Threads

- Thread creation 10 times faster than process in UNIX (TEVA87)
- Communication between processes requires kernel intervention for protection

Use of threads in single-user multiprocessing system:

- Foreground and background work
- Asynchronous elements (like backup)
- Speed of execution
- Modular program structure

Thread operations

1. Spawn
2. Block
3. Unblock
4. Finish

Windows Process & Thread Management

1. An Application

Consists of one or more
process that has:

- Unique process identifier
- Virtual address space
- Executable code
- Open handles to system objects
- Security context
- Environment variables
- Priority class
- Minimum and Maximum working set sizes
- At least one thread of execution (primary thread)

2. A Thread

- All threads within a process share its virtual address space and system resources
- Each thread maintains:
  - Unique thread identifier
  - Exception handlers
  - Scheduling priority
  - Local storage
  - Set of structures so system save in it thread context (until scheduled)

3. A Job Object

- Allows groups of processes to be managed as a unit - control their attributes
- They are:
  - Namable
  - Securable
  - Shareable
- Operations on it affect all processes like:
  - Enforcing limits (working set size, process priority)
  - Terminating all processes

4. A Thread Pool

A collection of worker threads that execute asynchronous callbacks on behalf of the application
Used to:
- Reduce number of application threads
- Provide management for worker threads

5. A Fiber

Unit of execution that is manually scheduled by the application (doesn't provide advantage over a well-designed multi-threaded application)
- Run in the context of the threads that scheduled them
- A thread schedule -> multiple fibers
- Can make it easier to port applications designed to schedule their own threads
- Associated with them:
  - Thread stack
  - Subset of thread registers
  - Fiber data provided upon its creation

User-mode Scheduling (UMS)

Lightweight mechanism for apps to schedule their own threads
- Switch between UMS threads without involving system scheduler
- Each has:
  - Its own thread context
- More efficient than thread pools for short duration work items with few system calls
- Useful for high performance apps that need to efficiently run many threads concurrently on multiprocessor or multicore systems

Backgrounds Tasks and App Lifecycle

Windows 8-10:
- Developers are responsible for managing the state of their apps
- Users have full control over the lifetime of a process

New Metro Interface:

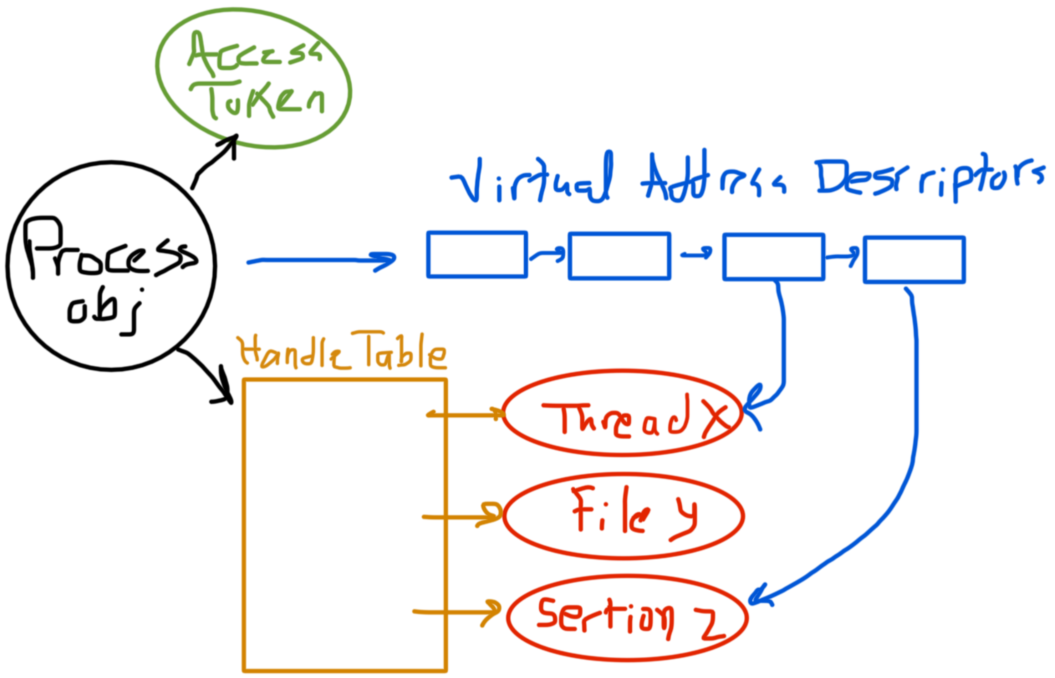
- Windows takes over the process lifecycle of an app
- Limited number of apps can run alongside the main app in the Metro UI using Snap View
- Only one store app cab run at a time

Foreground App in Metro Interface:

- Has access to all of the processor, network, and disk resources available for user
- All other apps are suspended and have no access with it

Suspended Mode:
- An event should be triggered to store state of user info
  responsibility of app developer

An app may be **terminated** sooo:
- You need to save app state when it's suspended sooo
  when app returns to foreground an event to restore app state from memory will be triggered



Processes Characteristics:

- Implemented as objects
- Created as a new process or a copy of an existing one
- May have one or more threads
- Process and thread objects have built-in synchronize capabilities

Process Object

An entity corresponding to a user job or application that owns resources

Attributes

Process ID	
Security Descriptor	- Who created it - who can gain access to or use it - who is denied access
Base Priority	Baseline execution priority for its threads
Default Processor Affinity	Set of processors on which its threads can run
Quota Limits	Max amount of (non)paged system memory, paging file space, processor time
Execution Time	Total amount of times all threads have executed
I/O Counters	Record number and type of I/O operations performed
VM operation counters	Record number and types of Virtual Memory operations performed
Exception/debugging ports	Connected to environment subsystem and debugger processes
Exit status	Reason for process termination

Thread Object
A dispatchable unit of work that executes sequentially and is interruptible

Attributes

Thread ID	
Thread Context	Set of register values - others that defines execution state
Dynamic priority	Execution priority at any given monent
Base priority	Lower limit of dynamic priority
Thread processor affinity	Set of processors on which it can run on - subset of process default processor affinity
Thread execution time	Total amount of execution in user/kernel modes
Alert status	Indicates whether a waiting thread can execute an asynchronous procedure call
Suspension count	Suspended without being resumed
Impersonation token	Temporary access token that allows the thread to perform operations on behalf of another process (used by subsystems)
Termination port	Interprocess communication channel where process manager sends a msg when thread terminates (used by subsystems)
Thread exit status	Reason for thread termination

States
1. Ready: may be scheduled for execution.
- The Kernel dispatcher keeps track of all ready threads and schedules them in priority order.

2. Standby: has been selected to run next on a particular processor.
- The thread waits in this state until that processor is made available.
- If the standby thread's priority is high enough it may move to running while the running thread may be preempted to ready state.

3. Running

4. Waiting: A thread enters the Waiting state when:
1. blocked on an event (ex I/O)
2. it voluntarily waits for synchronization purposes
3. an environment subsystem directs the thread to suspend itself.

When the waiting condition is satisfied the thread moves to the Ready state if all of its resources are available.

5. Transition: if ready to run, but the resources are not available.
For example, the thread's stack may be paged out of memory. When the resources are available, the thread goes to the Ready state.

6. Terminated

Linux

Linux Task (process)

Represented by task_struct
The structure has information in categories:

- **State:** The execution state of the process (executing, ready, suspended, stopped, zombie).
- **Scheduling information:**
  - A process can be:
    - normal
    - real time: scheduled before normal
    - has a priority.
  - A reference counter keeps track of the amount of time a process is allowed to execute.
- **Identifiers:**
  - PID
  - user identifiers.
  - group identifiers: used to assign resource access privileges to a group of processes.
- **Interprocess communication:** Linux supports the IPC mechanisms found in UNIX SVR4
- **Links:** to
  - its parent process
  - its siblings
  - all of its children
- **Times and timers:** Includes process
  - creation time
  - amount of processor time so far consumed
  - one or more interval timers: defines it by means of a system call; as a result, a signal is sent to the process when the timer expires. A timer may be single use or periodic.
- **File system:**
  - pointers to any files opened by this process
  - pointers to the current and the root directories for this process
- **Address space:** Defines the virtual address space assigned to this process
- **Processor-specific context:** The registers and stack information that constitute the context of this process

execution states of a process:

- **Running:** executing or ready to execute
- **Interruptible:** blocked state (waiting for event to end or availability of resource or a signal)
- **Uninterruptible:** another blocked state but directly waiting on hardware conditions so wont handle any signals
- **Stopped:** halted and can only resume by positive action from another process (ex process being debugged can be put into stopped state)
- **Zombie:** process has been terminated but for some reason still must have its task_struct in the process table

Linux Threads

- No distinction between threads and processes
- Multiple User level threads that constitute a single user level process —mapped to—> kernel level processes that share same group ID
  - This allows shared resources so avoid the need for context switch when the scheduler switches among processes in the same group
- New process is created by copying attributes of current process
- New process (to turn into a thread) is cloned so it shares resources with current process.

> Threads created using clone() instead of fork()

Linux Namespaces

A namespace enable a process to have a different view of the system than processes having other namespaces
Six namespaces

- Mnt (mount)
    - Provide different speech view of filesystem hierarchy
- PID
    - Isolate process Id space so that processes in different PID namespaces can have the same process Id
    - Used in Checkpoint/Restore In Userspace (CRIU)
        - You can freeze a running app or part of it and check it to hard drive a collection of files
- IPC (interprocess communication)
    - Isolates certain IPC resources such as semaphores, POSIX message queues and more. So that concurrency mechanisms can be employed
- NET (network)
    - Provide isolation for the system resources associated with networking. Each NET namespace has
        - IP addresses
        - IP routing tables
        - Port numbers
    - At any given time a network device belongs only to one net namespace
    - A socket can belong to only one namespace
- UTS
- User