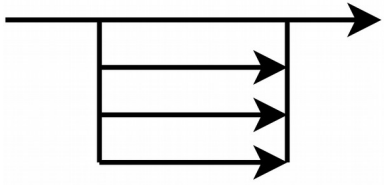
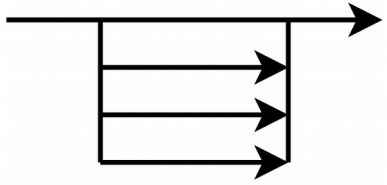
Pthreads Introduction

Parallel Computing

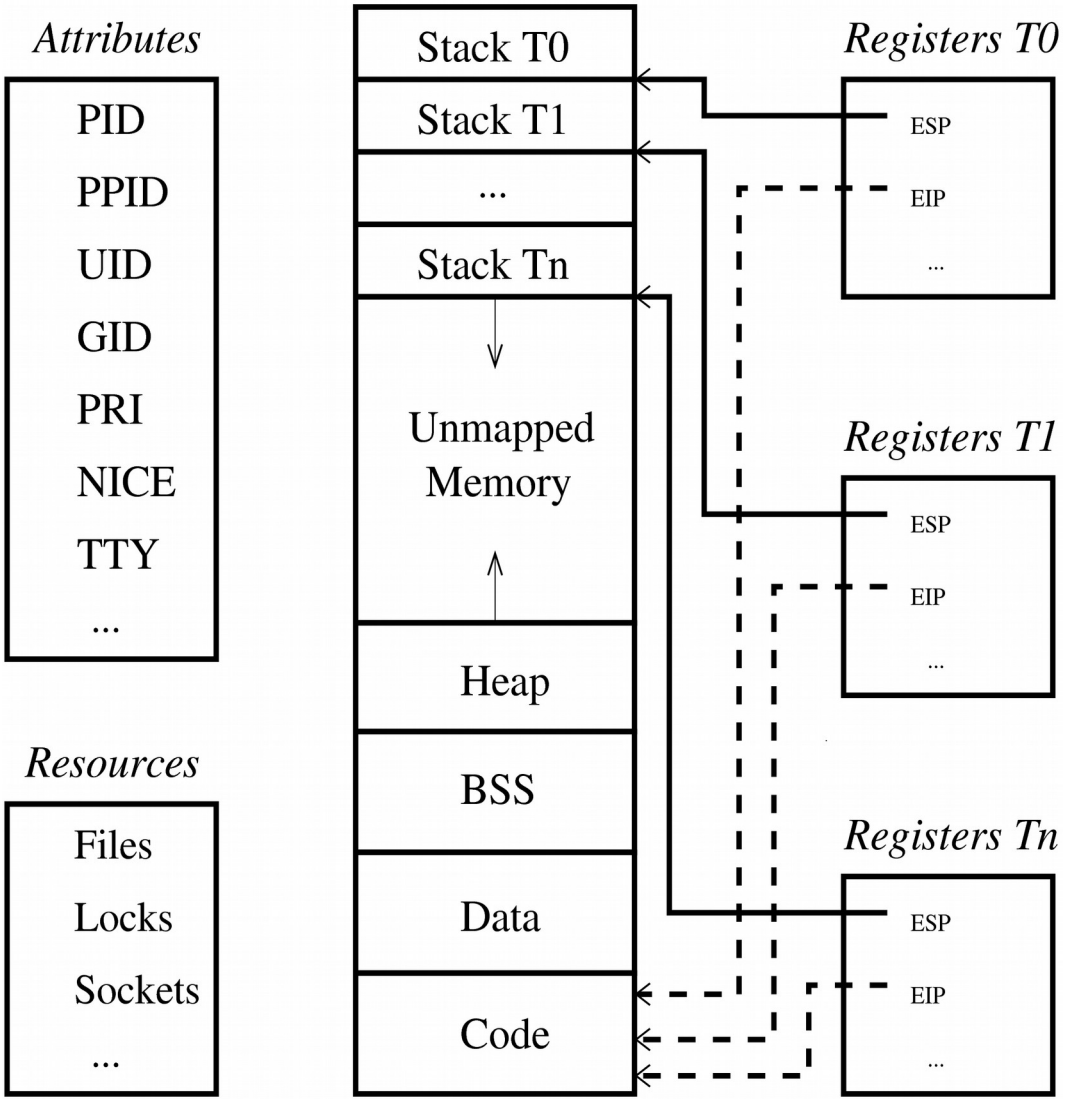
Institute for Formal Models and Verification
Johannes Kepler University, Linz, Austria





Threads vs. Processes

- Process can have multiple threads
- Thread: “lightweight” process
- Threads share address space, file descriptors, sockets,...
- Per-thread stack, program counter, registers: thread's *context*
- Switching threads more efficient than switching processes
  - “lightweight” context



Benefits of Threading

- Parallelism
  - computing independent tasks at the same time
    - speed-up (Amdahl's Law!)
  - need multiprocessor HW for “true” parallelism
  - exploiting capabilities of modern multi-core processors
- Concurrency
  - progress despite of blocking (overlapping) operations
  - no multiprocessor HW needed
  - “illusion” of parallelism
    - analogy: multiple running processes in multi-tasking operating systems
- Threaded programming model
  - shared-memory (no message passing)
  - sequential program: implicit, strong synchronization via ordering of operations
  - threaded program: explicit code constructs for synchronizing threads
  - synchronization clearly designates dependencies
  - better understanding of “real” dependencies

Costs of Threading

- Overhead (Synchronization, Computation)
  - directly: more synchronization → less parallelism, higher costs
  - indirectly: scheduling, memory architecture (cache coherence), operating system, calling C library,...
- Programming discipline
  - “thinking in parallel”
  - careful planning
  - avoidance of
    - deadlocks: circular waiting for resources
    - races: threads' speed (scheduling) determines outcome of operation
- Debugging and Testing
  - nondeterminism: timing of events depends on threads' speed (scheduling)
  - bugs difficult to reproduce
    - e.g. what thread is responsible for invalid memory access?
  - probe effect: adding debugging information can influence behaviour
  - how to test possible interleavings of threads?

When (not) to Use Threads?

- Pro threads
  - independent computations on decomposable data
    - Example: `arraysum`
  - frequently blocking operations, e.g. waiting for I/O requests
  - server applications
- Contra threads
  - highly sequential programs: every operation depends on the previous one
  - massive synchronization requirements
- Challenges in Threaded Programming
  - (applies to parallel computation in general)
  - Amdahl's Law is optimistic (ignores underlying HW, operating system,...)
  - keeping the sequential part small: less synchronization
  - increasing the parallel part: data decomposition