Lecture 12

CprE 308

February 4, 2015

Intro

Threads Implementations

- User Level Threads
- Kernel Level Threads
- Hybrid Implementations
- Performance Comparisons

What is a Thread?

- Execution context
  - Registers, Stack
- Thread execution context smaller than process execution context
- Any threads library should support the following operations:
  - `thread_create()`
  - `thread_exit()`
  - `thread_wait()`
  - `thread_yield()`

User Space

User Space Threads

- All the thread support provided by a user level library
- Kernel not even aware that the program has multiple threads
- User library also handles synchronization (mutexes, condition variables, etc.)

Implementing Threads in User Space

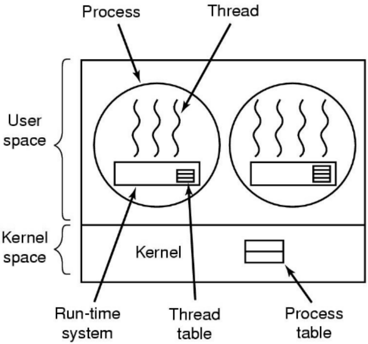

Figure 1:

A user-level threads package

User Space

- Good:
  - Thread related actions are fast (no system calls)
  - Can be used in OSes which don't implement threads

User Space

- Good:
  - Thread related actions are fast (no system calls)
  - Can be used in OSes which don't implement threads
- Bad:
  - System calls block and other threads will not be able to execute (pretty serious problem - why?)
  - Can't take advantage of a multiprocessor

Kernel

Implementing Threads in the Kernel

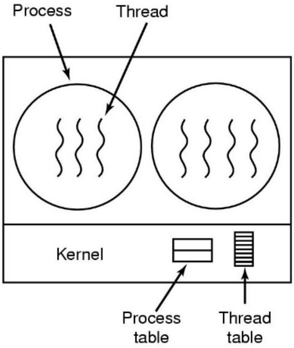

Figure 2:

A threads package managed by the kernel

Kernel Space

- Good:
  - Can take advantage of multiple processors
  - System call blocks only the thread which made the call

Kernel Space

- Good:
  - Can take advantage of multiple processors
  - System call blocks only the thread which made the call
- Bad:
  - Thread operations involve system calls (expensive)

Hybrid

Hybrid Implementations

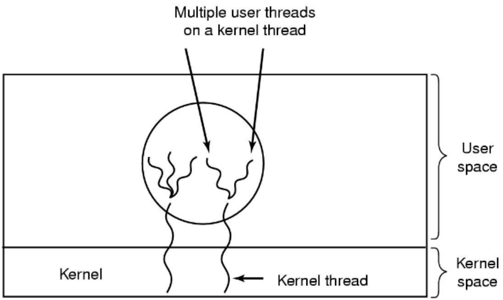

Figure 3:

Multiplexing user-level threads onto kernel-level threads

Comparisons

Processes and Threads

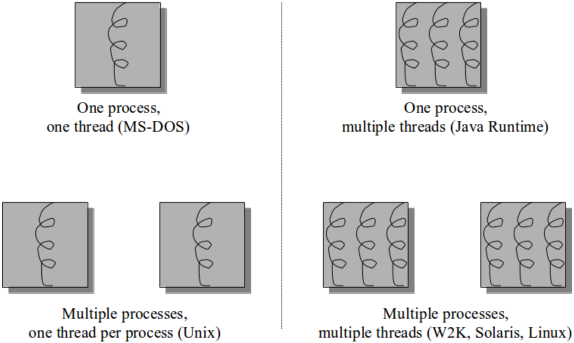

Figure 4:

- Resource Ownership - Process or Task
- Scheduling/execution - Thread or lightweight process

Single Threaded and Multithreaded Process Models

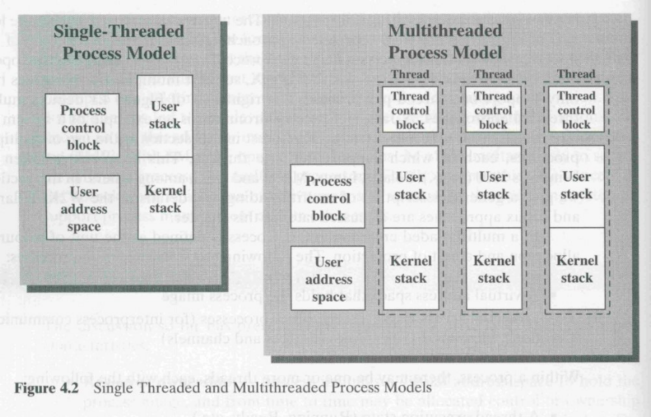

Figure 5:

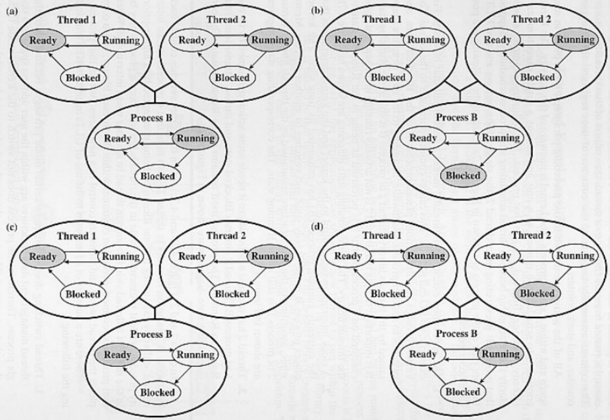

Figure 4.7 Examples of the Relationships between User-Level Thread States and Process States

Figure 6:

Key Benefits of Threads

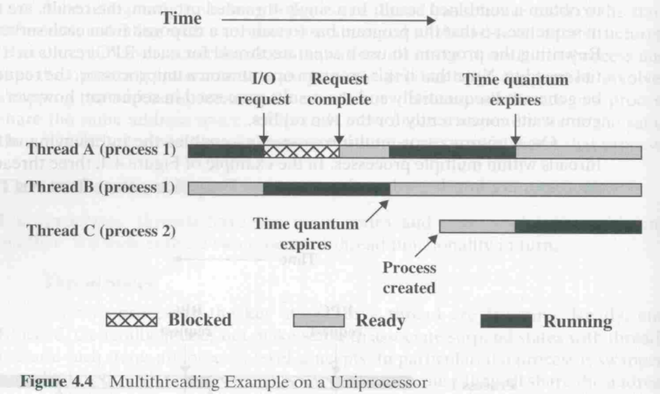

Figure 7:

User and Kernel-Level Threads Performance

- Performance
  - **Null fork**: the time to create, schedule, execute, and complete a process/thread that involves the null procedure
  - **Signal-Wait**: the time for a process/thread to signal a waiting process/thread and then wait on a condition
  - Procedure call: 7us, Kernel Trap: 17us
- Thread Operation Latencies

Operation	ULT	KLT	Process
Null fork	34	948	11,300
Signal wait	37	441	1,840

Threads Performance Observations

- While there is a significant speedup by using KLT multithreading compared to single-threaded processes, there is an additional speedup by using ULTs

Threads Performance Observations

- While there is a significant speedup by using KLT multithreading compared to single-threaded processes, there is an additional speedup by using ULTs
- However, whether or not the additional speedup is realized depends on the nature of the applications involved

Threads Performance Observations

- While there is a significant speedup by using KLT multithreading compared to single-threaded processes, there is an additional speedup by using ULTs
- However, whether or not the additional speedup is realized depends on the nature of the applications involved
- If most of the thread switches require kerenel mode access, then ULT may not perform much better than KLT