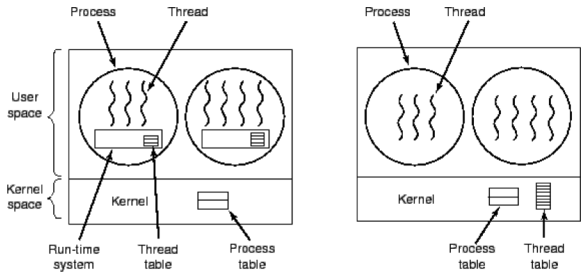
Graduate Operating Systems
(Threads & Events)

Fall 2020

User Threads vs. Kernel Threads



User Threads vs. Kernel Threads

- "Lightweight" vs. "heavyweight"
- Concurrency vs. parallelism
- Control (or lack thereof)
- (Portability)

- **Scheduler activations:** combine benefits of kernel-level threads and user-level threads

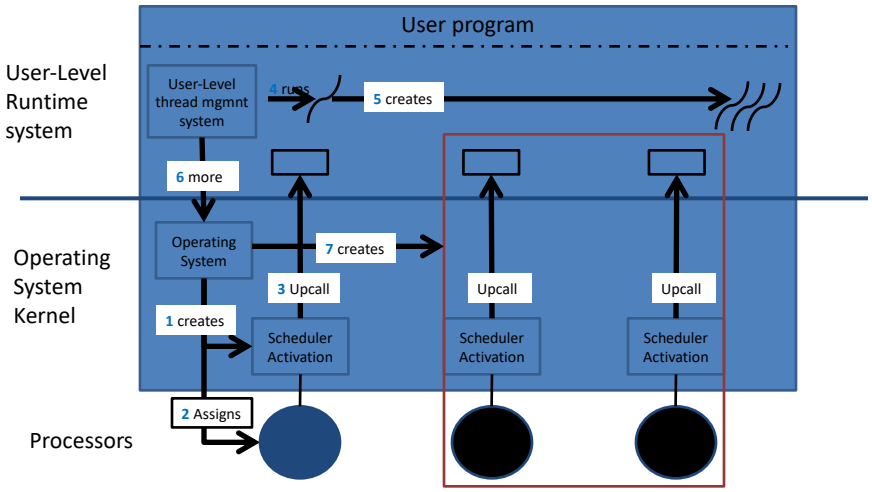
Kernel/User Level Integration

- "Virtual processors" allocated by kernel
- ULTS controls which threads to run
- Kernel **notifies** ULTS when changes are made (number of processors) or blocking occurs
- ULTS **notifies** kernel when more/fewer processors are needed

Scheduler Activations

- Tool for KL & UL communication
  - Kernel: “notify UL of events that impact user-level scheduling”
  - UL: “notify KL of events that can affect processor allocation”
- System calls vs. upcalls
- Scheduler activation: “execution context for an event vectored from the kernel to an address space”

Scheduler Activations



Scheduler Activations (Upcalls)

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 of the blocked scheduler activation.

Example: Blocking

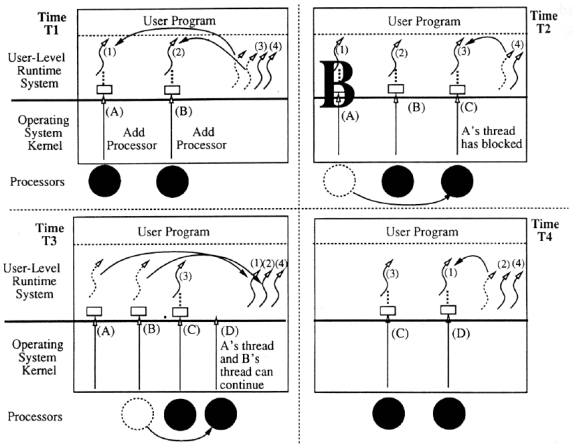


Fig. 1. Example: I/O request/completion.

Scheduler Activations (System Calls)

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.

Scheduler Activations

- What if user-level thread is in critical section when it is blocked or preempted?
- Prevention & recovery

Paper "DThreads"

- Multithreaded programming hard
- Enforce deterministic execution (but be efficient)
- Heisenbugs
- **Same program + same inputs = always same outputs**
- Goals of Dthreads: deterministic execution, easy to deploy, robust to changes in input/architectures/code, eliminates cache-line *false sharing*, efficient.
- How: turn multithreaded apps into multiple processes with private *copy-on-write* mappings to shared memory

Paper "DThreads"

- Pthread: race conditions (Figure 1)
- DThreads: deterministic output (Figure 2)
- Synchronization points
- Last-writer wins protocol
- Deterministic thread index
- Memory mapped files
- Global token (serialization, locks, condition variables, barriers)