Threads in the Operating System Kernel

Professor Hugh C. Lauer
CS-3013 — Operating Systems

(Slides include copyright materials from *Operating Systems: Three Easy Step*, by Remzi and Andrea Arpaci-Dusseau, from *Modern Operating Systems*, by Andrew S. Tanenbaum, 3rd edition, and from other sources)

In the old days …

- Operating system kernels did *one* thing at a time
- … with interrupts disabled
- … and all processes and threads suspended!
- Challenging enough to keep track of everything

No longer!

- **Desktop PC**
  - ~100 processes
  - > 1,000 threads
  - 8 processors
- **Shared system**
  - 1000s of processes
  - Many 1000s of threads
  - 100s of processors
- **Single-threaded kernel becomes serious bottleneck**

Alternatives

- **Microkernels — e.g., MACH**
  - Different subsystems operate in separate address spaces
  - Communication via message passing
  - Performance issues
- **Cluster systems**
  - Partition applications across computers
  - Shared files, but …
  - … not much else

Need for multi-threaded kernel!

Multi-threaded Kernel

- **Linux kernel became multi-threaded in mid-2000s**
  - Between Linux 2.4.x and 2.6.x
- **Windows, other forms of Unix at about same time**
- **CS-3013 was using a multithreaded kernel in virtual machines by C-term 2008**

Linux approach

- ***Thread*** **is unit of scheduling**

- **Kernel maintains**
  - Interrupt stack for each processor (or core)
    - 4-8 kilobytes
  - Kernel stack for each thread
    - 4-8 kilobytes
    - Fixed location within address space

Interrupt handler

- **Entered with interrupts disabled**
- **Do minimal processing to handle interrupt**
  - Using interrupt stack of interrupted processor!
- **Hand off to some thread for real work**

More later in the course!

Definition – System Call

- A *structured* function call across a protection boundary between less privileged applications and more privileged operating system functions

- Also, across privilege layers of the operating system itself

Protection Boundary

- **Application programs are *not allowed* to**
  - Read or write data structures in the kernel
  - Call functions in the kernel directly
  - Change settings of the machine
  - Control arbitrary devices directly
  - Interfere with the operation of the kernel in any way

- **Enforced by hardware**
  - I.e., one or more bits in PSW

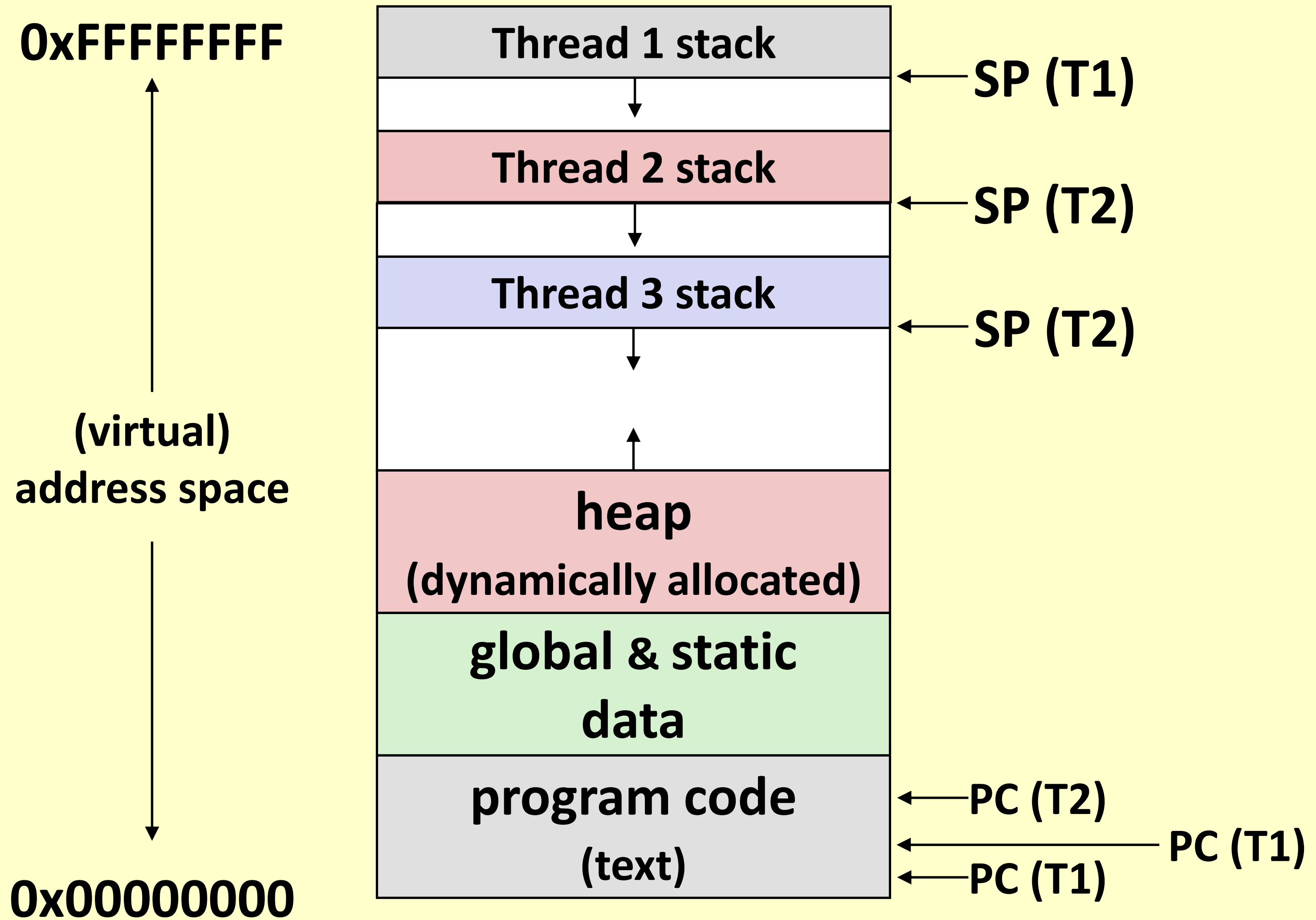
System Call

- A *trap* caused by executing a special machine language instruction
- Causes a *synchronous interrupt* to a specific interrupt/trap handler in the OS
- Allows the OS to control access, check arguments, manage behavior, etc.
- Causes machine to switch modes from "user" to "system" or "privileged"
    - As indicated by bits in the PSW

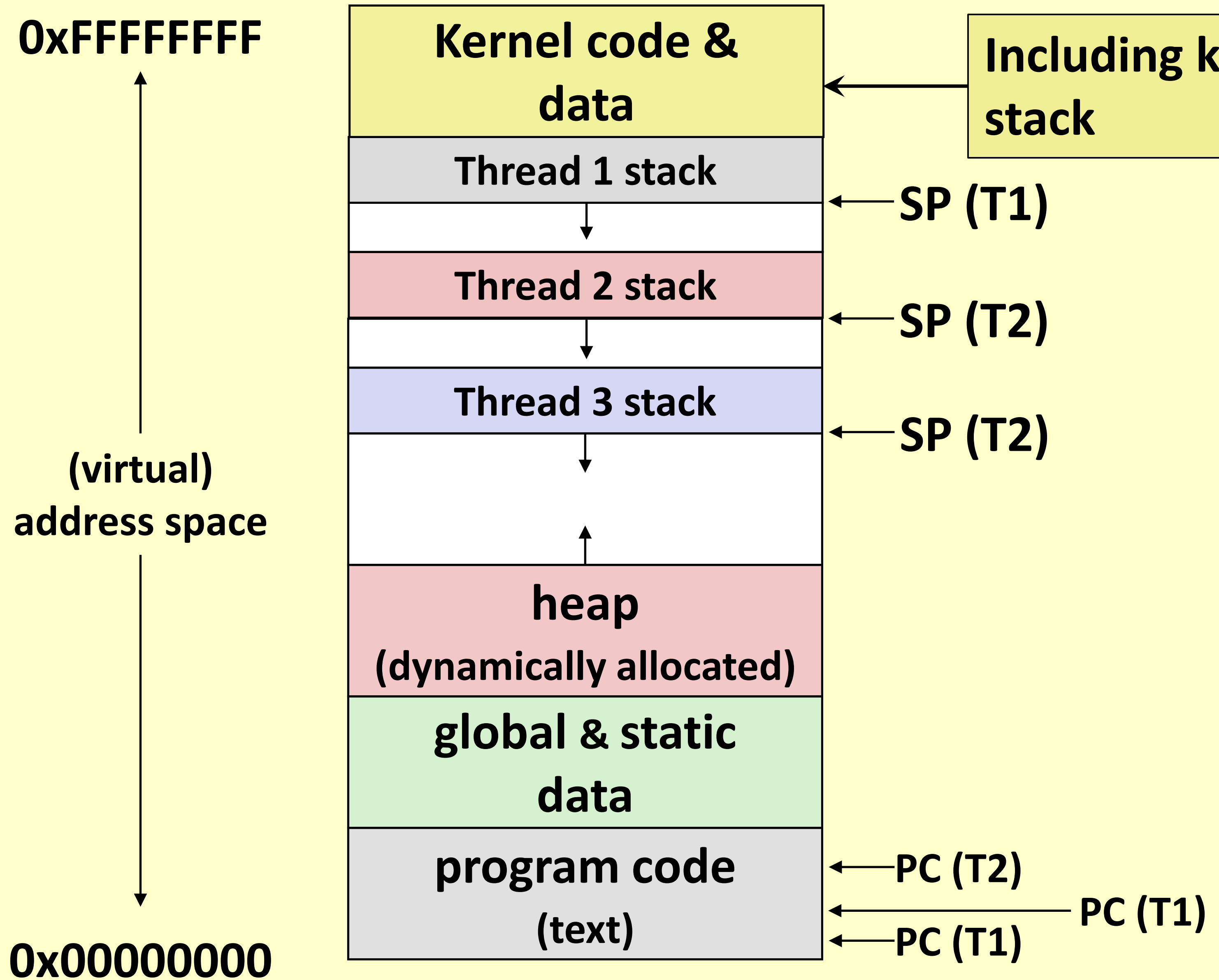
Trap handling

- **Find kernel stack for *this thread***
- **Enable interrupts**
- **Handle the trap or system call**
  - As a kernel function
  - Like the syscall stubs that we implemented in Project 0

From previous topic



In reality:–


0xFFFFFFFF
(virtual)
address space
0x00000000
Kernel code & data
Including kernel stack
Thread 1 stack
SP (T1)
Thread 2 stack
SP (T2)
Thread 3 stack
SP (T2)
heap
(dynamically allocated)
global & static data
program code
(text)
PC (T2)
PC (T1)
PC (T1)

Digression – Process Address Space

- **Linux includes (parts of) *kernel* in every address space**
  - Protected ⟵ But not touchable in non-privileged mode
  - Easy to access ⟵ In privileged mode
  - Allows *kernel functions* to see into client processes
    - Transferring data
    - Examining state
    - …

- **Also many other operating systems**

Linux Kernel Implementation

- **Kernel may execute in either *Interrupt context* or *Process context***

- **In *Interrupt context*, no assumption about what process was executing (if any)**
  - No access to virtual memory, files, resources
  - May not sleep, take page faults, wait for input, etc.

- **In *Process context*, kernel has access to**
  - Virtual memory, files, other process resources
  - May sleep, take page faults, etc., on behalf of process
  - May access shared resources & wait till available, etc.

Modern Linux Threads (continued)

- **Multiple threads can be executing *in kernel* at same time**
  - In various states of activity

- **Multiple processors can be executing *in kernel* at the same time**
  - Handling interrupts
  - In process context on behalf of some thread

- **Made possible by**
  - One kernel stack per thread
  - One interrupt stack per processor

Threads in Linux Kernel

- **Kernel has its own threads**
  - No associated *process context*
- **Supports concurrent activity within kernel**
  - Multiple devices operating at one time
  - Multiple application activities at one time
  - Multiple processors in kernel at one time
- **A useful tool**
  - Special kernel thread packages, synchronization primitives, etc.
  - Useful for complex OS environments

Questions?