Department of Physics,
Computer Science & Engineering

CPSC 410 – Operating Systems I

Chapter 4:Threads

Keith Perkins

Adapted from original slides by Dr. Roberto A. Flores

Chapter 4 Topics

- Processes & Threads
  - Approaches, States, Processes vs. Threads, Benefits
- Types of Threads
  - User-level, Kernel-level, Hybrid
- Multi-core & Multi-threading
  - Performance, Winners
- Examples
  - ~~Windows 7~~
  - Linux

Multi-Threading

- A process has 2 characteristics
  - Resource ownership
    - virtual address space holding the process image
  - Scheduling & Execution
    - execution state (Running, Ready, etc.) a
    - dispatching priority

- Multi-threading
  - ability of an OS to support multiple concurrent paths of execution within a single process

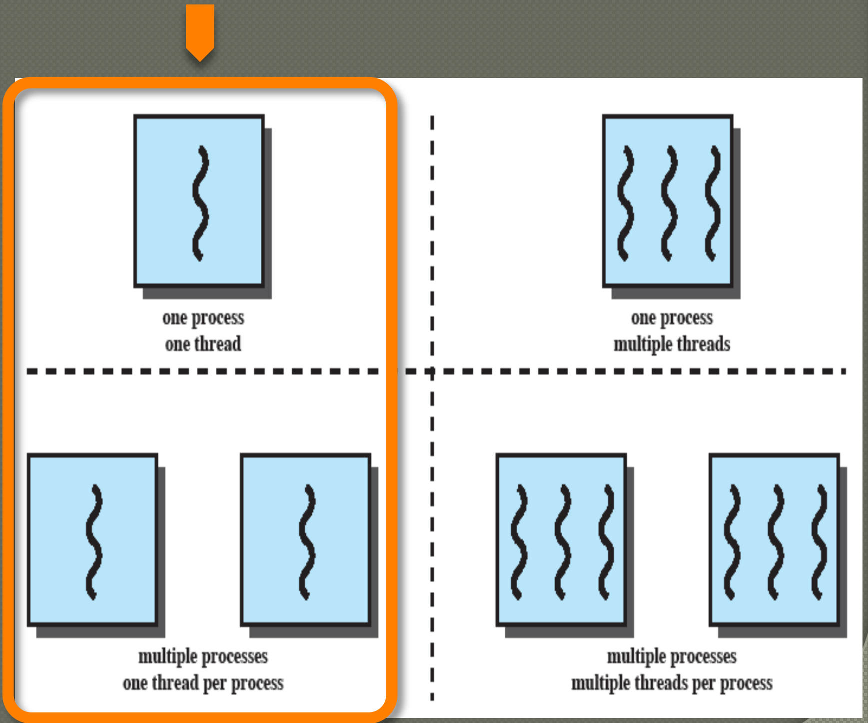
Multi-Threading

- ability of an OS to support multiple concurrent paths of execution within a single process

Approaches

- Single-Threaded

1 thread per process…



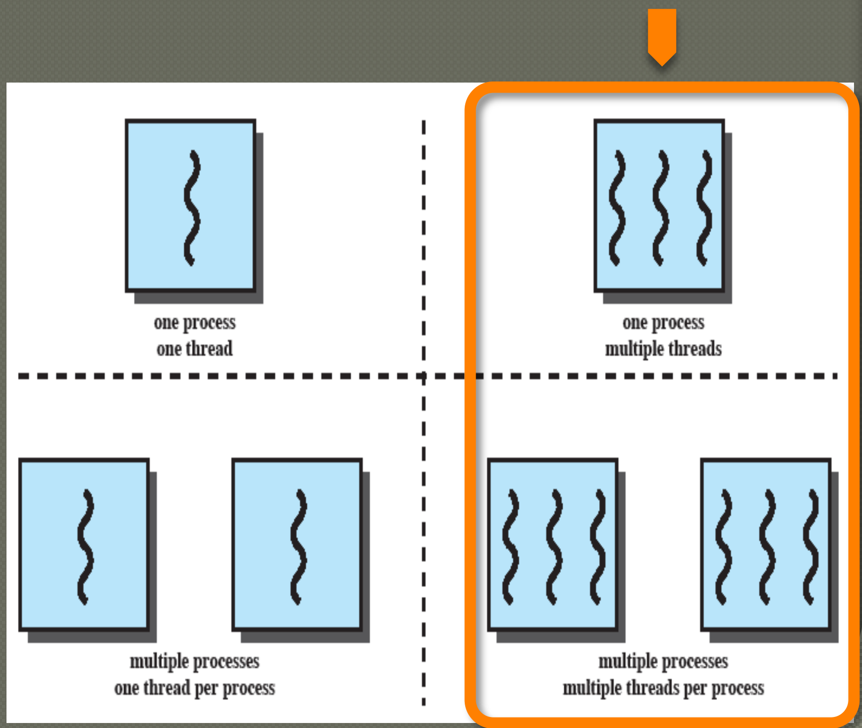
Multi-Threading

- ability of an OS to support multiple concurrent paths of execution within a single process

Approaches

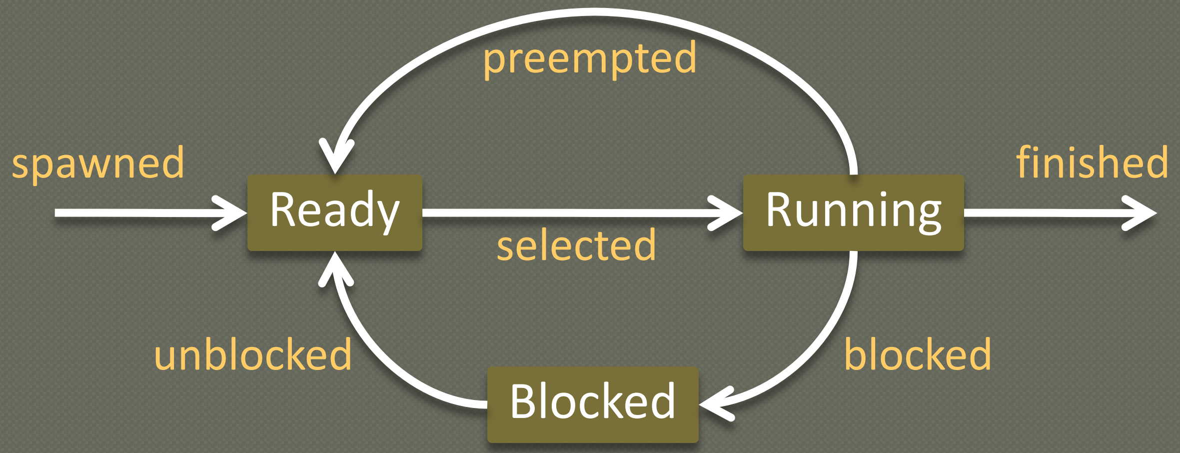
- Multi-Threaded

1+ threads per process



Threads

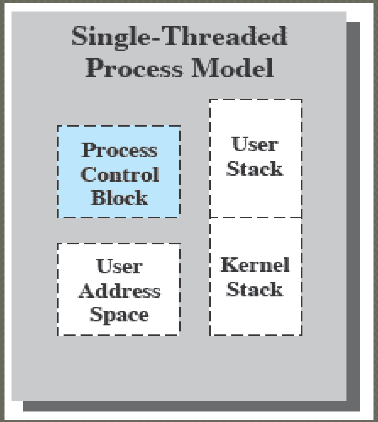
- States



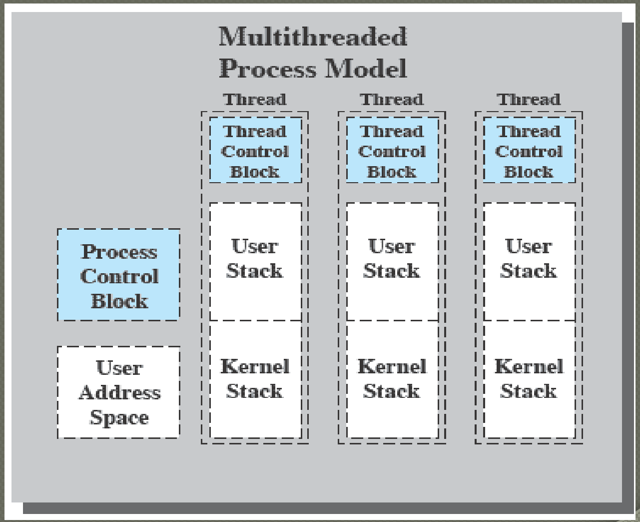
- That all thread info is stored in a process means that…
  - suspending/terminating a process suspends/terminates all its threads.

Processes vs. Threads

- A process has data structures to run in 1 thread
  - PCB, memory space, user/kernel stacks



- A thread duplicates (most of) them to run in its own terms
  - except PCB (replaced by TCB) & memory (which is shared among threads)
  - All in the host process space

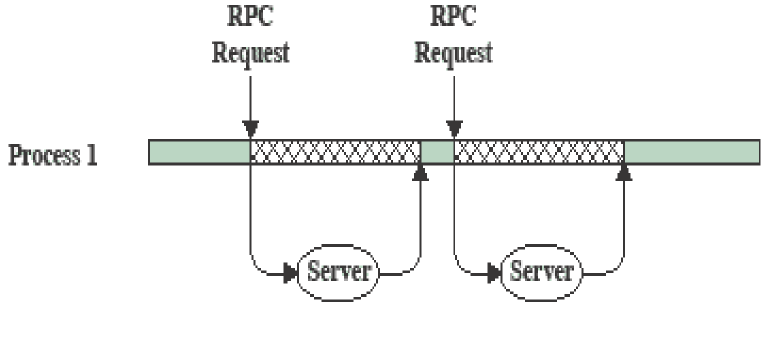


Benefits of Threads

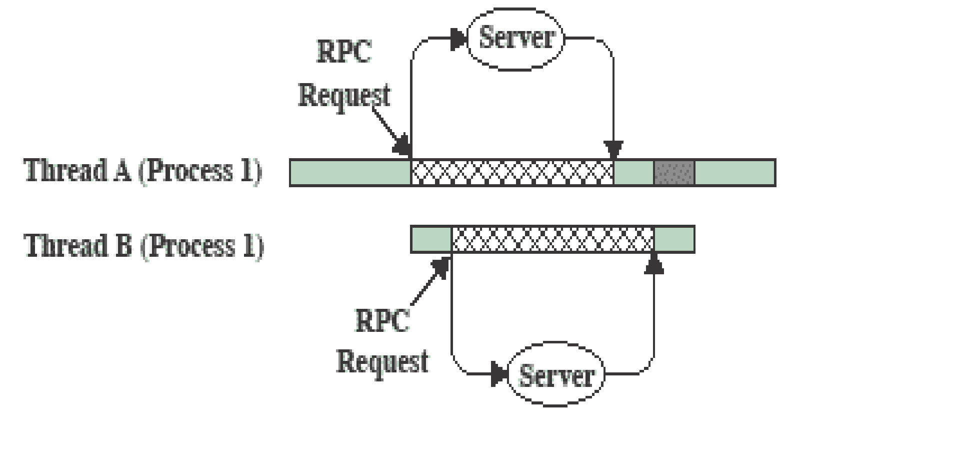
- Takes less time to create a new thread than a process
- Less time to terminate a thread than a process
- Switching between two threads takes less time than switching between processes
- Threads enhance efficiency in communication between programs

Threads

- Why multi-threading?
  - improve efficiency in communications between programs



(a) RPC Using Single Thread



(b) RPC Using One Thread per Server (on a uniprocessor)

Threads

- Why multi-threading?
  - Improves efficiency in asynchronous execution
    Responsiveness
    - 1 thread handles GUI, another background processing
  - Speed of execution (depends on if you can parallelize tasks)
    - e.g., in video games: rendering, AI, physics
  - Modular programming (SE)
    - programmers design modular code

Thread synchronization
- Needed because:
  - all threads of a process share the same memory/resources
    - i.e., any changes by one thread affects other threads in process

My first thread

```
//see thread1 project
// The function we want to execute on the new thread.
void task1()
{
    imaglobal--;
}
int main()
{
    // Constructs the new thread and runs it.
    //Does not block execution.
    thread t1(task1);

    //hit ctrl-f11 in debug see that the following
    //equals 3 assembly instructions, interrupt can
    //happen after any of those three
    imaglobal++;

    //Makes the main thread wait for the new thread to
    //finish execution, therefore blocks its own execution.
    t1.join();
}
```

Under the hood

```
    //hit ctrl-f11 in debug see that the following
    //equals 3 assembly instructions, interrupt can
    //happen after any of those three
    imaglobal++;
013A87BF  mov         eax,dword ptr [imaglobal (013B3428h)]     get imaglobal
013A87C4  add         eax,1                                     increment
013A87C7  mov         dword ptr [imaglobal (013B3428h)],eax     put it back

    // Makes the main thread wait for the new thread to finish execution, there
    t1.join();
013A87CC  lea         ecx,[t1]
013A87CF  call        std::thread::join (013A1294h)
}
013A87D4  mov         dword ptr [ebp-4],0FFFFFFFFh
```

Watch 1

Name	Value	Typ
&imaglobal	0x013b3428 {thread_1.exe!int imaglobal} {-1}	int

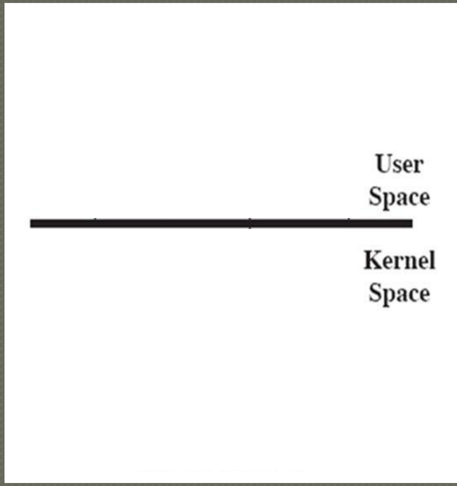
where in memory

Chapter 4 Topics

- Processes & Threads
  - Approaches, States, Processes vs. Threads, Benefits
- Types of Threads
  - User-level, Kernel-level, Hybrid
- Multi-core & Multi-threading
  - Performance, Winners
- Examples
  - ~~Windows 7~~
  - Linux

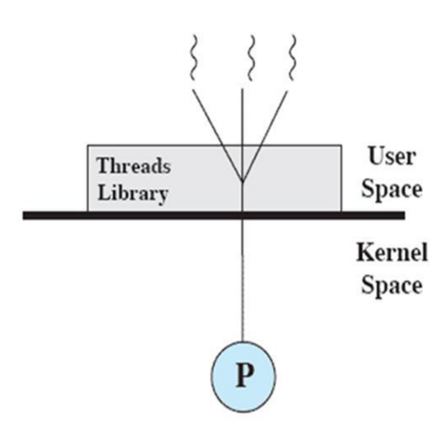
Types of Threads

- Types
  - Thread management is done at
    - The application level (User-level threads)
      - Library (pthreads library)
      - Kernel is aware of process only not its multiple threads (library manages them)
    - The OS level (Kernel-level threads)
      - processes use an OS API to access threads
      - approach favored by MS Windows



Types of Threads

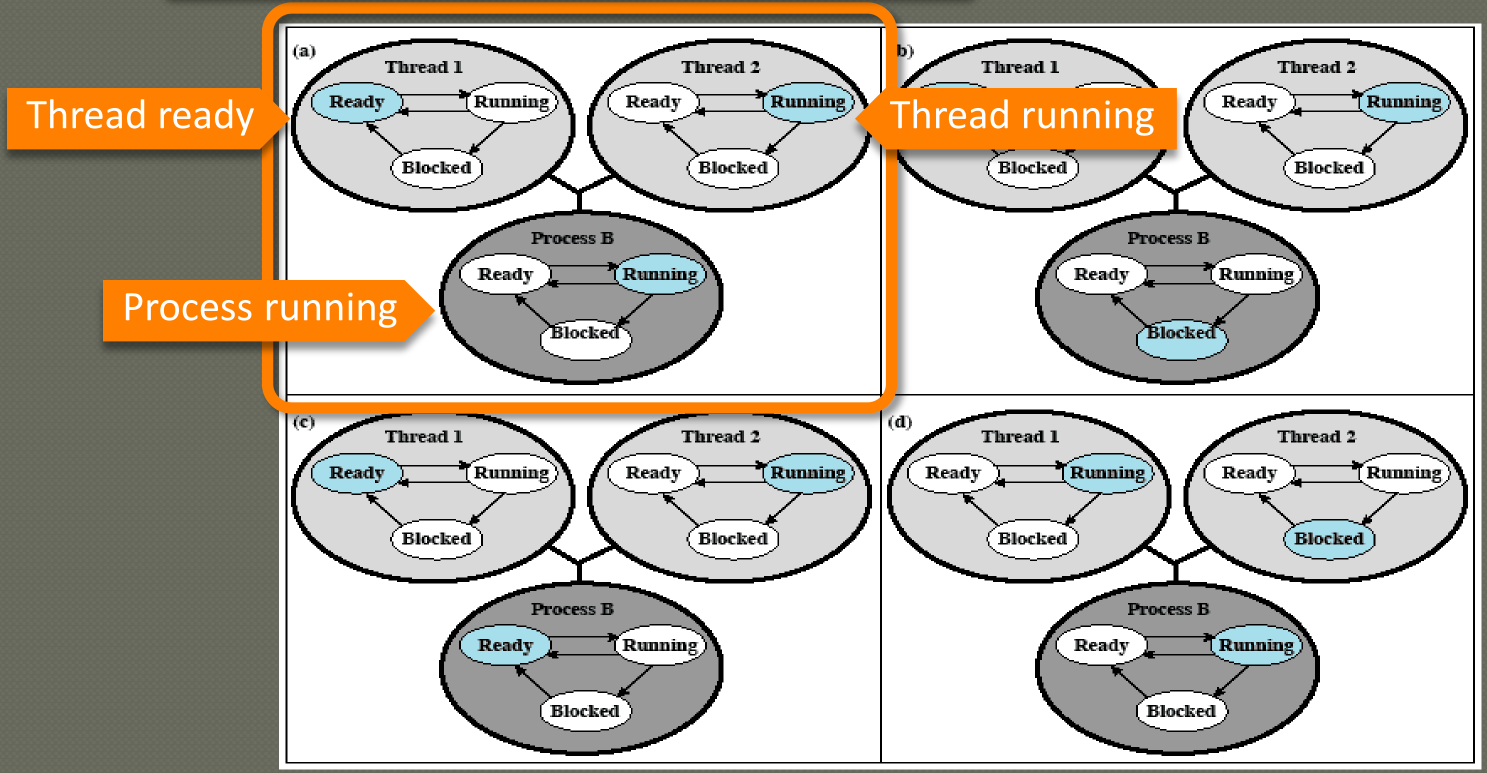
- User-level Threads (ULT)
  - All management done by a threads library
    - Program & library run in a one-thread process.
    - A program spawning a new thread (within the same process) invokes the library, which creates data structures for the thread.
    - When the library is given control, it executes threads using its own scheduling algorithm, saving threads states as it switches execution from one to the next.



(a) Pure user-level

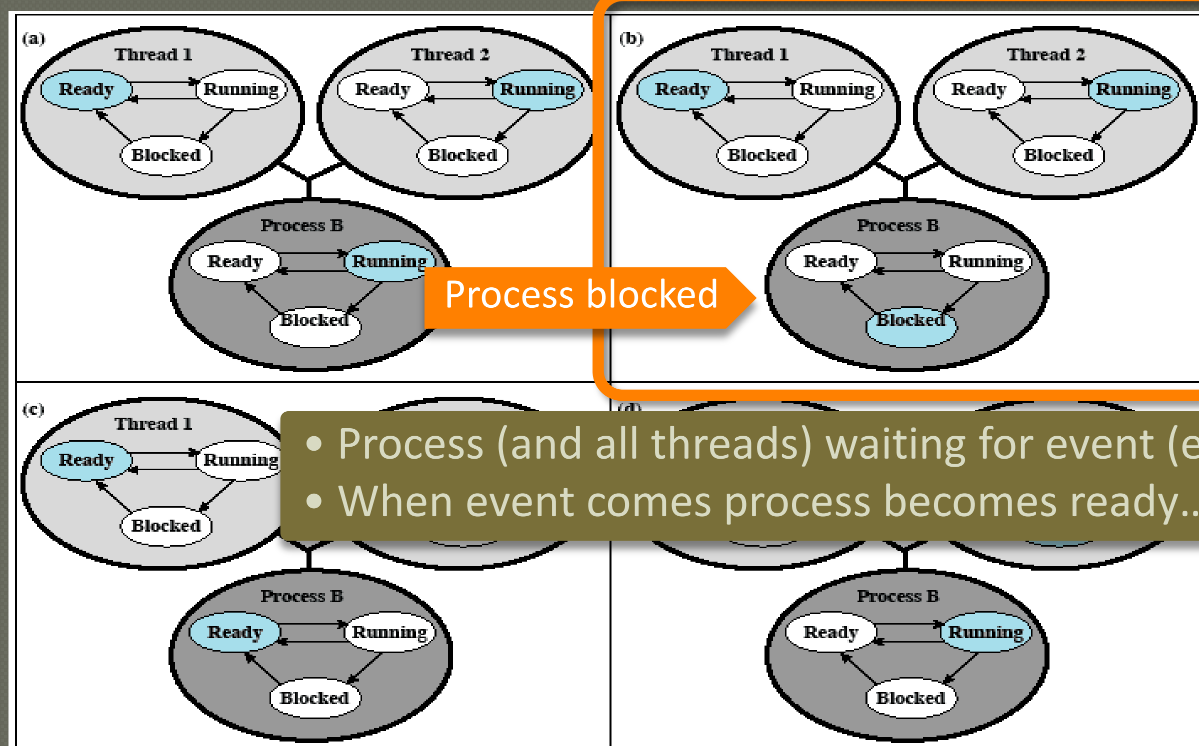
Types of Threads

- User-level Threads (ULT) in context
  - Thread requests system call (e.g., I/O)
  - Entire process is blocked by OS



Types of Threads

- User-level Threads (ULT) in context

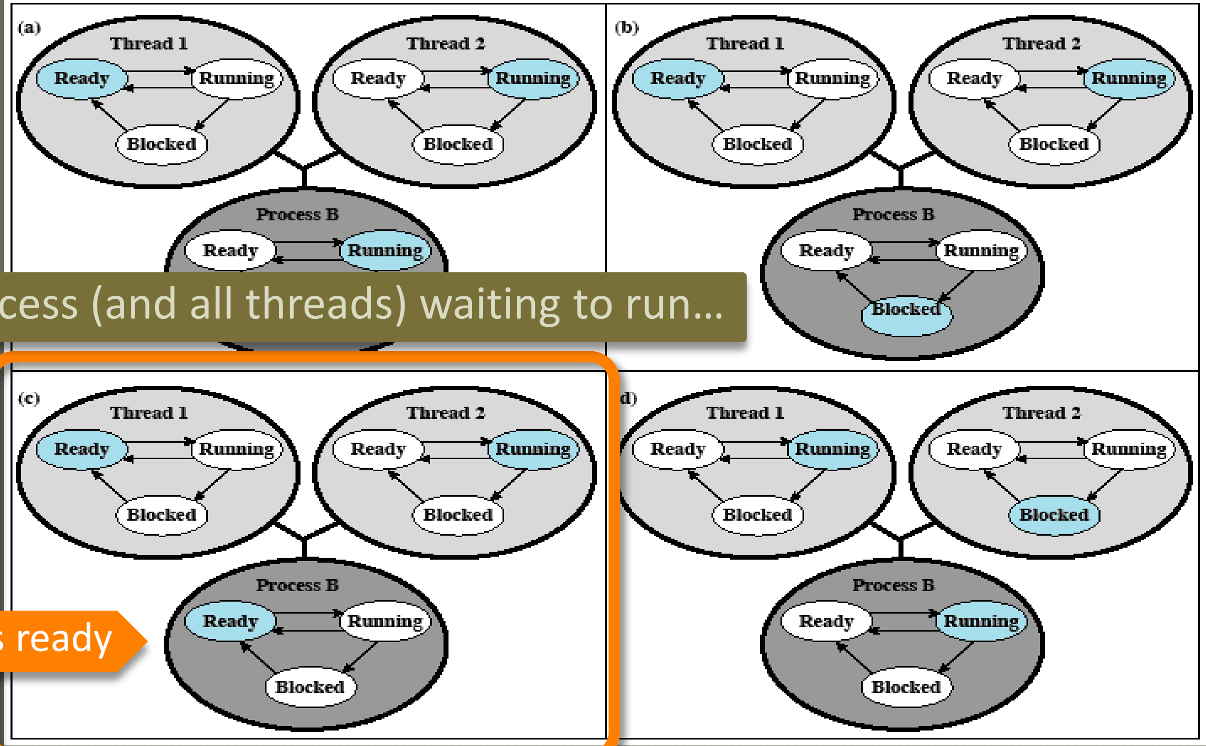


Process blocked

- Process (and all threads) waiting for event (e.g., I/O interrupt)
- When event comes process becomes ready…

Types of Threads

- User-level Threads (ULT) in context

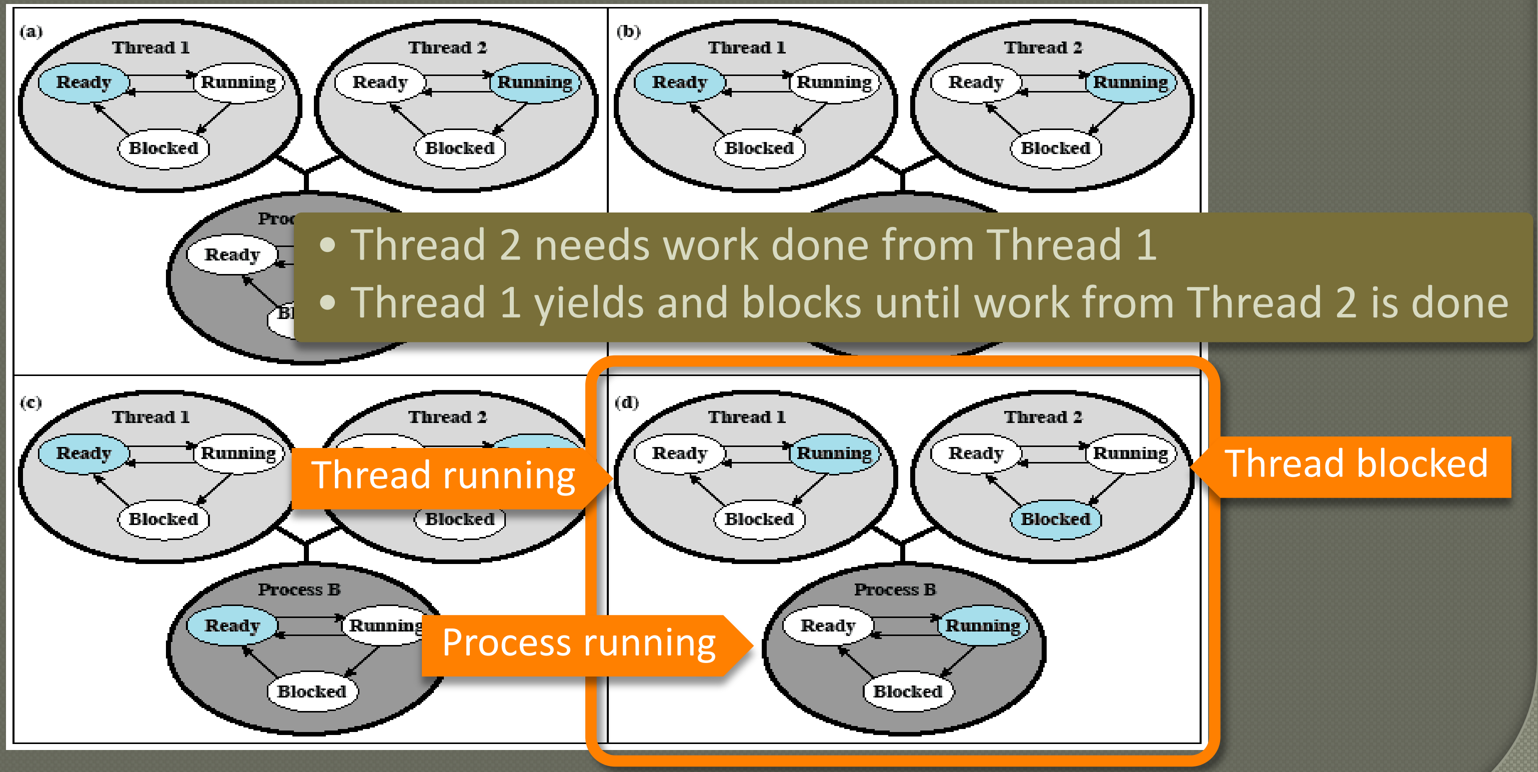


- Process (and all threads) waiting to run…

Process ready

Types of Threads

- User-level Threads (ULT) in context

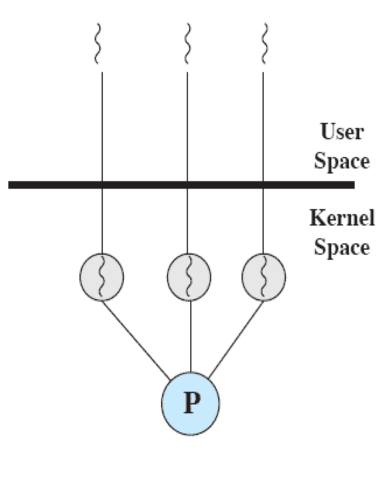


Types of Threads

- User-level Threads (ULT)
  - Advantages
    - Switching threads does not require Kernel mode.
    - Scheduling can be application-specific.
    - Threads can run on any OS.
  - Disadvantages
    - If a thread requests a system call, then the process (and all threads) are blocked.
    - ULT cannot take advantage of (OS scheduled) multi-processing.
  - Overcoming Disadvantages
    - Jacketing (technique)
      - Application level code sees if IO device is busy, if so it calls the threads library to switch to another thread

Types of Threads

- Kernel-level Threads (KLT)
  - All management done by OS
    - Threads like processes (with shared memory)
  - Advantages
    - Threads from the same process can…
      - …run on multiple processors.
      - …keep running if one gets blocked.
  - Disadvantages
    - Transferring control from one thread to another (even from the same process) requires a switch to kernel mode.

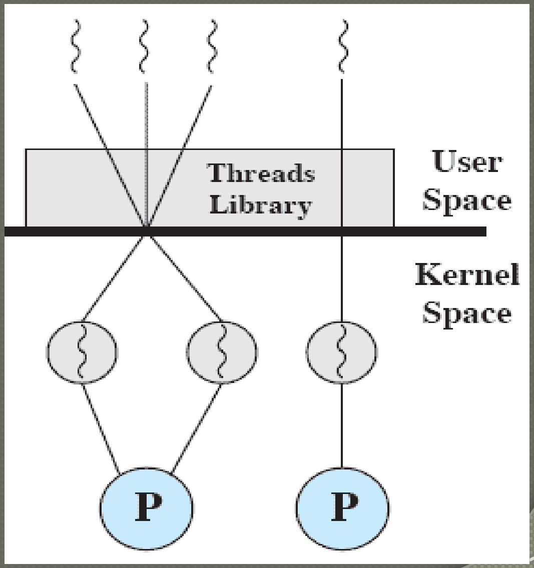


(b) Pure kernel-level

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

Types of Threads

- Combined Approach (Hybrid)
  - All management done by a threads library, which takes advantage of OS multithreading capabilities
    - (if library properly implemented) Threads from the same process could run in parallel on multiple processors.
      - Combines advantages of ULT & KLT…
        - Scheduling can be application-specific.
        - Threads could run on any OS, and on multiple processors.
        - Threads (same process) are not blocked.
      - …while minimizing disadvantages
        - Switching threads may not need Kernel mode.



Chapter 4 Topics

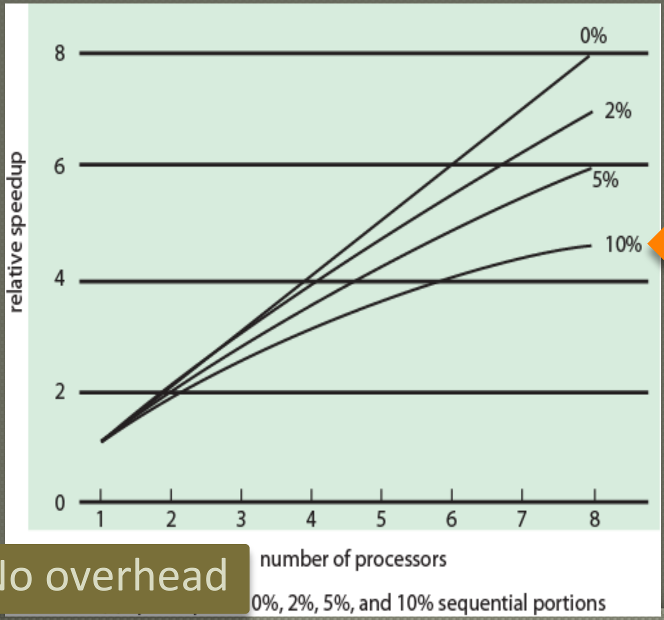
- Processes & Threads
  - Approaches, States, Processes vs. Threads, Benefits
- Types of Threads
  - User-level, Kernel-level, Hybrid
- Multi-core & Multi-threading
  - Performance, Winners
- Examples
  - ~~Windows 7~~
  - Linux

Multi-core & Multi-threading

- Performance
  - Benefits depend on program design
  - Amdahl's law

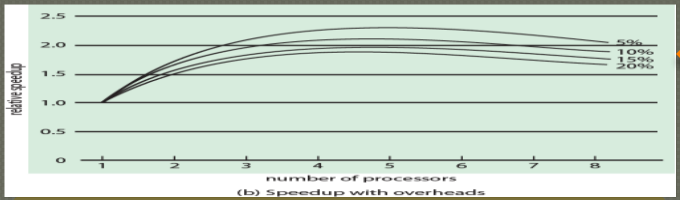
f = % parallelism

$$\text{Speedup} = \frac{\text{Time to run in 1 processor}}{\text{Time to run on N parallel processors}} = \frac{1}{(1 - f) + (f / N)}$$



f=90%, speedup=4.7

No overhead



f=90%, speedup=1.9

With synchronization overhead

Multi-core & Multi-threading

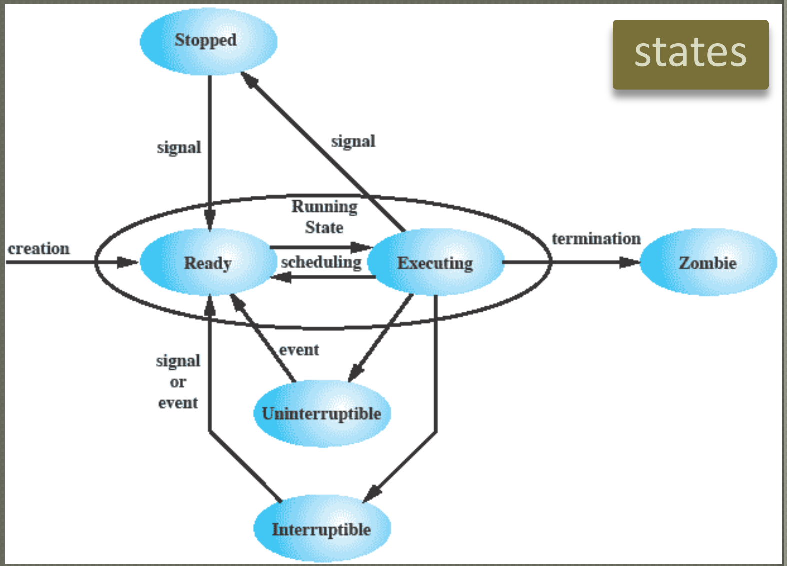
- Highest Gains
  - Multithreaded native applications
    - Programs with a small number of highly threaded processes
  - Multi-process applications
    - Programs with many single-threaded processes
      - e.g., Database management systems (transactions)
  - Java applications
    - Innate thread support
  - Multi-instance applications
    - Multiple instances of the same program running in parallel

Chapter 4 Topics

- Processes & Threads
  - Approaches, States, Processes vs. Threads, Benefits
- Types of Threads
  - User-level, Kernel-level, Hybrid
- Multi-core & Multi-threading
  - Performance, Winners
- Examples
  - ~~Windows 7~~
  - Linux

Linux

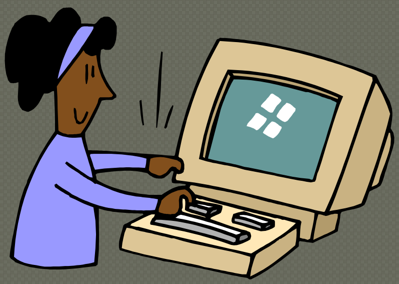
- Processes & Threads
  - No difference in Linux (also !priority).
  - ULT are mapped into KLT processes.
  - New processes are created…
    - …by spawning | cloning
      - ID of parent is copied
      - all resources are shared
      - parent/child have separate stacks
  - When switching processes
    - If they have same ID, then Linux runs the instructions of incoming process (without switching address space)



Linux Threads

- Linux does not recognize a distinction between threads and processes
- User-level threads are mapped into kernel-level processes
- A new process is created by copying the attributes of the current process
- The new process can be *cloned* so that it shares resources
- The clone() call creates separate stack spaces for each process

Linux Clone () Flags



CLONE_CLEARID	Clear the task ID.
CLONE_DETACHED	The parent does not want a SIGCHLD signal sent on exit.
CLONE_FILES	Shares the table that identifies the open files.
CLONE_FS	Shares the table that identifies the root directory and the current working directory, as well as the value of the bit mask used to mask the initial file permissions of a new file.
CLONE_IDLETASK	Set PID to zero, which refers to an idle task. The idle task is employed when all available tasks are blocked waiting for resources.
CLONE_NEWNS	Create a new namespace for the child.
CLONE_PARENT	Caller and new task share the same parent process.
CLONE_PTRACE	If the parent process is being traced, the child process will also be traced.
CLONE_SETTID	Write the TID back to user space.
CLONE_SETTLS	Create a new TLS for the child.
CLONE_SIGHAND	Shares the table that identifies the signal handlers.
CLONE_SYSVSEM	Shares System V SEM_UNDO semantics.
CLONE_THREAD	Inserts this process into the same thread group of the parent. If this flag is true, it implicitly enforces CLONE_PARENT.
CLONE_VFORK	If set, the parent does not get scheduled for execution until the child invokes the *execve()* system call.
CLONE_VM	Shares the address space (memory descriptor and all page tables).

My first thread revisit

```
//see thread1 project
// The function we want to execute on the new thread.
void task1()
{
    imaglobal--;
}
int main()
{
    // Constructs the new thread and runs it.
    //Does not block execution.
    thread t1(task1);

    //hit ctrl-f11 in debug see that the following
    //equals 3 assembly instructions, interrupt can
    //happen after any of those three
    imaglobal++;

    //Makes the main thread wait for the new thread to
    //finish execution, therefore blocks its own execution.
    t1.join();
}
```

Chapter 4 Topics

- Processes & Threads
  - Approaches, States, Processes vs. Threads, Benefits
- Types of Threads
  - User-level, Kernel-level [illegible]
- Multi-core & Mul[illegible]ading
  - Performance, V[illegible]
- Examples
  - Windows 7
  - Linux

Done!