CISC 3320

C12e: Examples of Operating Systems Threads

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Acknowledgement

- These slides are a revision of the slides provided by the authors of the textbook

Outline

- Design and Implementation of Operating System Examples
  - POSIX Threads
  - Windows Threads

Operating System Examples

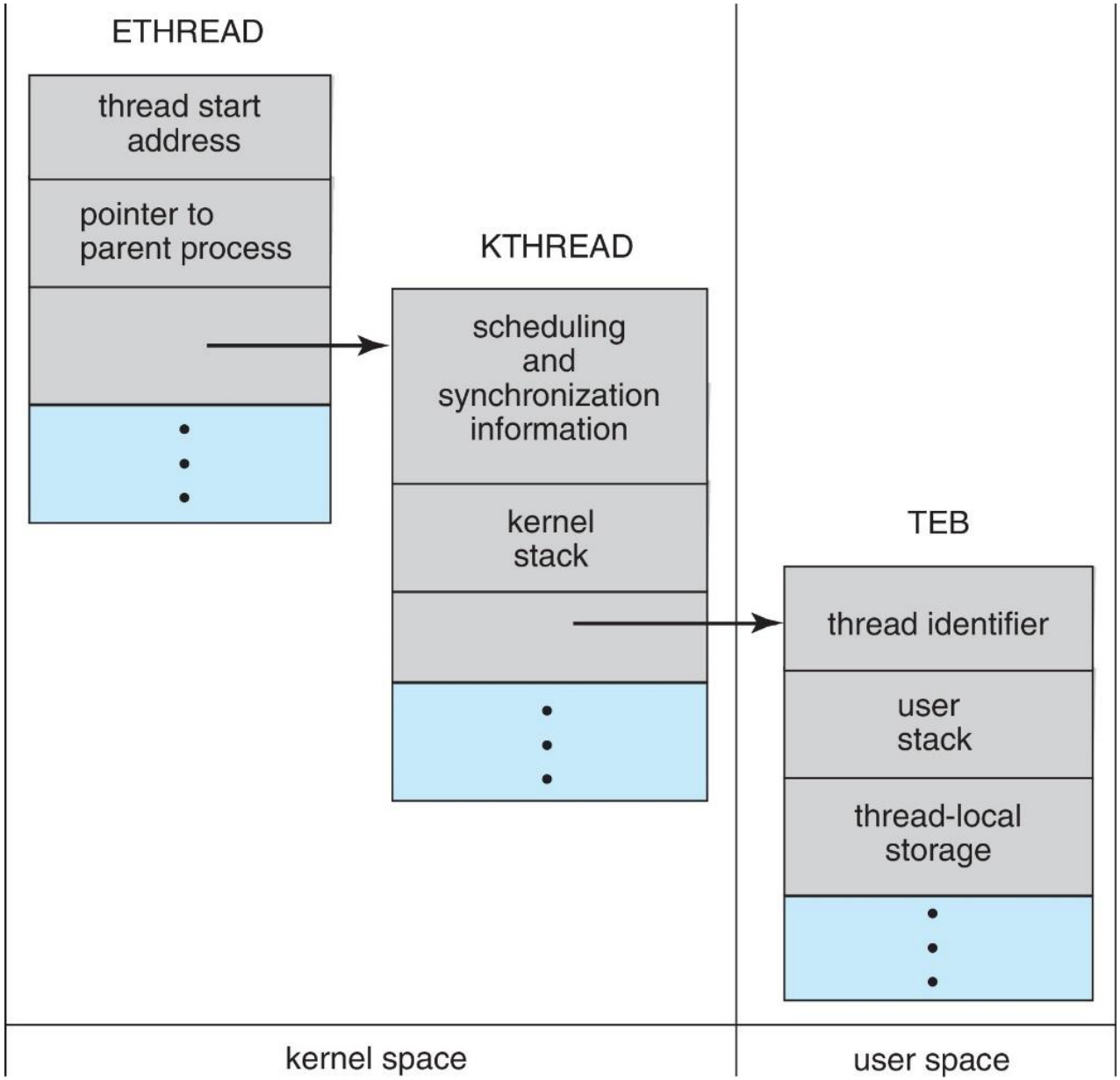
- Windows Threads
- Linux Threads

Windows Threads

- Windows API
  - Primary API for Windows applications
- Implements kernel-level threads
- Implements the one-to-one mapping (between user space and kernel space threads)
- Each thread contains
  - A thread id
  - Register set representing state of processor
  - Separate user and kernel stacks for when thread runs in user mode or kernel mode
  - Private data storage area used by run-time libraries and dynamic link libraries (DLLs)
- The register set, stacks, and private storage area are known as the **context** of the thread

Windows Threads: Data Structures

- The primary data structures of a thread include:
  - ETHREAD (executive thread block) – includes pointer to process to which thread belongs and to KTHREAD, in kernel space
  - KTHREAD (kernel thread block) – scheduling and synchronization info, kernel-mode stack, pointer to TEB, in kernel space
  - TEB (thread environment block) – thread id, user-mode stack, thread-local storage, in user space


ETHREAD
thread start address
pointer to parent process
KTHREAD
scheduling and synchronization information
kernel stack
TEB
thread identifier
user stack
thread-local storage
kernel space
user space

Linux Threads

- Linux refers to them as ***tasks*** rather than ***threads***
- Thread creation is done through **`clone()`** system call
- Both fork() and clone() create a child task
- **`clone()`** allows a child task to share the address space of the parent task
- Remark: Linux does not really distinguish between processes and threads

Linux Thread: Clone System Call

- Linux PCB, the task_struct structure contains pointers to other data structures
  - e.g., the list of open files, signal-handling information, and virtual memory.
- The fork() system call
  - When it is invoked, a new task is created, along with a copy of all the associated data structures of the parent process.
- The clone() system call
  - A new task is also created when the clone() system call is made.
  - However, rather than copying all data structures, the new task points to the data structures of the parent task, depending on the set of flags passed to clone().

Flags Control Behavior of the clone() System Call

- The flag controls whether data in the task_struct are shared or copied between child and parent tasks

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

Questions?

- Design and implementation of threads in Window
- Design and implementation of threads in Linux