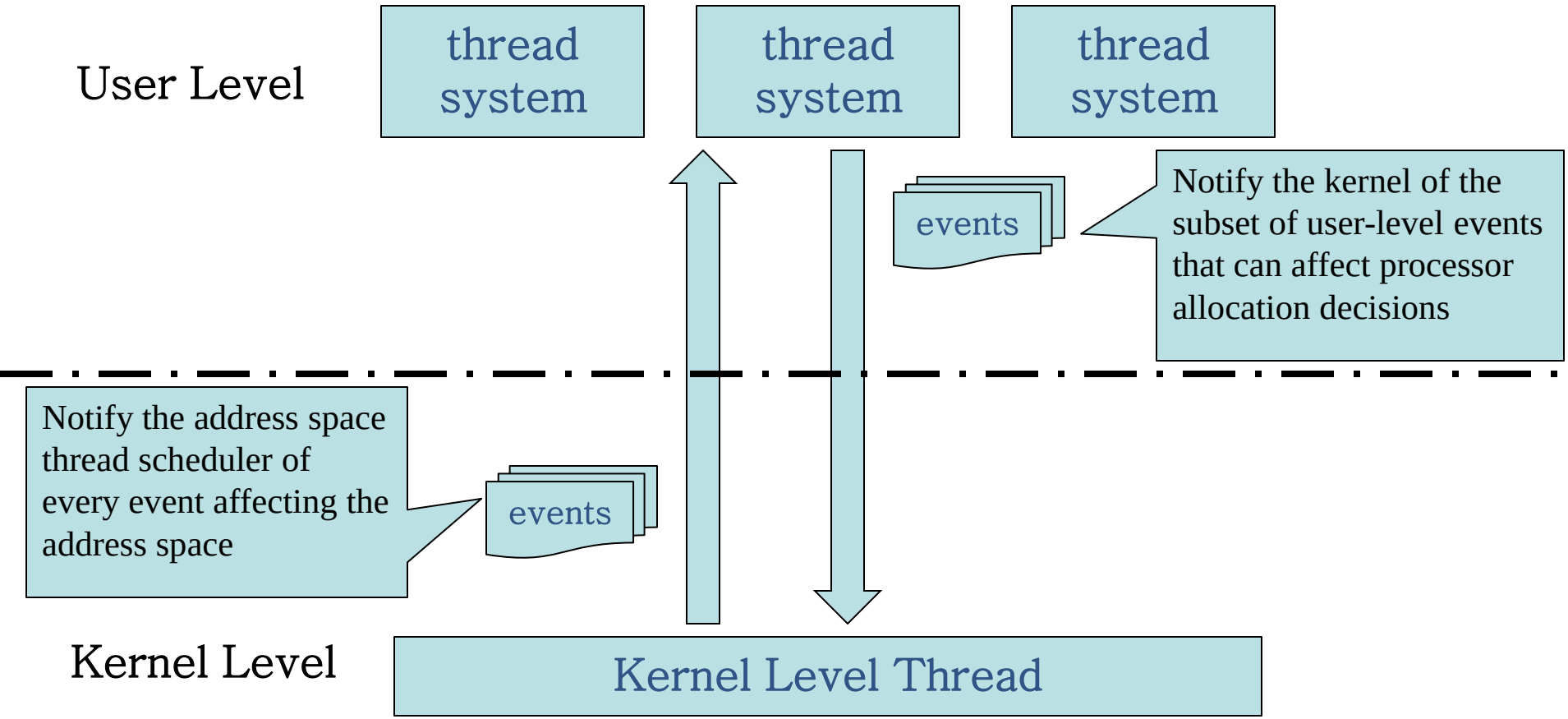
FLORIDA INTERNATIONAL UNIVERSITY

The Approach



FIU | FLORIDA INTERNATIONAL UNIVERSITY

The Approach



FIU FLORIDA INTERNATIONAL UNIVERSITY

Outline

- Background
- Problem Statement
- The Approach
- Related Works
- User-level Thread
- Kernel-level Thread
- Scheduler Activation
- Implementation
- Experimental Evaluation
- Summary

Related Works

- Psyche and Symunix

	Scheduler Activations	Psyche and Symunix
Platform	Topaz (UNIX)	Psyche: UNIX Symunix: New OS
Communication	Between user-level and kernel-level	Between kernel and application
Synchronization	Low overhead	High overhead, Should be handled between applications and kernel

Related Works

- Compare with traditional asynchronous I/O system.
  - Uniformly addresses preemption, I/O, and page faults.
  - No need to major change the structure for application and kernel code.

Related Works

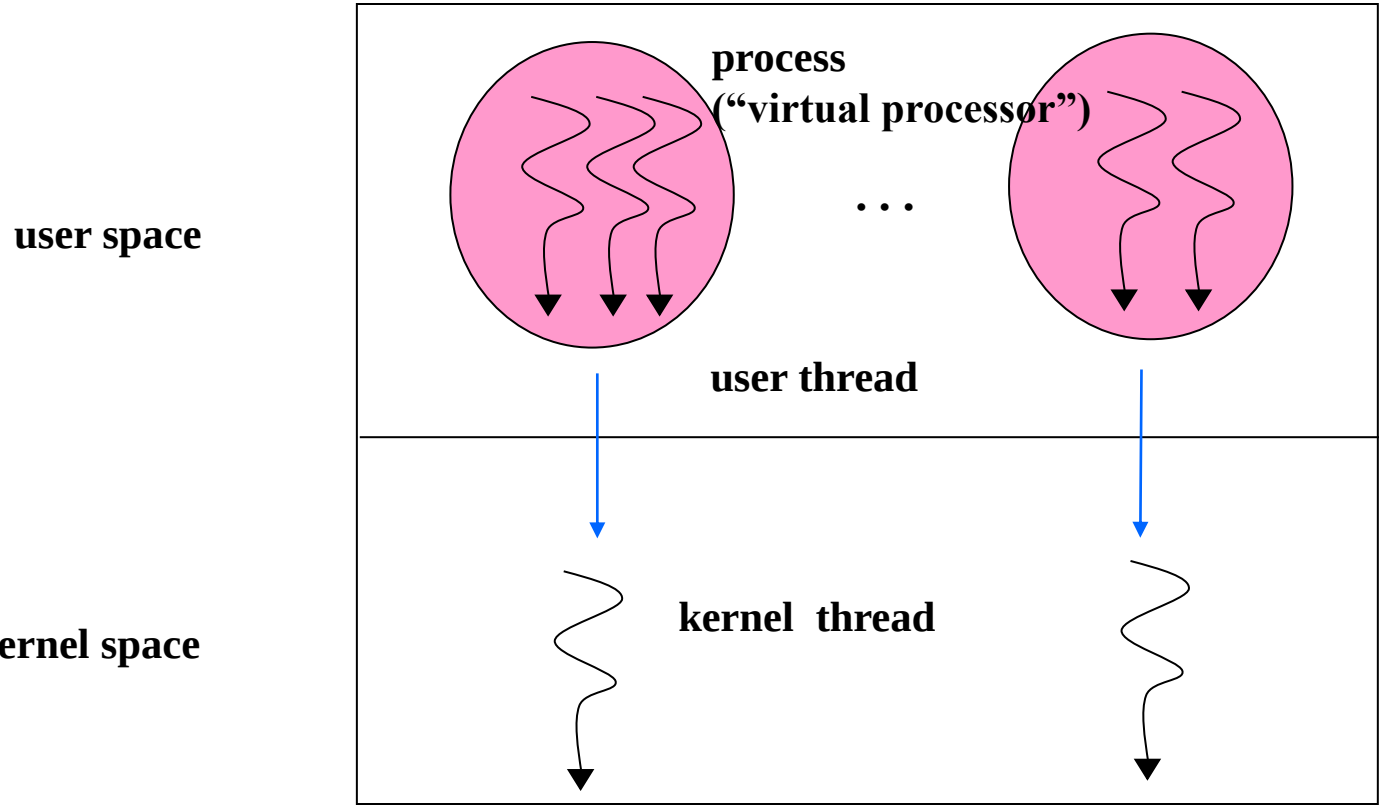
- Compare with Hydra
  - Both implement scheduling policy out of kernel.
  - Hydra: scheduling requires coordinating between scheduler server and kernel. Scheduler Activation: Scheduling is autonomic.

FIU FLORIDA INTERNATIONAL UNIVERSITY

Outline

- Background
- Problem Statement
- The Approach
- Related Works
- User-level Thread
- Kernel-level Thread
- Scheduler Activation
- Implementation
- Experimental Evaluation
- Summary

User-level Thread



User-level Thread

- **Divide up the process resources between threads.**
  - N:1 arrangement, with N user threads sharing one kernel thread.
  - Managed by runtime library routines.
  - Require no kernel intervention.
- **E.g.: GNU PTH Thread Library, FSU Threads.**
- **The thread package views each process as a virtual processor.**
  - Each virtual processor runs user-level code that pulls threads off the ready queue and runs them.

Advantages of User-level Threads

- **Occupy the user address space of the process.**
  - No overhead of kernel intervention when switching between threads.
  - Not much more expensive than a regular procedure call.
- **Flexible and amenable to specialization.**
  - Only need to meet the requirements of the specific applications that will use them.
    - If priority scheduling is not needed, for example, leave it out.

Disadvantages of User-level Threads

- **There is a lack of kernel support in existing systems.**
  - A process gets the same amount of time regardless of how many threads it is running.
  - System is not able to make use of multiprocessors.
- **Kernel threads block, resume, and are preempted without the user level threads being notified.**
  - Page faults, I/O can block the whole process.
    - Poor performance or cause errors.

Example: I/O

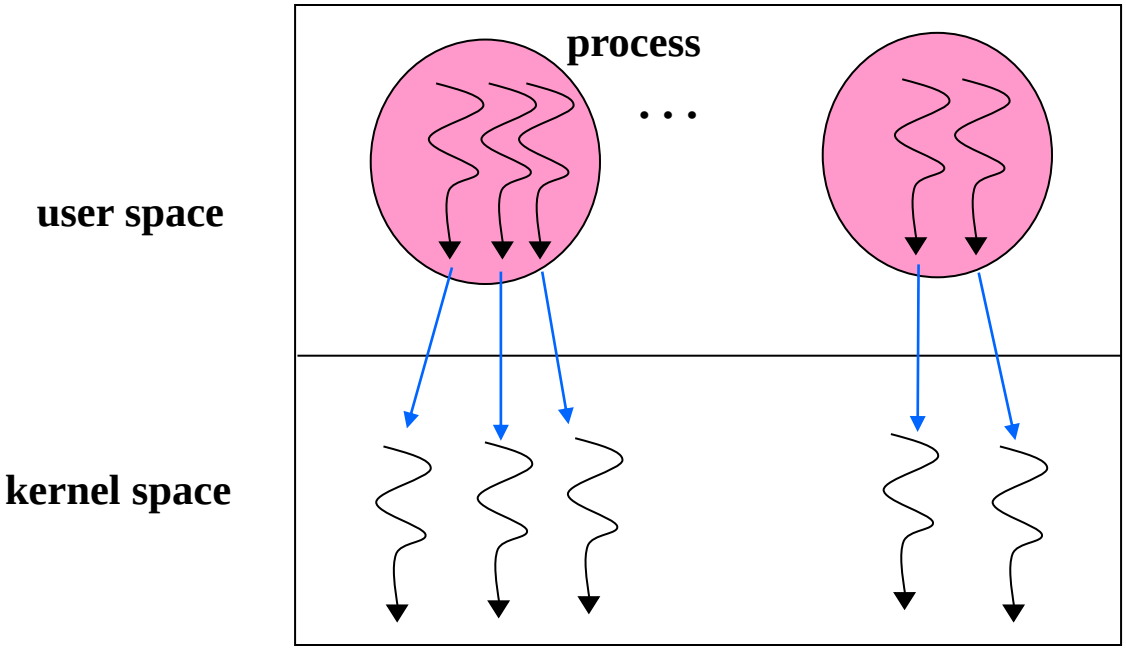
- **One application running five user-level threads.**
  - Performs inexpensive application-level context switches between threads.
- **Thread3 performs a system call for a file read().**
  - Kernel blocks the whole process, including the remaining four threads that are waiting to run.
- **Solution?**
  - Application intercepts system calls and substitutes an asynchronous (nonblocking) call.
    - An asynchronous interface call appears to return control to the caller immediately. Results will arrive later.
    - The remaining threads can now be run while thread3 is blocked.
  - Page faults cannot be predicted and mitigated, however, and will also result in the whole process being blocked.

FIU FLORIDA INTERNATIONAL UNIVERSITY

Outline

- Background
- Problem Statement
- The Approach
- Related Works
- User-level Thread
- Kernel-level Thread
- Scheduler Activation
- Implementation
- Experimental Evaluation
- Summary

Kernel-level Thread



Kernel-level Thread

- **Kernel threads directly schedule each processes threads onto physical processors.**
  - 1:1 arrangement, each application threads maps to one kernel thread.
  - Enjoy direct OS support, so lack the problems that hinder user-level thread reliability.
- **E.g.: Linux, IRIX, Windows NT.**
- **Too much overhead to use in many parallel programs.**
  - Performance isn't as good as that of user threads.

FIU | FLORIDA INTERNATIONAL UNIVERSITY

Advantages, Disadvantages of Kernel-level Threads

- **Kernel is aware of the threads.**
  - Scheduler can assign time to processes based on the number of threads they carry.
  - I/O, page faults, and interrupts are not a problem.
  - Supports the needs of any application that runs it.
- **Hundreds of times slower than user-level threads.**
  - Kernel threads use system calls to perform operations.
    - Switching between threads requires a full context switch.
  - Kernel threads are heavyweights, must support every need of an application.
  - Each kernel thread consumes kernel memory.

FIU FLORIDA INTERNATIONAL UNIVERSITY

Outline

- Background
- Problem Statement
- The Approach
- Related Works
- User-level Thread
- Kernel-level Thread
- Scheduler Activation
- Implementation
- Experimental Evaluation
- Summary

N:M Thread System

Goal: System with the functionality of kernel threads and the performance and flexibility of user-level threads.

- Maps some no. N of application threads onto a no. M of kernel entities (usually N>M).
- N:M thread systems are more complicated and effective
- Implemented by associating groups of threads with single kernel entities.
- Both AIX and Solaris use N:M thread system.

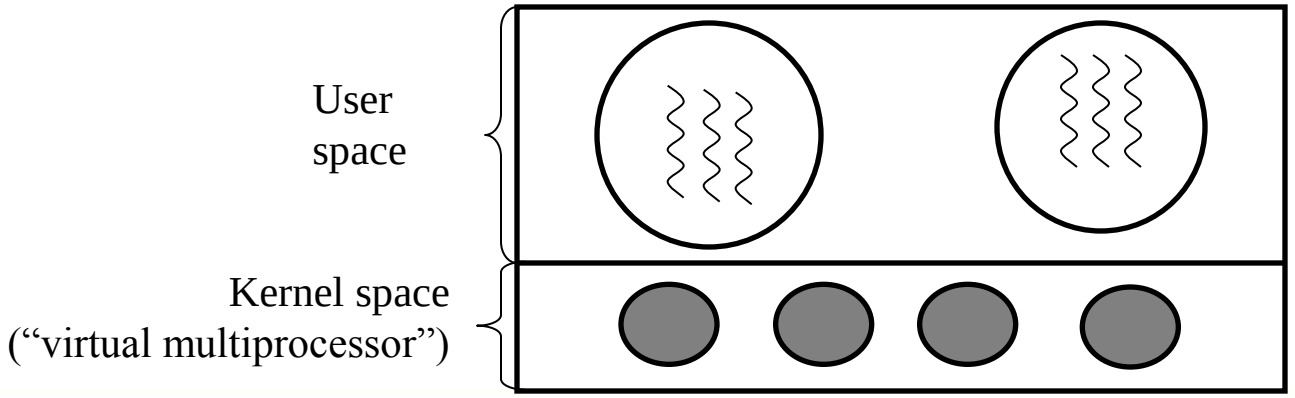
Problem

- Application has knowledge of the user-level thread state but has little knowledge of or influence over critical kernel-level events (by design! to achieve the virtual machine abstraction)
- Kernel has inadequate knowledge of user-level thread state to make optimal scheduling decisions

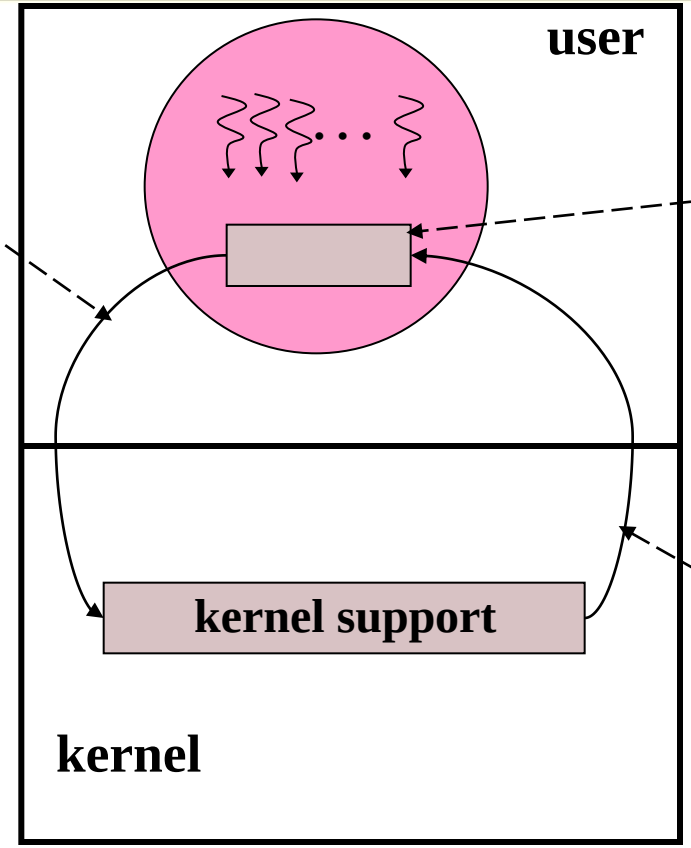
Solution: a mechanism that facilitates exchange of information between user-level and kernel-level mechanisms.

Virtual Multiprocessor

- **Application knows how many and which processors allocated to it by kernel.**
- **Application has complete control over which threads are running on processors.**
- **Kernel notifies thread scheduler of events affecting address space.**
- **Thread scheduler notifies kernel regarding processor allocation.**



Scheduler Activations: Structure



- Change in processor requirements

thread library

Scheduler activations

- change in processor allocation
- change in thread status

Scheduler Activations

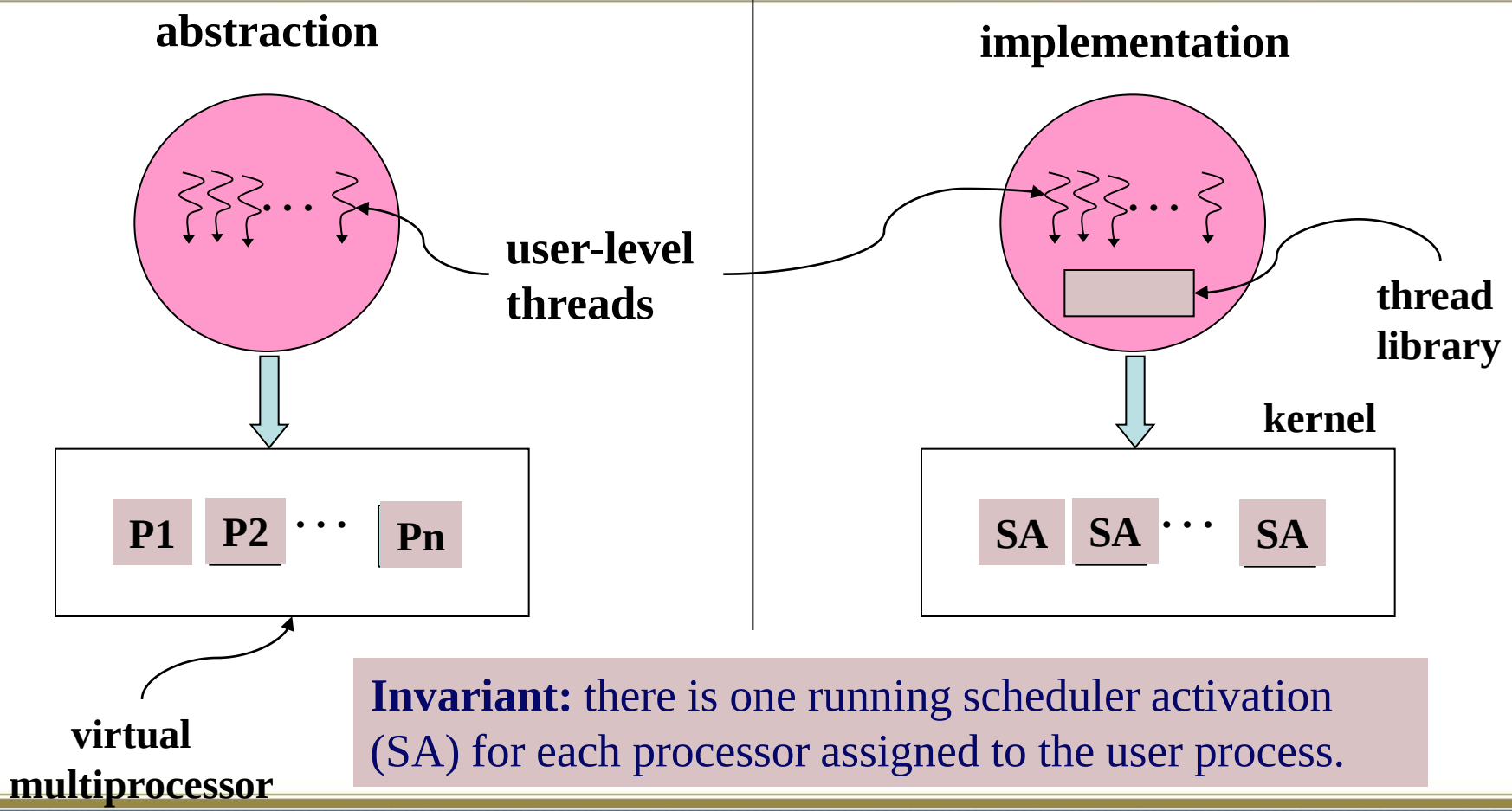
- **Structure to provide communication between the kernel processor and the user-level thread system.**
- **A scheduler activation:**
  - Takes the place of a kernel thread as an execution context for user-level threads.
  - Notifies the user-level thread system of kernel changes.
  - Provides space in the kernel for storing the processor context when the thread is swapped out.

FIU | FLORIDA INTERNATIONAL UNIVERSITY

Scheduler Activations

- **The kernel provides each user-level thread system with its own virtual processor.**
  - Kernel has control over how many processors to give.
    - Each user-level thread system chooses which threads to run on it's allocated processors.
    - Kernel notifies the user-level thread system of any changes, through the scheduler activation.
      - processors assigned, I/O interrupts, page faults.
- **One scheduler activation per processor assigned to address space.**

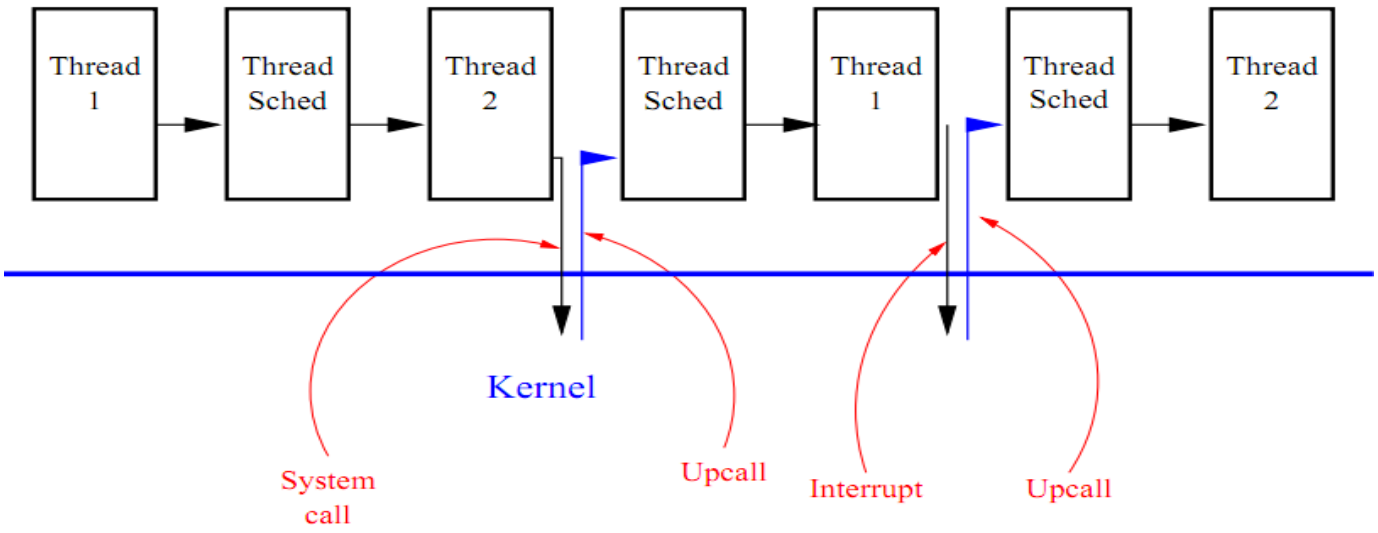
Role of Scheduler Activations



Invariant: there is one running scheduler activation (SA) for each processor assigned to the user process.

Communication via Upcalls

The kernel-level scheduler activation mechanism communicates with the user-level thread library by a set of upcalls:



Communication : Kernel to User Level

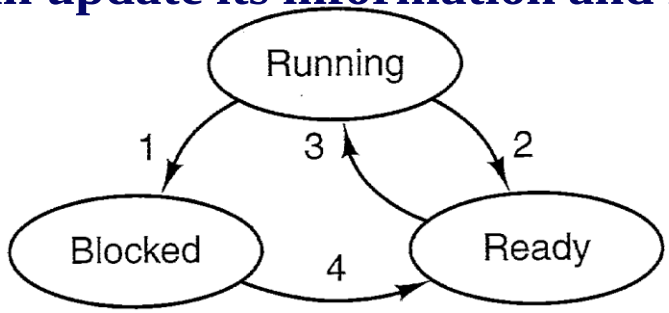
- **Add this processor** (processor #)
  - *Execute a runnable user-level thread.*
- **Processor has been preempted** (preempted activation # and its machine state)
  - *Return to the ready list the user-level thread that was executing in the context of the preempted scheduler activation.*
- **Scheduler activation has blocked** (blocked activation #)
  - *The blocked scheduler activation is no longer using its processor.*
- **Scheduler activation has unblocked** (unblocked activation # and its machine state)
  - *Return to the ready list the user-level thread that was executing in the context*

Communication : Kernel to User Level

Notice that these four messages correspond neatly with this graph.

- It shows the different states a process or kernel thread can be in: running, ready to run, or blocked.
- Whenever a kernel thread system would change the state of a thread "transparently", scheduler activations contact the user-level scheduler.

It can update its information and rethink.

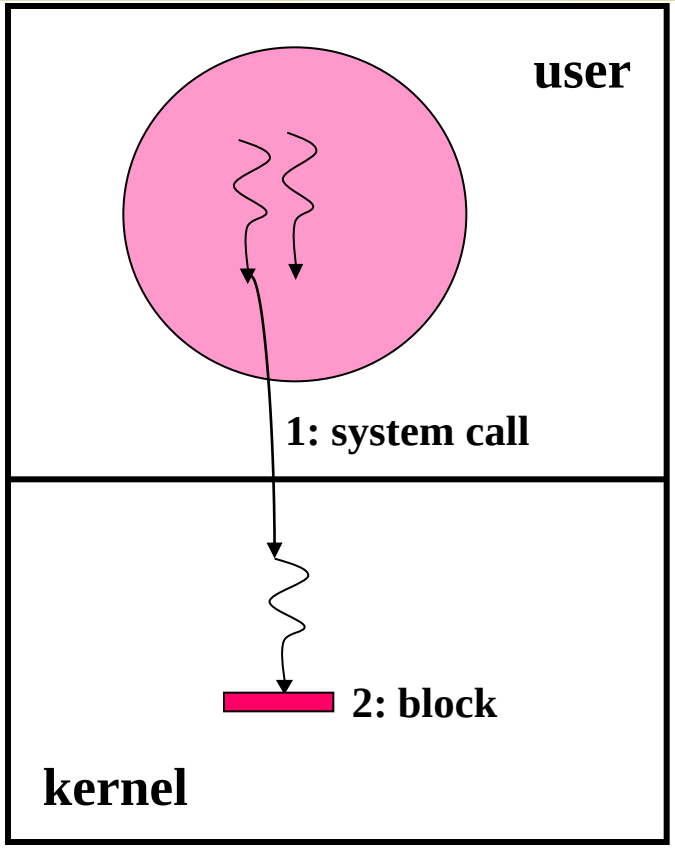


1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

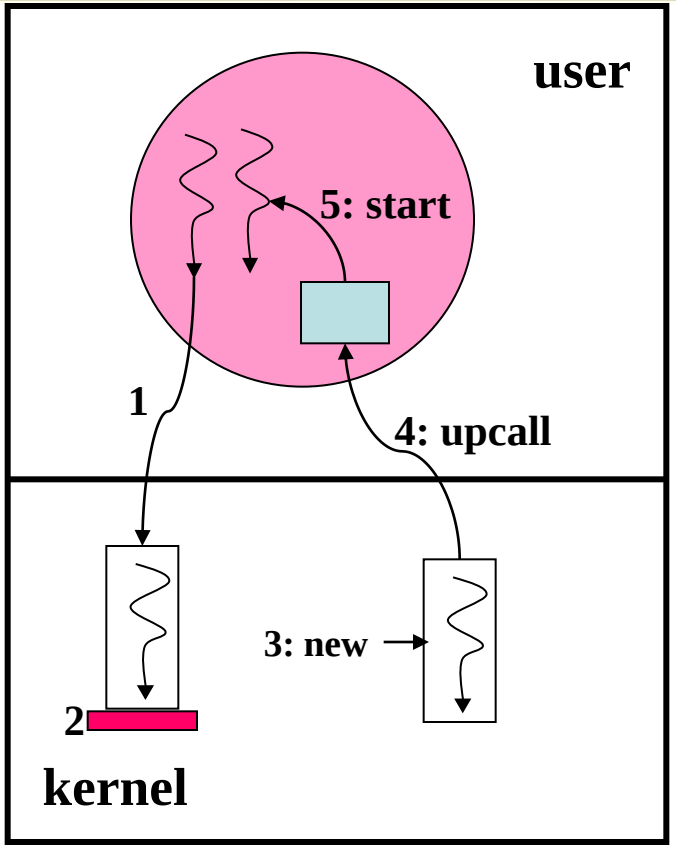
Figure 2-2. A process can be in running, blocked, or ready state. Transitions between these states are as shown.

FLORIDA INTERNATIONAL UNIVERSITY

Avoid Effects of Blocking

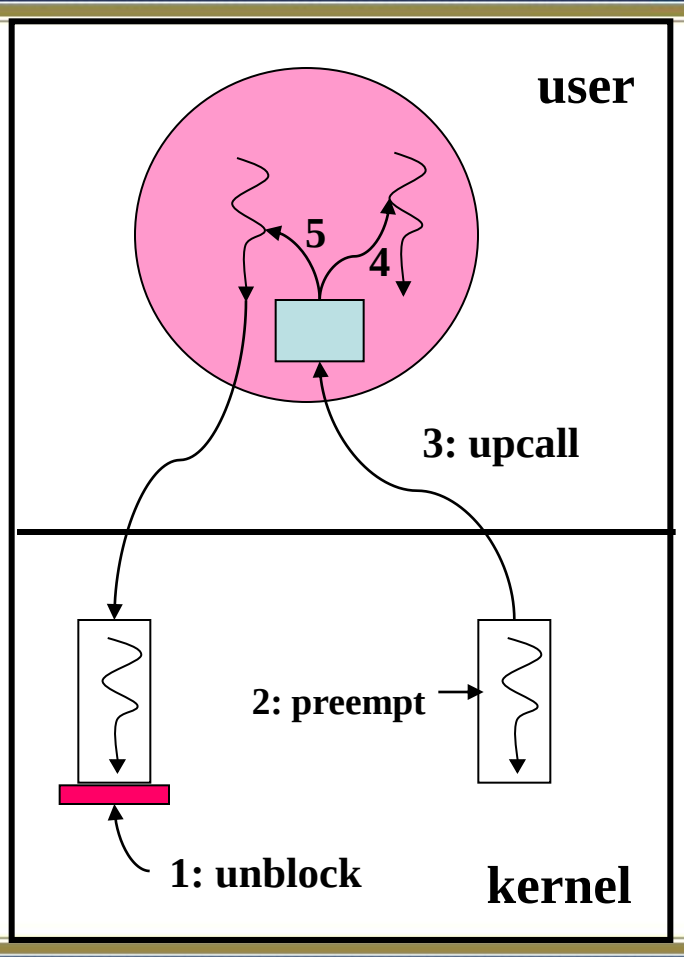


Kernel threads



Scheduler Activations

Resuming Blocked Thread



4: preempt
5: resume

Communication : User Level to Kernel

- **#Runnable Threads != # Processors**
- **Add more processors (additional # of processors needed)**
  - Allocate more processors to this address space and start them running scheduler activations.
- **This processor is idle ()**
  - Preempt this processor if another address space needs it.

Critical Sections

- **A user-level thread can be preempted while it is executing a critical section.**
  - Could result in deadlock or performance drop.
- **If a thread that has been preempted was executing a critical section, a user-level context switch is necessary.**
  - Thread continues temporarily, finishes its critical section, and then relinquishes control back to the upcall.
- **Implemented by making a copy of each low-level critical section.**
  - In the copy, code is added to yield the processor back to the resuming thread.
  - If a preemption occurs, the kernel resumes the thread in the corresponding place in the copy.

FIU | FLORIDA INTERNATIONAL UNIVERSITY

Example: Managing I/O

T1: The application receives two processors to run on from the kernel.

It starts running threads on them.

T2: One of the user-level threads blocks in the kernel.

Kernel uses a fresh scheduler activation to notify the user-level system of this.

T3: The blocking-event completes. Kernel preempts second processor to do the upcall.

Upcall notifies user level of upcall and of event completion.

T4: Upcall takes the thread from the ready list and begins running it.

The first two scheduler activations have now been discarded and replaced.

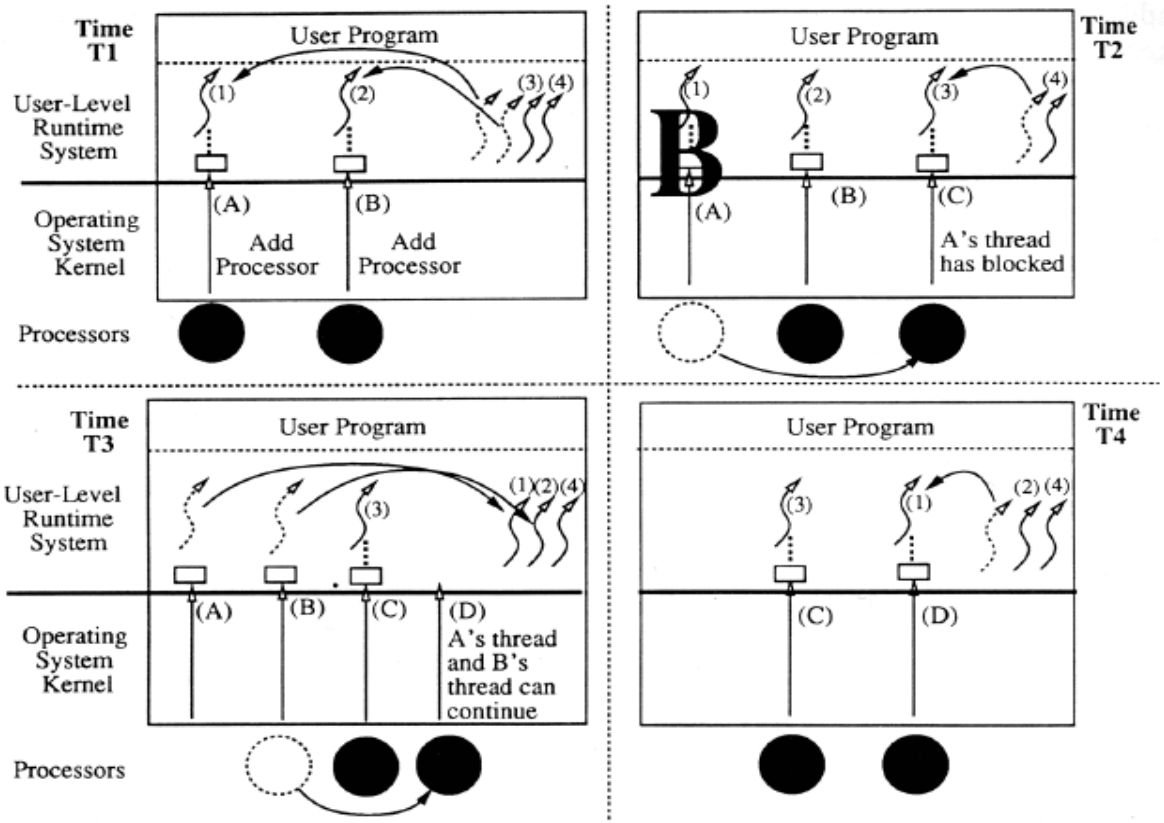


Fig. 1. Example: I/O request/completion.

FIU FLORIDA INTERNATIONAL UNIVERSITY

Outline

- Background
- Problem Statement
- The Approach
- Related Works
- User-level Thread
- Kernel-level Thread
- Scheduler Activation
- Implementation
- Experimental Evaluation
- Summary

FIU | FLORIDA INTERNATIONAL UNIVERSITY

Implementation

- Modification of Topaz Operating System
- Topaz is OS for DEC SRC Firefly a mutli-processor workstation
- Modification of FastThreads a user level thread package
- Lines of code used

FIU FLORIDA INTERNATIONAL UNIVERSITY

Implementation

Implementation Sections

- Processor Allocation Policy
- Thread Scheduling Policy
- Performance Enhancement
- Debugging Considerations

FIU FLORIDA INTERNATIONAL UNIVERSITY

Processor Allocation Policy

- Similar to dynamic policy of Zahorjan and McCann using Space-Shares
- Possible for address space to use kernel threads
- Support for Topaz kernel threads
- No need for static partitioning of processors

Thread Scheduling Policy

- Kernel has not knowledge of an application's concurrency model or scheduling policy
- Applications has freedom to decide how to handle scheduling
- FastThreads use of per processor ready lists with Last In First out order

FIU | FLORIDA INTERNATIONAL UNIVERSITY

Thread Scheduling Policy

- Hysteresis used to avoid unnecessary processor reallocation
- Hysteresis Example

Performance Enhancement

- Critical sections
  - Typical solution using flags. Disadvantage: has overhead
  - Solution used, similar to Trellis/Owl garbage collector. Advantage: no overhead in common case

Performance Enhancement

- Management of scheduler activations
  - Unused scheduler activations can be cached for reuse
  - Scheduler activation sent in bulk for reuse

Debugging Considerations

- Firefly Topaz Debugger
  - User level
  - Application

FIU FLORIDA INTERNATIONAL UNIVERSITY

Outline

- Background
- Problem Statement
- The Approach
- Related Works
- User-level Thread
- Kernel-level Thread
- Scheduler Activation
- Implementation
- Experimental Evaluation
- Summary

FIU FLORIDA INTERNATIONAL UNIVERSITY

Experimental Evaluation

- Three questions
  - What is the cost of user-level thread operations
  - What is the cost of communication between kernel and the user level
  - What is the overall effect on the performance of applications

FIU FLORIDA INTERNATIONAL UNIVERSITY

Experimental Evaluation

Experimental Evaluation Sections

- Thread Performance
- Upcalls
- Application Performance

Thread Performance

- Cost of user threads, same as FastThreads

Operation	FastThreads on Topaz threads	FastThreads on Scheduler Activations	Topaz threads	Ultrix processes
Null Fork	34	37	948	11300
Signal-Wait	37	42	441	1840

Table 4: Thread Operation Latencies (μsec.)

Upcall

- Measure of Upcall performance
  - Two user-level threads to signal and wait

FIU | FLORIDA INTERNATIONAL UNIVERSITY

Application Performance

- Parallel Application using Topaz kernel threads
- Setup of application and modifications
- Procedure and methods.
- Comparison to FastThreads

FIU FLORIDA INTERNATIONAL UNIVERSITY

Application Performance

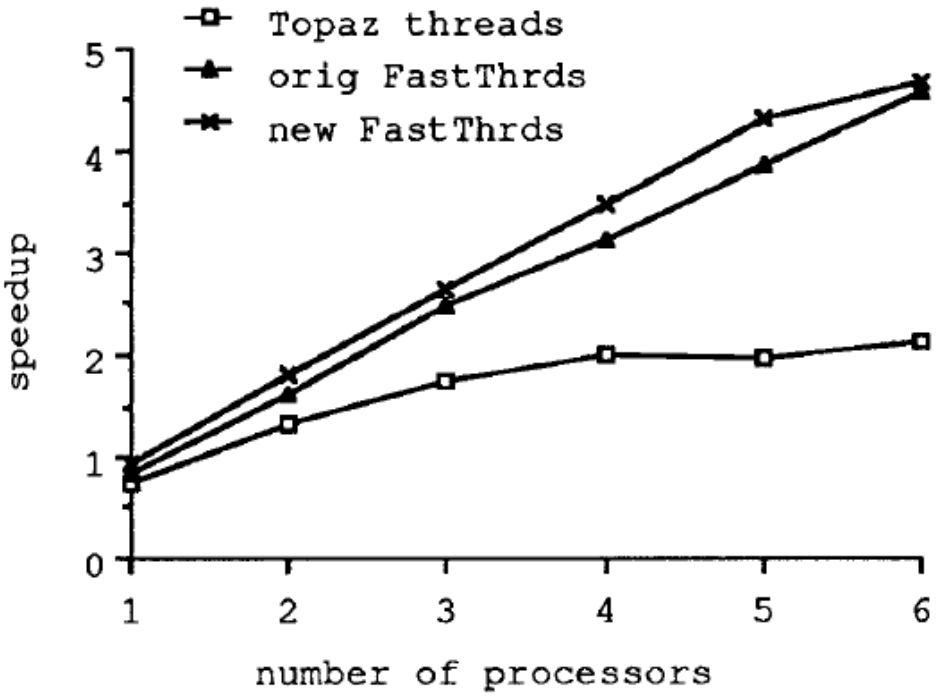


Figure 1: Speedup of N-Body Application vs. Number of Processors, 100% of Memory Available

FIU FLORIDA INTERNATIONAL UNIVERSITY

Application Performance

- Effect of performance application-induced kernel events
- Better performance than FastThreads

FIU FLORIDA INTERNATIONAL UNIVERSITY

Application Performance

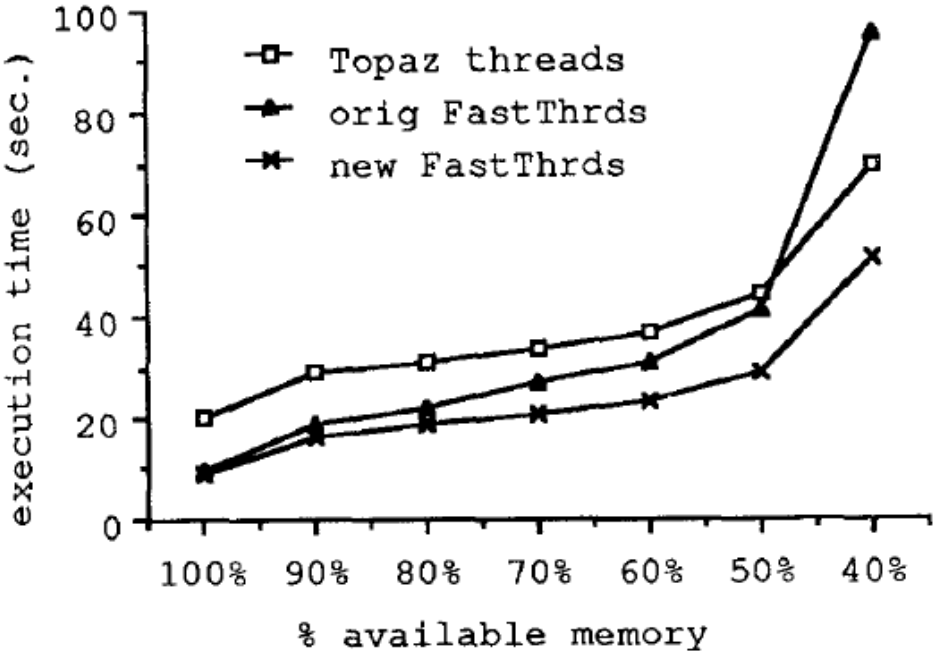


Figure 2: Execution Time of N-Body Application vs. Amount of Available Memory, 6 Processors

Application Performance

Topaz threads	Original FastThreads	New FastThreads
1.29	1.26	2.45

Table 5: Speedup for N-Body Application, Multiprogramming Level = 2, 6 Processors, 100% of Memory Available

FIU | FLORIDA INTERNATIONAL UNIVERSITY

Outline

- Background
- Problem Statement
- The Approach
- Related Works
- User-level Thread
- Kernel-level Thread
- Scheduler Activation
- Implementation
- Experimental Evaluation
- Summary

Summary

- Implement user-level thread package
- Virtual Multiprocessor in application address space
- Divided responsibilities

FIU | FLORIDA INTERNATIONAL UNIVERSITY

Summary

- Processor allocation
- Thread Scheduling
- Kernel notifications to address space thread scheduler
- Address space notifications to kernel for processor allocations

FIU FLORIDA INTERNATIONAL UNIVERSITY

Questions?

- Questions?