CS 134:
Operating Systems
Threads

2012-12-06

CS34

CS 134:
Operating Systems
Threads

Overview

Wiki Answers

- Thread Questions
- Scheduler Questions
- Synchronization Questions

Threads

- Concepts
- Uses
- Models
- Design

Overview
Wiki Answers
Thread Questions
Scheduler Questions
Synchronization Questions
Threads
Concepts
Uses
Models
Design

Thread Questions (1)

What happens to a thread when it exits (i.e., calls thread_exit())? What about when it sleeps?

When a thread exits, it ensures the stack isn't mangled, removes its virtual memory space and destroys it, decrements the counter of whatever vnode it may be poitning at, puts itself into a zombie state, S_ZOMB, and preps itself to panic if it ever runs again before it dies. When it sleeps, it makes sure it's not in an interrupt handler, yields control to the next thread, enters the S_SLEEP state, and only starts taking control once more when wakeup() is called on its address.

Thread Questions (1)

What happens to a thread when it exits (i.e., calls thread_exit())? What about when it sleeps?

When a thread exits, it ensures the stack isn't mangled, removes its virtual memory space and destroys it, decrements the counter of whatever vnode it may be poitning at, puts itself into a zombie state, S_ZOMB, and preps itself to panic if it ever runs again before it dies. When it sleeps, it makes sure it's not in an interrupt handler, yields control to the next thread, enters the S_SLEEP state, and only starts taking control once more when wakeup() is called on its address.

Thread Questions (2)

What function(s) handle(s) a context switch?

There are two functions that handle a context switch: mi_switch, which is the high level, machine-independent context switch function, and md_switch, which is the machine-independent code that actually does the context switch. mi_switch is in thread.c, and md_switch is in pcb.c

Thread Questions (2)

What function(s) handle(s) a context switch?

There are two functions that handle a context switch: mi_switch, which is the high level, machine-independent context switch function, and md_switch, which is the machine-independent code that actually does the context switch. mi_switch is in thread.c, and md_switch is in pcb.c

Thread Questions (3)

How many thread states are there? What are they?

There are four thread states - S_RUN, S_READY, S_SLEEP, and S_ZOMB. These states are defined in kern/thread/thread.c. They express whether the thread is running, ready to run, sleeping, or a zombie.

Thread Questions (3)

How many thread states are there? What are they?

There are four thread states - S_RUN, S_READY, S_SLEEP, and S_ZOMB. These states are defined in kern/thread/thread.c. They express whether the thread is running, ready to run, sleeping, or a zombie.

Thread Questions (4)

What does it mean to turn interrupts off? How is this accomplished? Why is it important to turn off interrupts in the thread subsystem code?

If interrupts are turned off, then even if an interrupt is signaled the handler is not called until interrupts are turned back on. Interrupts are turned off using the function splhigh (set priority level high) and back on again using spl0 (set priority level zero). The priority level can also be set to intermediate levels (or at least, it could if OS/161 supported them) using the splx function. Turning off interrupts for thread operations is necessary to ensure that these operations complete successfully and aren't broken mid-execution. For example, things could go pretty badly if the scheduler interrupted us in the middle of a context switch and tried to start executing a thread that wasn't finished setting up its stack. And it would be really awful if someone interrupted us in the middle of forking!

2012-12-06

Thread Questions (4)

What does it mean to turn interrupts off? How is this accomplished? Why is it important to turn off interrupts in the thread subsystem code?

If interrupts are turned off, then even if an interrupt is signaled the handler is not called until interrupts are turned back on. Interrupts are turned off using the function splhigh (set priority level high) and back on again using spl0 (set priority level zero). The priority level can also be set to intermediate levels (or at least, it could if OS/161 supported them) using the splx function. Turning off interrupts for thread operations is necessary to ensure that these operations complete successfully and aren't broken mid-execution. For example, things could go pretty badly if the scheduler interrupted us in the middle of a context switch and tried to start executing a thread that wasn't finished setting up its stack. And it would be really awful if someone interrupted us in the middle of forking!

Thread Questions (5)

What happens when a thread wakes up another thread? How does a sleeping thread get to run again?

It removes the sleeping thread from the queue, and calls make_runnable on the thread, which currently adds it to the end of the runqueue. The thread gets to run again when an mi_switch is called, and that thread is returned by the scheduler.

Thread Questions (5)

What happens when a thread wakes up another thread? How does a sleeping thread get to run again?

It removes the sleeping thread from the queue, and calls make_runnable on the thread, which currently adds it to the end of the runqueue. The thread gets to run again when an mi_switch is called, and that thread is returned by the scheduler.

Scheduler Questions (6)

What function is responsible for choosing the next thread to run?
How does that function pick the next thread?

struct thread * scheduler(void); it uses a round-robin run queue that schedules each thread in the queue in equal time-slice without priorities.

Scheduler Questions (6)

What function is responsible for choosing the next thread to run?
How does that function pick the next thread?

struct thread * scheduler(void); it uses a round-robin run queue that schedules each thread in the queue in equal time-slice without priorities.

Scheduler Questions (7)

What role does the hardware timer play in scheduling? What hardware independent function is called on a timer interrupt?

The interrupt handler for the hardware timer calls hardclock, defined in src/kern/thread/hardclock.c. The method hardclock finishes by calling thread_yield every time it is run, forcing a context switch.

Scheduler Questions (7)
What role does the hardware timer play in scheduling? What hardware independent function is called on a timer interrupt?
The interrupt handler for the hardware timer calls hardclock, defined in src/kern/thread/hardclock.c. The method hardclock finishes by calling thread_yield every time it is run, forcing a context switch.

Synchronization Questions (8)

Describe how thread_sleep() and thread_wakeup() are used to implement semaphores. What is the purpose of the argument passed to thread_sleep()?

thread_sleep is used in the P function of the semaphore. This function suspends the current thread until the semaphore count is greater than zero.
thread_wakeup() is used in the V function of the semaphore. This function wakes up all the suspended threads waiting on the current semaphore.
The addr argument that is passed in is the address of the object (in this case, semaphore) the sleeping thread is associated with. This is required so that when thread_wakeup is called on the same semaphore, it can selectively wake up only the threads associated with that particular semaphore.

Synchronization Questions (8)
Describe how thread_sleep() and thread_wakeup() are used to implement semaphores. What is the purpose of the argument passed to thread_sleep()?
thread_sleep is used in the P function of the semaphore. This function suspends the current thread until the semaphore count is greater than zero.
thread_wakeup() is used in the V function of the semaphore. This function wakes up all the suspended threads waiting on the current semaphore.
The addr argument that is passed in is the address of the object (in this case, semaphore) the sleeping thread is associated with. This is required so that when thread_wakeup is called on the same semaphore, it can selectively wake up only the threads associated with that particular semaphore.

Why does the lock API in OS/161 provide lock_do_i_hold(), but not lock_get_holder()?

???

Why does the lock API in OS/161 provide lock_do_i_hold(), but not lock_get_holder()?

???

The thread subsystem in OS/161 uses a queue structure to manage some of its state. This queue structure does not contain any synchronization primitives. Why not? Under what circumstances should you use a synchronized queue structure?

The runqueue queue used by the scheduler in the thread subsystem is only accessed by a single scheduler thread, so does not need any synchronization primitives to prevent other (non-existent) threads from messing up the queue. You should use a synchronized queue structure for any queue that multiple threads could access simultaneously.

The thread subsystem in OS/161 uses a queue structure to manage some of its state. This queue structure does not contain any synchronization primitives. Why not? Under what circumstances should you use a synchronized queue structure?

The runqueue queue used by the scheduler in the thread subsystem is only accessed by a single scheduler thread, so does not need any synchronization primitives to prevent other (non-existent) threads from messing up the queue. You should use a synchronized queue structure for any queue that multiple threads could access simultaneously.

Generalizing Processes

Simple view of process is

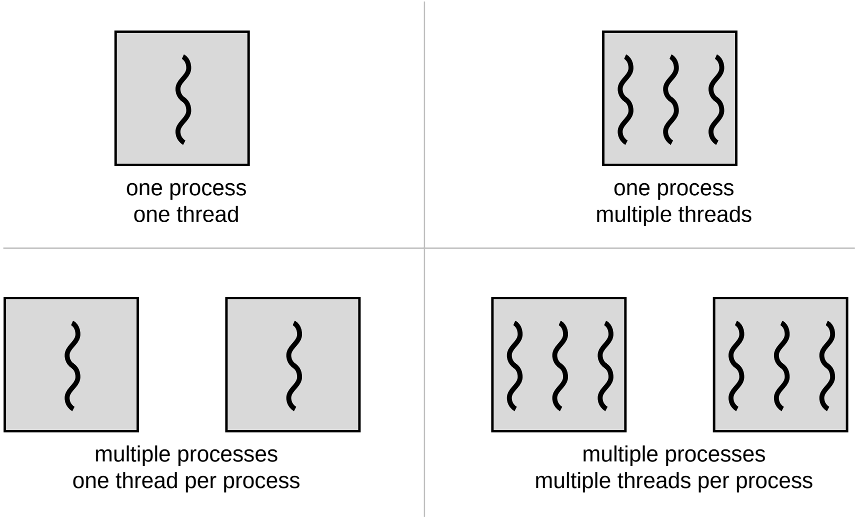
Address space

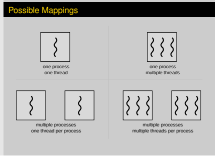
\+ Thread of execution

Does the mapping need to be one-to-one?

Generalizing Processes
Simple view of process is
Address space
+ Thread of execution
Does the mapping need to be one-to-one?

Possible Mappings





Threads

Motivation:

- Traditional processes: Virtual uniprocessor machine
- Multithreaded processes: Virtual multiprocessor machine

Threads

Motivation:
- Traditional processes: Virtual uniprocessor machine
- Multithreaded processes: Virtual multiprocessor machine

Uses of Threads

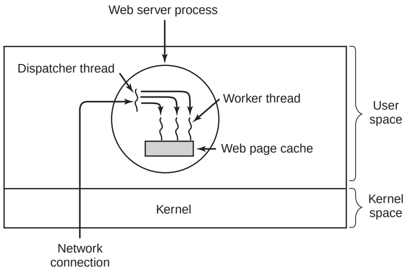
Various reasons why people use threads

- Performing foreground and background work
- Supporting asynchronous processing
- Speeding execution
- Organizing programs

Uses of Threads

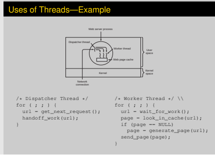
Various reasons why people use threads
- Performing foreground and background work
- Supporting asynchronous processing
- Speeding execution
- Organizing programs

Uses of Threads—Example




```
/* Dispatcher Thread */
for ( ; ; ) {
  url = get_next_request();
  handoff_work(url);
}
```

```
/* Worker Thread */ \\
for ( ; ; ) {
  url = wait_for_work();
  page = look_in_cache(url);
  if (page == NULL)
    page = generate_page(url);
  send_page(page);
}
```

Class Exercise

Can an application implement threads without built-in thread support in the OS?

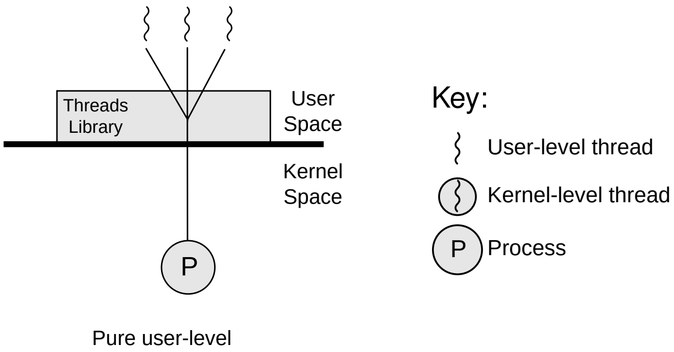
If so, what *does* it need from the from the OS to support threads?

Class Exercise

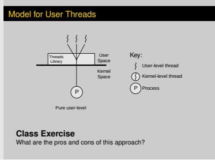
Can an application implement threads without built-in thread support in the OS?

If so, what *does* it need from the from the OS to support threads?

Model for User Threads

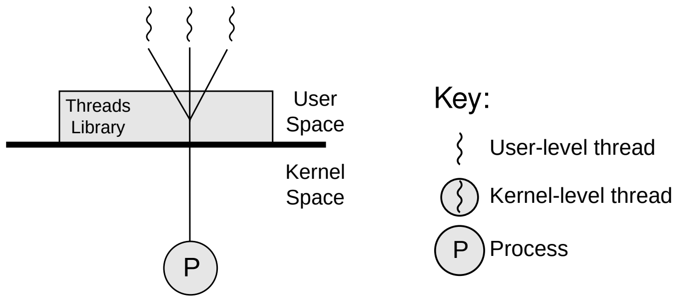


Class Exercise
What are the pros and cons of this approach?



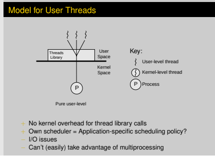
So, maybe we should put the threads in the kernel?

Model for User Threads



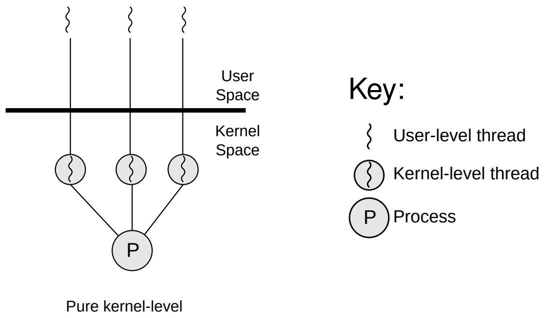
Pure user-level

\+ No kernel overhead for thread library calls
\+ Own scheduler = Application-specific scheduling policy?
– I/O issues
– Can't (easily) take advantage of multiprocessing



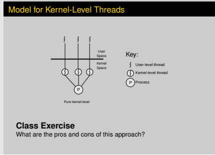
So, maybe we should put the threads in the kernel?

Model for Kernel-Level Threads

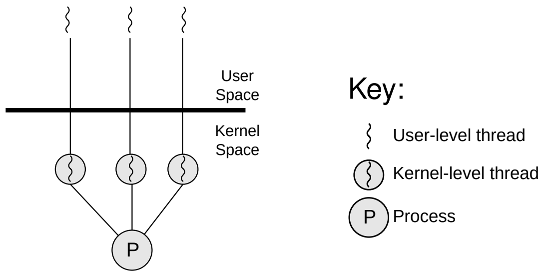


Class Exercise

What are the pros and cons of this approach?



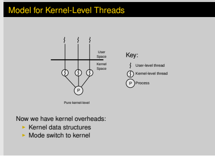
Model for Kernel-Level Threads



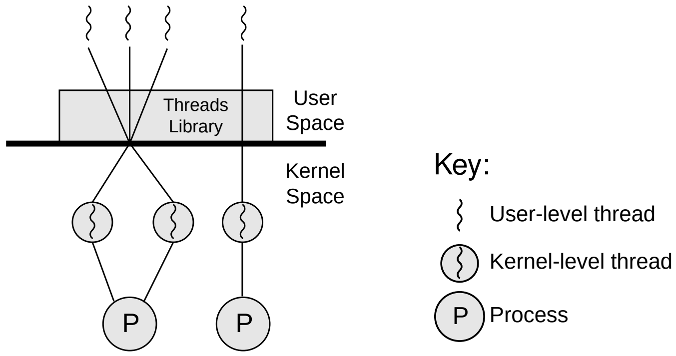
Pure kernel-level

Now we have kernel overheads:
- Kernel data structures
- Mode switch to kernel

2012-12-06



Hybrid Thread Schemes



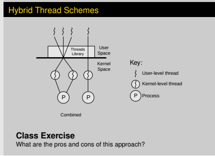
Combined

Class Exercise

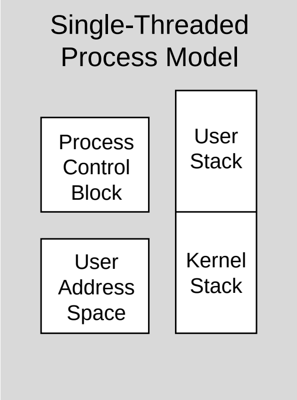
What are the pros and cons of this approach?

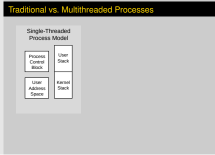
2012-12-06

CS34
└─Threads
  └─Models
    └─Hybrid Thread Schemes

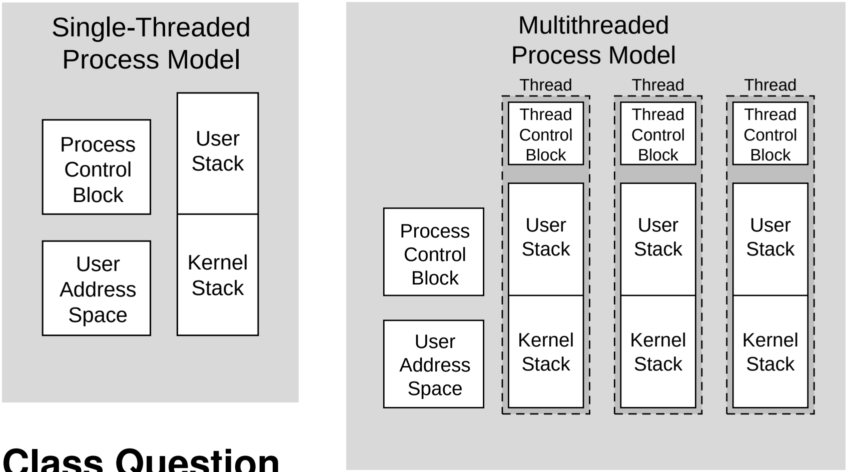


Traditional vs. Multithreaded Processes





Traditional vs. Multithreaded Processes

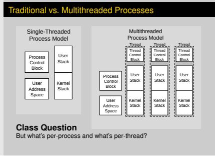


Class Question

But what's per-process and what's per-thread?

2012-12-06

CS34
└Threads
 └Models
  └Traditional vs. Multithreaded Processes



Per-Process vs. Per-Thread—*You* Decide. . .

- Execution state
  - Registers
  - Program counter
  - Program status word
  - Stack pointer
- Scheduling information
  - Process state
  - Priority
  - Class, etc.
- Memory
  - Text area
  - Data area
  - Stack area
- Security/Authentication Info
  - User ID
  - Group ID
- I/O State
  - File descriptors
  - Working directory
  - Root directory
- Event Notifications
  - Signals waiting
  - Signal mask
  - Time of next alarm
- Other
  - Process ID
  - Parent process
  - Process group
  - Controlling terminal
  - Start time
  - CPU time
  - Children's CPU time

Per-Process vs. Per-Thread—*You* Decide. . .
Execution state Registers Program counter Program status word Stack pointer Scheduling information Process state Priority Class, etc. Memory Text area Data area Stack area Security/Authentication Info User ID Group ID I/O State File descriptors Working directory Root directory Event Notifications Signals waiting Signal mask Time of next alarm Other Process ID Parent process Process group Controlling terminal Start time CPU time Children's CPU time