Announcements

- Project proposals due today
  - Use proper citations (at least: author(s), title, published where/when)
  - Ideally, provide URL also
- Expect email with comments from me this week
  - Revisions due Oct 12

Recap

- Tuple-based communication
  - Example: JavaSpaces/JINI
  - Replication in implementation, scalability
- Publish/subscribe Systems
  - Example: TIB/Rendezvous
  - Implementation via multicast: PGM

Outline for Today

- Threads and Processes
  - Use of threads: local and distributed
- Threads vs. Events
- Implementation of threading packages

Definition: Process/Thread

- Process
  - “program in execution”
  - resources: CPU context, memory maps, open file, privileges, ….; isolated
- Threads
  - CPU context (+ stack); not isolated
- Hybrids: Weaves, Java multitasking
- Warning: some people say “process” when they mean “thread”
Why use Threads?

*(in nondistributed case)*

- Overlap I/O and computation
- Exploit multiprocessors
  - CPU concurrency
- Software engineering reasons
  - Separation of concerns

Threads in Servers

Implementing Threads

- Issues:
  - Who maintains thread state/stack space
  - Scheduling of threads onto CPUs
  - Synchronization between threads
    - Mutexes
    - Condition variables
  - Interaction with I/O
- User level vs. Kernel level

Managing Stack Space

- Traditional stacks require continuous virtual address space
  - virtual address space fragmentation
- What size should stack have?

Preemption

- Nonpreemptive threads (“coroutines”)
  - CPU switches at well-defined points (“yield”, or synchronization points: “lock”, “wait”)
  - Cost similar to procedure call
  - Can increase latency
- Preemptive threads
  - CPU can switch at any time
    - Higher context switch cost: more state to save
    - Necessary for quasi-parallelism

Synchronization

- Common model is mutexes + condition variables
  - Separation of mutual exclusion and unilateral synchronization
User-level (aka 1:N)

- Kernel sees one thread
- Scheduling + synchronization done in user-space:
  - fast context switches
- Often nonpreemptive
  - preemption possible via signals; expensive
- I/O blocks entire process
  - can be remedied with asynchronous I/O facilities + signals

Kernel-level (aka 1:1)

- Kernel manages threads
- I/O blocks only current thread
- Context switch + synchronization requires kernel trap
- OS timer interrupts provides preemption, kernel scheduler schedules threads

Lightweight Processes (M:N)

Source: Multithreading in the Solaris Operating Environment, Sun 2002

History of LWPs

- Solaris discarded LWPs
  - Linux never introduced them
- Issues:
  - Automatic concurrency control hard
    - How many LWPs should be allocated?
  - Signal implementation difficult
  - Limited gain from faster context switches
  - _schedlock contention
- Instead 1:1 models; synchronization (fast-path) in user mode:
  - Linux: “futexes”; Solaris: adaptive mutexes

Summary

- Threads vs. Processes
- Threads vs. Events

- Thursday:
  - Resource Management in Grids
  - Eraser