

Concurrent Systems

Exercise 02 – Processes, Threads, Coroutines

Stefan Reif

2019-11-07



Agenda

Control Flows

Coroutines

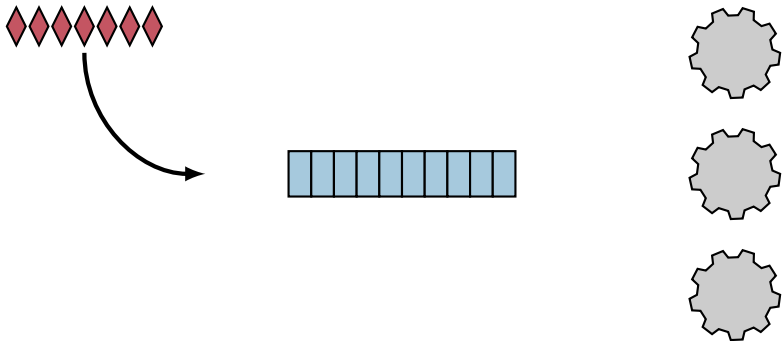
Threads

Assignment 2



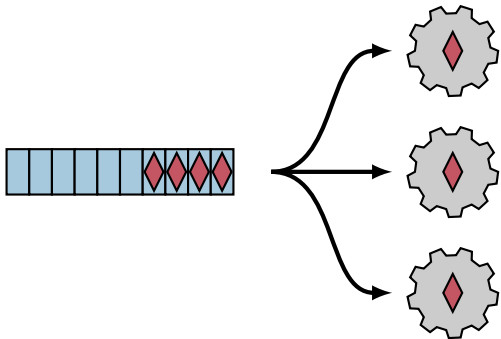
Review: Executor Service

- Executor Service (\Rightarrow Assignment 1)
 - Jobs have **run-to-completion** semantics
 - No inter-job coordination
 - No inter-job dependencies
- Example



Review: Executor Service

- Executor Service (\Rightarrow Assignment 1)
 - Jobs have **run-to-completion** semantics
 - No inter-job coordination
 - No inter-job dependencies
- Example



- **Examples**

 - ⇒ Instruction sequence, function call, interrupts, coroutines, threads, ...

- **Overlapping patterns**

 - ⇒ Sequential, stack-like, pseudo-parallel, arbitrary, ...

- **Associated resources**

 - ⇒ Stack space, address space, file descriptors, ...

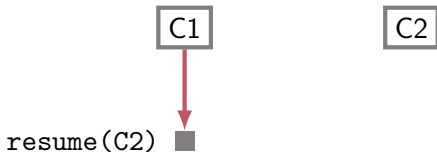
- **Synchronization**

 - Manage concurrent control flows
 - Consider application dependencies and overlapping patterns



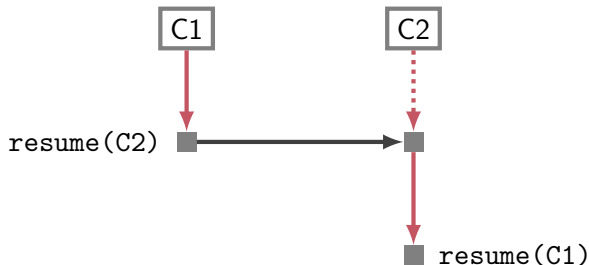
Coroutines

- Voluntarily release the processor
 - `create()`, `resume()`, `destroy()`
 - Switch to another coroutine explicitly
- Symmetric relation
- Example



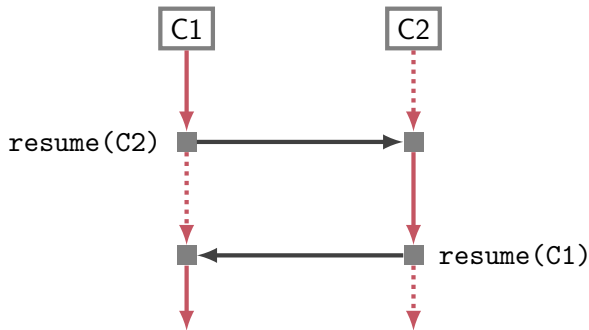
Coroutines

- Voluntarily release the processor
 - `create()`, `resume()`, `destroy()`
 - Switch to another coroutine explicitly
- Symmetric relation
- Example



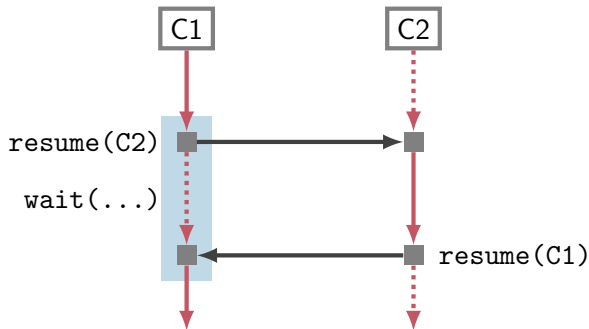
Coroutines

- Voluntarily release the processor
 - `create()`, `resume()`, `destroy()`
 - Switch to another coroutine explicitly
- Symmetric relation
- Example



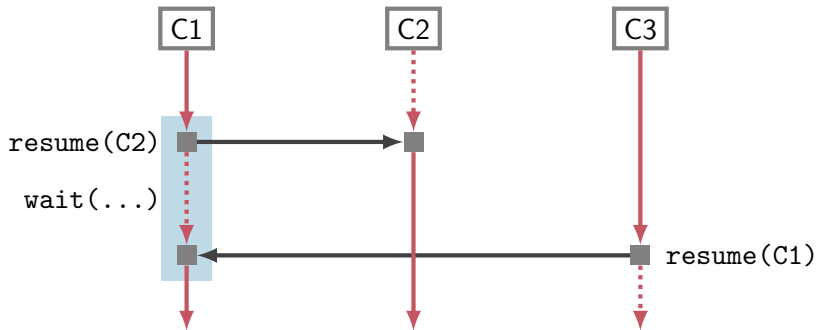
Coroutines

- Voluntarily release the processor
 - `create()`, `resume()`, `destroy()`
 - Switch to another coroutine explicitly
- Symmetric relation
- Example



Coroutines

- Voluntarily release the processor
 - `create()`, `resume()`, `destroy()`
 - Switch to another coroutine explicitly
- Symmetric relation
- Example



- Context Switches

- How can a processor switch between coroutines?
- How can a coroutine be continued?



■ Context Switches

- How can a processor switch between coroutines?
- How can a coroutine be continued?

■ Language Considerations

- Typical high-level programming languages cannot implement resume
- Some languages offer “coroutines” to programmers
 - e.g. Python `yield`
- Context switches need assembler language support



■ Context Switches

- How can a processor switch between coroutines?
- How can a coroutine be continued?

■ Language Considerations

- Typical high-level programming languages cannot implement resume
- Some languages offer “coroutines” to programmers
 - e.g. Python `yield`
- Context switches need assembler language support

■ Continuation

- Data structure for the **Context** of a coroutine
- Stopped control flow can proceed later
- Stores at least an address of the next instruction
- Typically associated with an individual stack



■ Problems with Coroutines

- Unstructured `resume()` calls
- How to choose a successor?

■ Threads extend Coroutines

- Structured synchronization primitives
 - Mutex, Condition Variable, Semaphore, Monitor, Signal, ...
- Thread states
 - READY, RUNNING, BLOCKED, TERMINATED, ...
- Scheduling
 - Manage control flows explicitly
 - Implement a strategy for idle processors

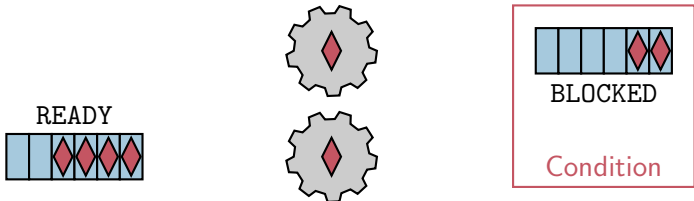


Assignment 2

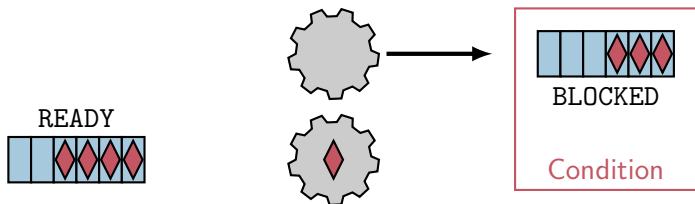
- Implement a Light-weight Threading Library (LWT)
 - Extend jobs to coroutines/threads
 - Implement synchronization primitives
- Scheduling
 - Cooperative, non-preemptive
 - One shared ready list
- Use a thread library as back-end
 - Pthreads represent “logical processor cores”
 - Pthread synchronization mechanisms are available
- Simplifications
 - No graceful termination
 - No dynamic adaption of parallelism
 - Simple scheduler



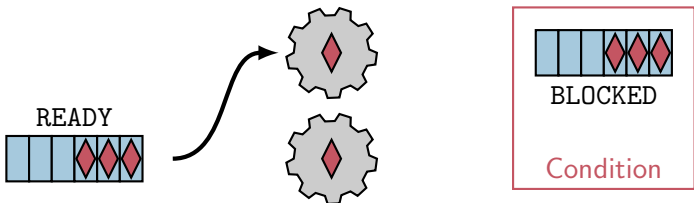
- **Blocking synchronization**
 - Set thread state to BLOCKED
 - Add thread to condition-specific waiting queue
- **Thread notification**
 - Set thread state to READY
 - Add thread to ready list



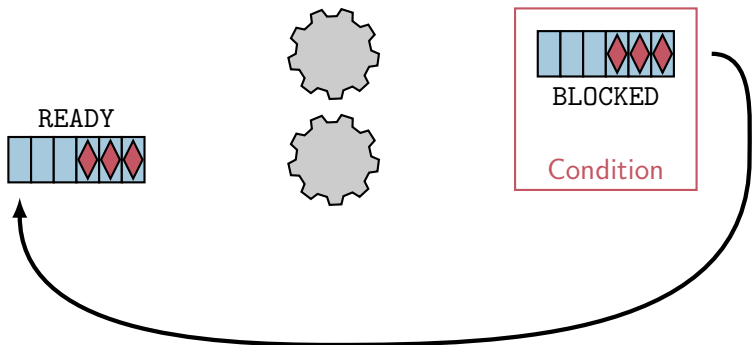
- **Blocking synchronization**
 - Set thread state to **BLOCKED**
 - Add thread to condition-specific waiting queue
- **Thread notification**
 - Set thread state to **READY**
 - Add thread to ready list



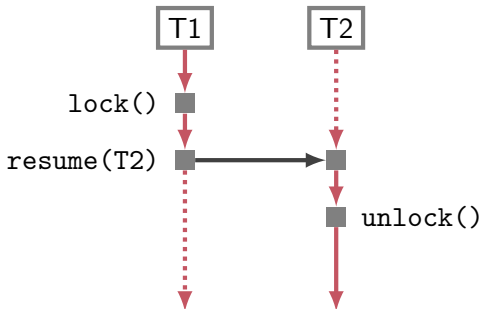
- **Blocking synchronization**
 - Set thread state to **BLOCKED**
 - Add thread to condition-specific waiting queue
- **Thread notification**
 - Set thread state to **READY**
 - Add thread to ready list



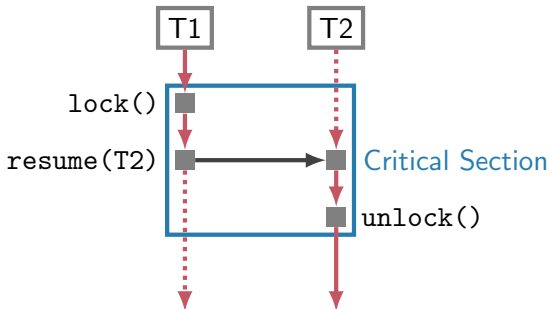
- **Blocking synchronization**
 - Set thread state to **BLOCKED**
 - Add thread to condition-specific waiting queue
- **Thread notification**
 - Set thread state to **READY**
 - Add thread to ready list



- Scheduler data structures need protection
 - We can use a pthread mutex
 - We will use non-blocking synchronization in later assignment
- What to do in idle state?
 - We can use pthread condition variables to wait passively



- Scheduler data structures need protection
 - We can use a pthread mutex
 - We will use non-blocking synchronization in later assignment
- What to do in idle state?
 - We can use pthread condition variables to wait passively



- Scheduler data structures need protection
 - We can use a pthread mutex
 - We will use non-blocking synchronization in later assignment
- What to do in idle state?
 - We can use pthread condition variables to wait passively

