LECTURE-18
THREADS

Threads

- In most traditional OS, each process has an address space and a single thread of control.
- It is desirable to have multiple threads of control sharing one address space but running in quasi-parallel.

Introduction to threads

- Thread is a lightweighted process.
- The analogy: thread is to process as process is to machine.
- Each thread runs strictly sequentially and has its own program counter and stack to keep track of where it is.
- Threads share the CPU just as processes do: first one thread runs, then another does.
- Threads can create child threads and can block waiting for system calls to complete.

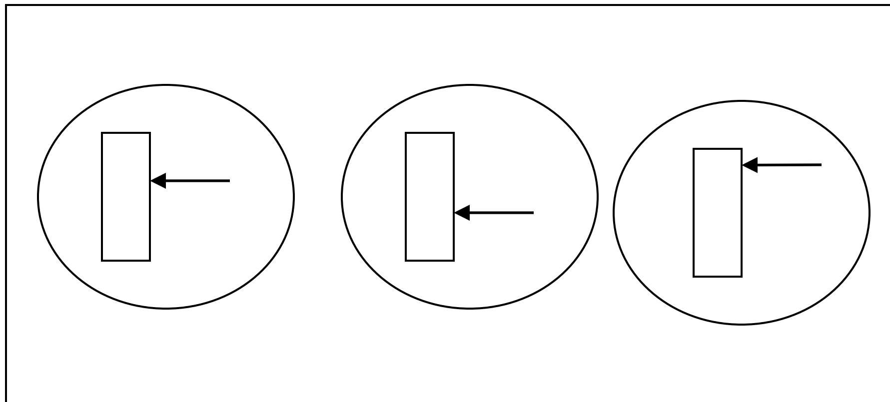
Cont…

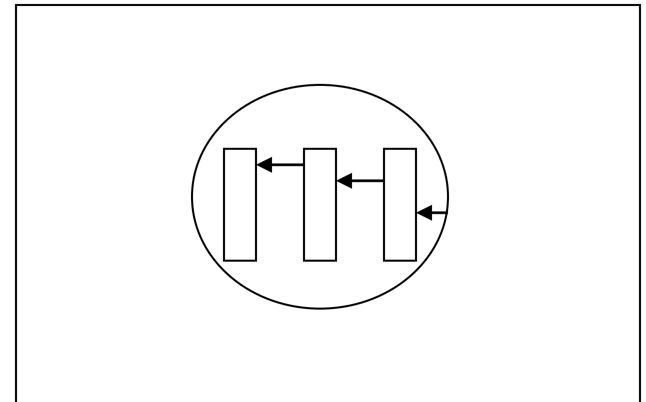
- All threads have exactly the same address space. They share code section, data section, and OS resources (open files & signals). They share the same global variables. One thread can read, write, or even completely wipe out another thread's stack.
- Threads can be in any one of several states: running, blocked, ready, or terminated.

Cont..

- There is no protection between threads:
- (1) it is not necessary (2) it should not be necessary: a process is always owned by a single user, who has created multiple threads so that they can cooperate, not fight.

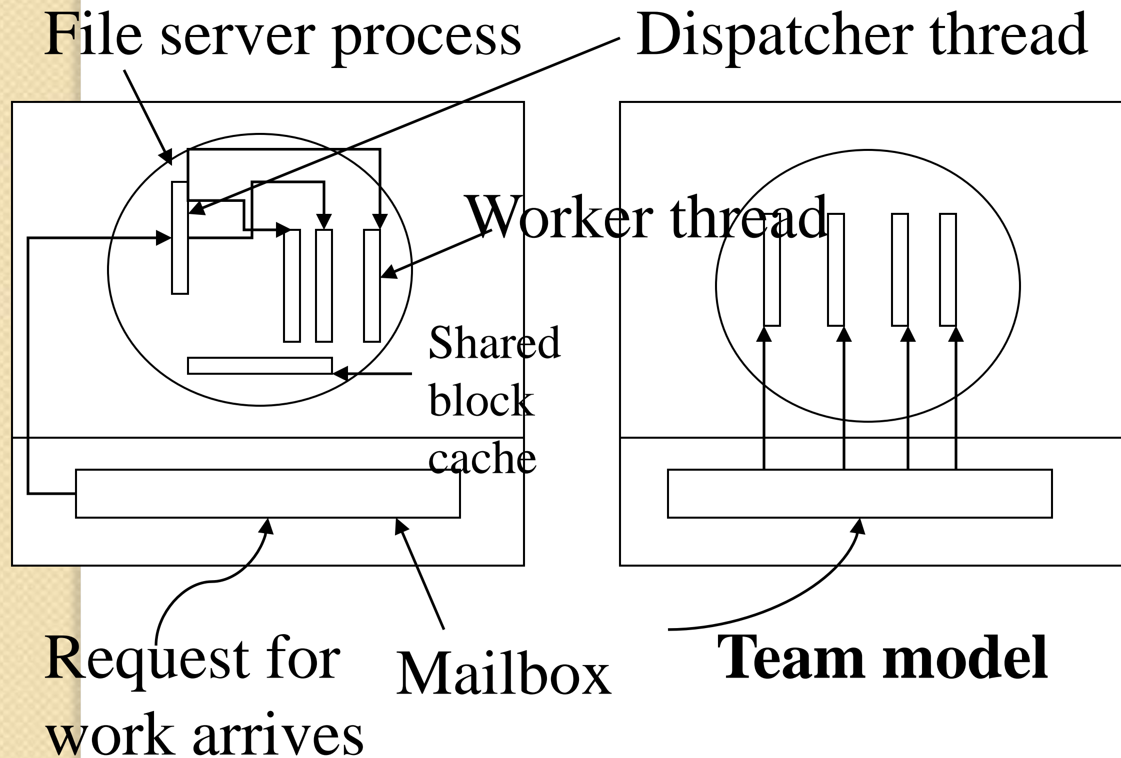
Threads



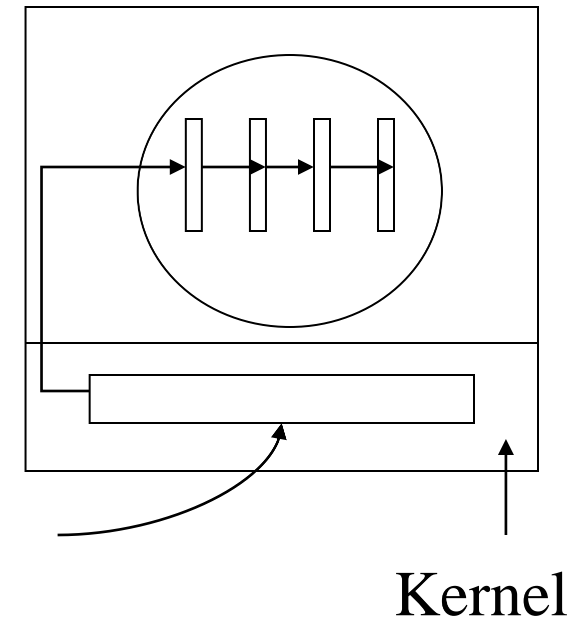


Thread usage

Dispatcher/worker model



Pipeline model



Advantages of using threads

1. Useful for clients: if a client wants a file to be replicated on multiple servers, it can have one thread talk to each server.
2. Handle signals, such as interrupts from the keyboard. Instead of letting the signal interrupt the process, one thread is dedicated full time to waiting for signals.
3. Producer-consumer problems are easier to implement using threads because threads can share a common buffer.
4. It is possible for threads in a single address space to run in parallel, on different CPUs.

Design Issues for Threads Packages

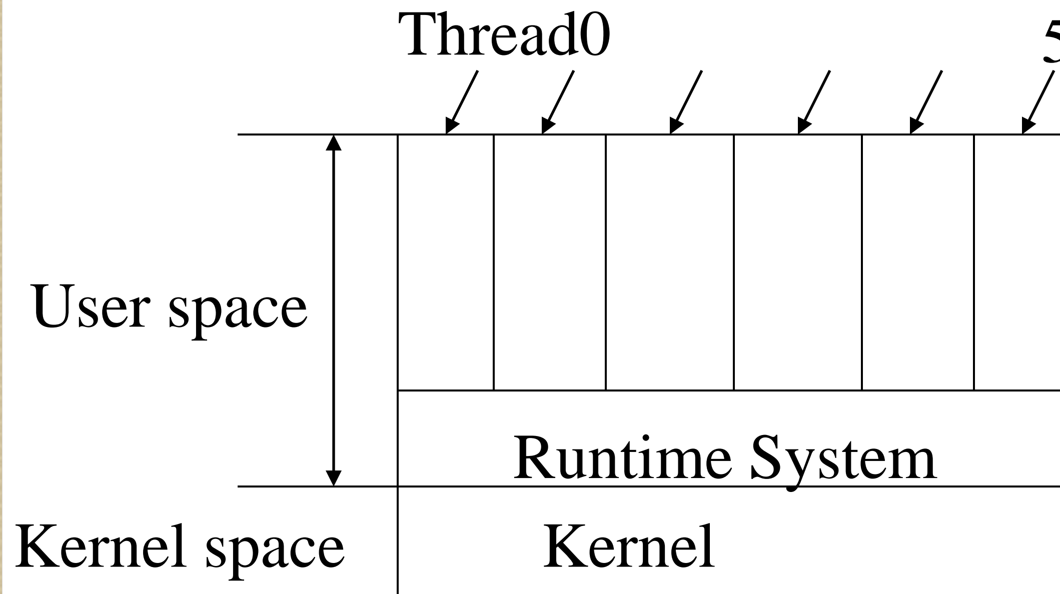
- A set of primitives (e.g. library calls) available to the user relating to threads is called a **thread package**.
- **Static thread**: the choice of how many threads there will be is made when the program is written or when it is compiled. Each thread is allocated a fixed stack. This approach is simple, but inflexible.
- **Dynamic thread**: allow threads to be created and destroyed on-the-fly during execution.

Mutex

- If multiple threads want to access the shared buffer, a mutex is used. A mutex can be locked or unlocked.
- Mutexes are like binary semaphores: 0 or 1.
- Lock: if a mutex is already locked, the thread will be blocked.
- Unlock: unlocks a mutex. If one or more threads are waiting on the mutex, exactly one of them is released. The rest continue to wait.
- Trylock: if the mutex is locked, Trylock does not block the thread. Instead, it returns a status code indicating failure.

Implementing a threads package

- **Implementing threads in user space**



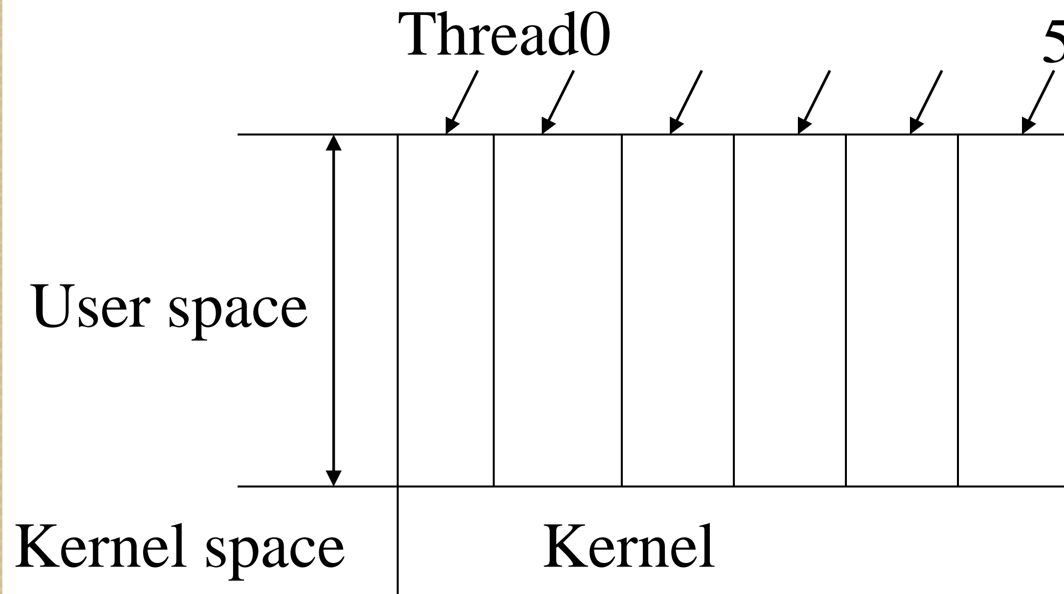
Advantage

- User-level threads package can be implemented on an operating system that does not support threads. For example, the UNIX system.
- The threads run on top of a runtime system, which is a collection of procedures that manage threads. The runtime system does the thread switch. Store the old environment and load the new one. It is much faster than trapping to the kernel.
- User-level threads scale well. Kernel threads require some table space and stack space in the kernel, which can be a problem if there are a very large number of threads.

Disadvantage

- Blocking system calls are difficult to implement. Letting one thread make a system call that will block the thread will stop all the threads.
- Page faults. If a thread causes a page faults, the kernel does not know about the threads. It will block the entire process until the page has been fetched, even though other threads might be runnable.
- If a thread starts running, no other thread in that process will ever run unless the first thread voluntarily gives up the CPU.
- For the applications that are essentially CPU bound and rarely block, there is no point of using threads. Because threads are most useful if one thread is blocked, then another thread can be used.

Implementing threads in the kernel



Cont…

- The kernel knows about and manages the threads. No runtime system is needed. When a thread wants to create a new thread or destroy an existing thread, it makes a kernel call, which then does the creation and destruction.
- To manage all the threads, the kernel has one table per process with one entry per thread.
- When a thread blocks, the kernel can run either another thread from the same process or a thread from a different process.

Scheduler Activations

- Scheduler activations combine the advantage of user threads (good performance) and kernel threads.
- The goals of the scheduler activation are to mimic the functionality of kernel threads, but with the better performance and greater flexibility usually associated with threads packages implemented in user space.
- Efficiency is achieved by avoiding unnecessary transitions between user and kernel space. If a thread blocks, the user-space runtime system can schedule a new one by itself.

Cont..

- Disadvantage:

  Upcall from the kernel to the runtime system violates the structure in the layered system.

ASSIGNMENT

- Differentiate between user level and kernel level threads.