

CSE

## Scheduler Activations

THE UNIVERSITY OF NEW SOUTH WALES

Including some slides modified from Raymond Namyst, U. Bordeaux

CSE

## Learning Outcomes

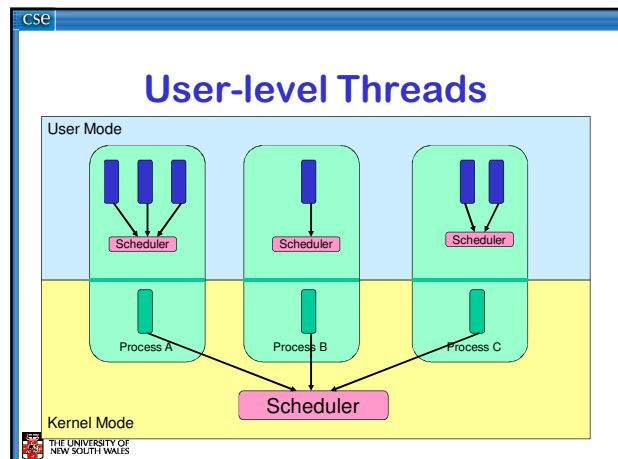
- An understanding of hybrid approaches to thread implementation
- A high-level understanding of scheduler activations, and how they overcome the limitations of user-level and kernel-level threads.

THE UNIVERSITY OF NEW SOUTH WALES

CSE

- Thomas Anderson, Brian Bershad, Edward Lazowska, and Henry Levy. Scheduler Activations: Effective Kernel Support for the User-Level management of Parallelism. ACM Trans. on Computer Systems 10(1), February 1992, pp. 53-79.

THE UNIVERSITY OF NEW SOUTH WALES

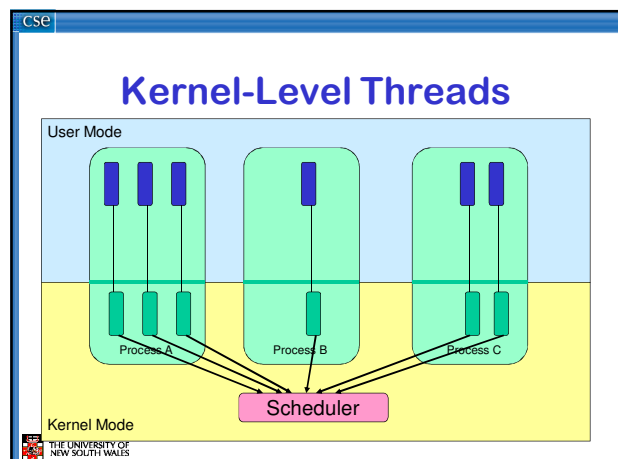


CSE

## User-level Threads


- ✓ Fast thread management (creation, deletion, switching, synchronisation...)
- ✗ Blocking blocks all threads in a process
  - Syscalls
  - Page faults
- ✗ No thread-level parallelism on multiprocessor

THE UNIVERSITY OF NEW SOUTH WALES



## Kernel-level Threads

- ✗ Slow thread management (creation, deletion, switching, synchronisation...)
  - System calls
- ✓ Blocking blocks only the appropriate thread in a process
- ✓ Thread-level parallelism on multiprocessor




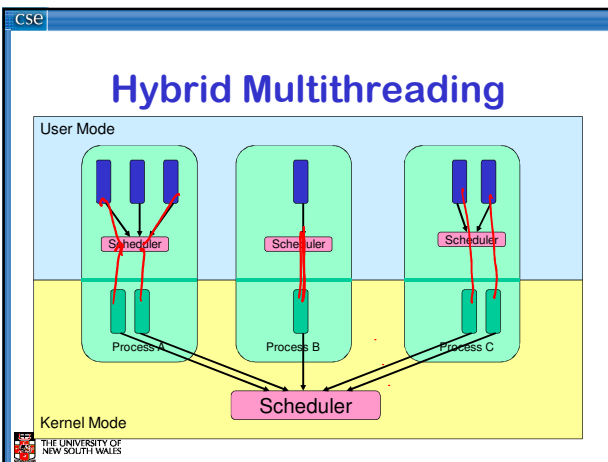
## Performance

Table I: Thread Operation Latencies (µsec.)

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


User-level threads

Kernel-level threads


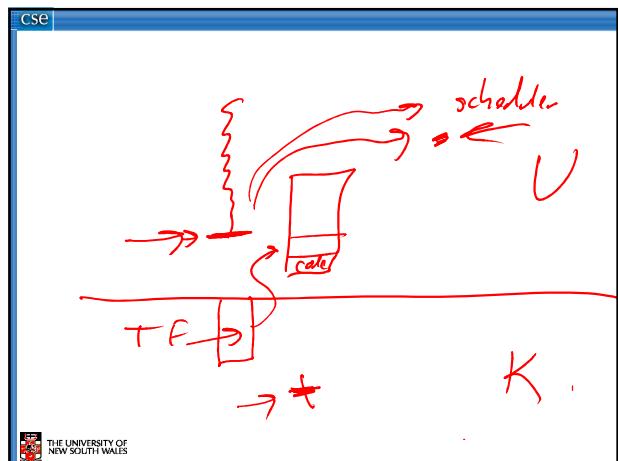
## Hybrid Multithreading

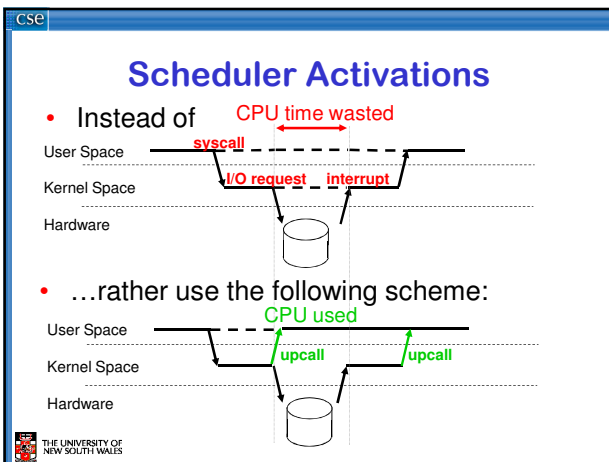
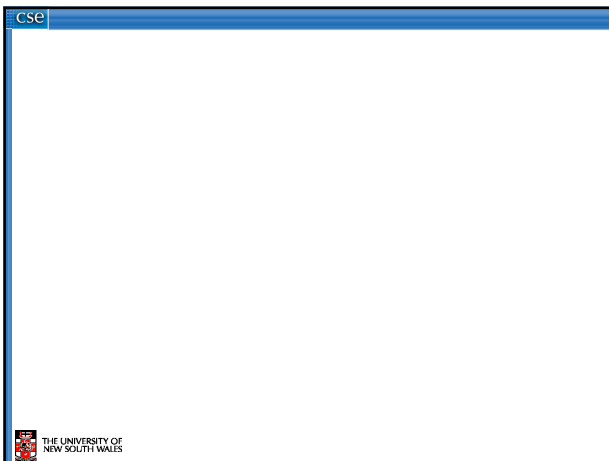
- ✓ Can get real thread parallelism on multiprocessor
- ✗ Blocking still a problem!!!



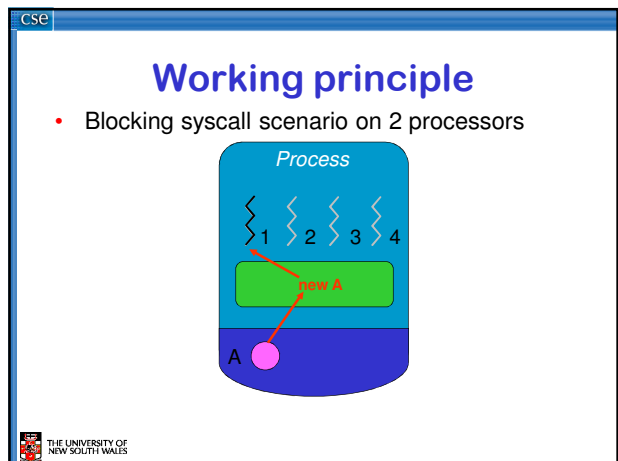
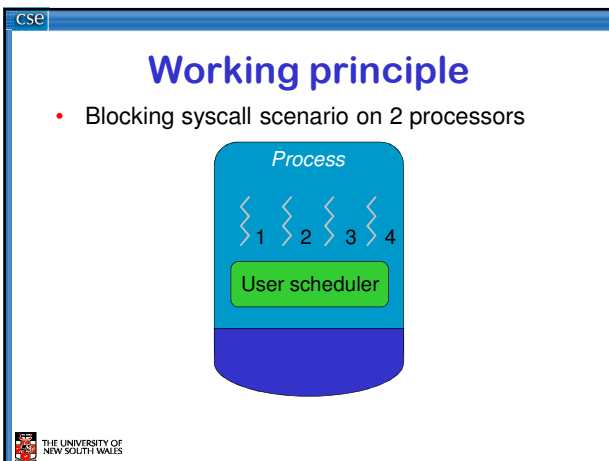
## Scheduler Activations

- First proposed by [Anderson et al. 91]
- Idea: Both schedulers co-operate
  - User scheduler uses system calls
  - Kernel scheduler uses upcalls!
- Two important concepts
  - Upcalls
    - Notify the user-level of kernel scheduling events
  - Activations
    - A new structure to support upcalls and execution
      - approximately a kernel thread
    - As many running activations as (allocated) processors
    - Kernel controls activation creation and destruction



- ### Upcalls to User-level scheduler
- **New** (processor #)
    - Allocated a new virtual CPU
    - Can schedule a user-level thread
  - **Preempted** (activation # and its machine state)
    - Deallocated a virtual CPU
    - Can schedule one less thread
  - **Blocked** (activation #)
    - Notifies thread has blocked
    - Can schedule another user-level thread
  - **Unblocked** (activation # and its machine state)
    - Notifies a thread has become runnable
    - Must decided to continue current or unblocked thread



Cse

### Working principle

- Blocking syscall scenario on 2 processors

Process

1 2 3 4

A B

New B

THE UNIVERSITY OF NEW SOUTH WALES

Cse

### Working principle

- Blocking syscall scenario on 2 processors

Process

1 2 3 4

A B

THE UNIVERSITY OF NEW SOUTH WALES

Cse

### Working principle

- Blocking syscall scenario on 2 processors

Process

1 2 3 4

A B

Preempt A-B

Preempt

THE UNIVERSITY OF NEW SOUTH WALES

Cse

### Working principle

- Blocking syscall scenario on 2 processors

Process

1 2 3 4

B

THE UNIVERSITY OF NEW SOUTH WALES

Cse

### Working principle

- Blocking syscall scenario on 2 processors

Process

1 2 3 4

A B

Blocking syscall

THE UNIVERSITY OF NEW SOUTH WALES

Cse

### Working principle

- Blocking syscall scenario on 2 processors

Process

1 2 3 4

A B C

New C - blocked B

THE UNIVERSITY OF NEW SOUTH WALES

### Working principle

- Blocking syscall scenario on 2 processors

The diagram shows a process with four threads (1, 2, 3, 4) and three processors (A, B, C). Thread 1 is on processor A, thread 2 on processor B, and thread 3 on processor C. Thread 4 is blocked. An arrow labeled 'I/O completion' points to processor C.

### Working principle

- Blocking syscall scenario on 2 processors

The diagram is similar to the previous one, but with a red arrow pointing from thread 3 to thread 4, labeled 'Unblock B + prevent C'.

### Working principle

- Blocking syscall scenario on 2 processors

The diagram is similar to the previous ones, but with a red lightning bolt symbol over processor B, indicating a scheduler activation.

### Scheduler Activations

- Thread management at user-level
  - Fast
- Real thread parallelism via activations
  - Number of activations (virtual CPUs) can equal CPUs
- Blocking (syscall or page fault) creates new activation
  - User-level scheduler can pick new runnable thread.
- Fewer stacks in kernel
  - Blocked activations + number of virtual CPUs

### Performance

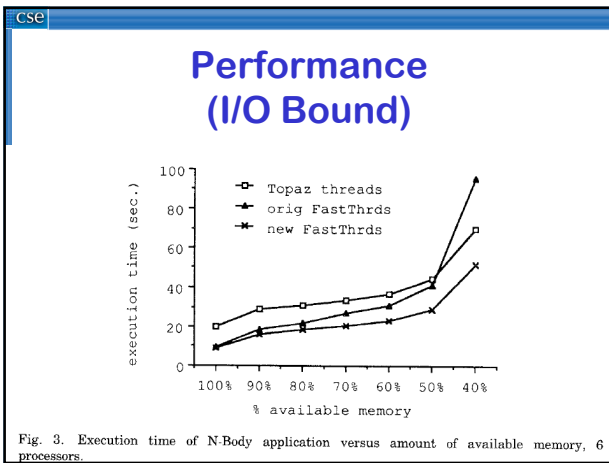
Table IV. Thread Operation Latencies ( $\mu\text{sec.}$ )

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

### Performance (compute-bound)

The graph shows speedup vs number of processors for three thread management methods. 'Topaz threads' (open squares) shows the lowest speedup, reaching about 2.2 at 6 processors. 'orig FastThrds' (filled triangles) and 'new FastThrds' (filled stars) show much higher speedup, both reaching nearly 5 at 6 processors.

Fig. 2. Speedup of N-Body application versus number of processors, 100% of memory available.



- ### Adoption
- Adopters
    - BSD "Kernel Scheduled Entities"
      - Reverted back to kernel threads
    - Variants in Research OSs: K42, Barrelfish
    - Digital UNIX
    - Solaris
    - Mach
    - Windows 7 64-bit *User Mode Scheduling*
  - Linux -> kernel threads

