System and Parallel Programming

Prof. Dr. Felix Wolf

PROCESSES & THREADS

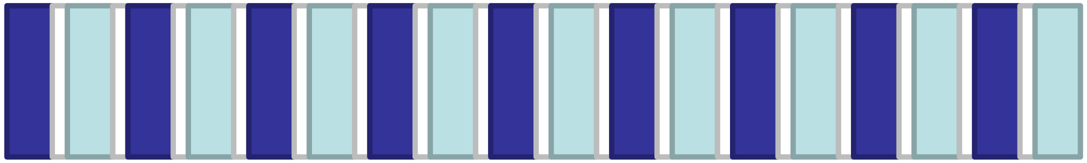
Outline

- Processes vs. threads
- On-chip multithreading

Time sharing

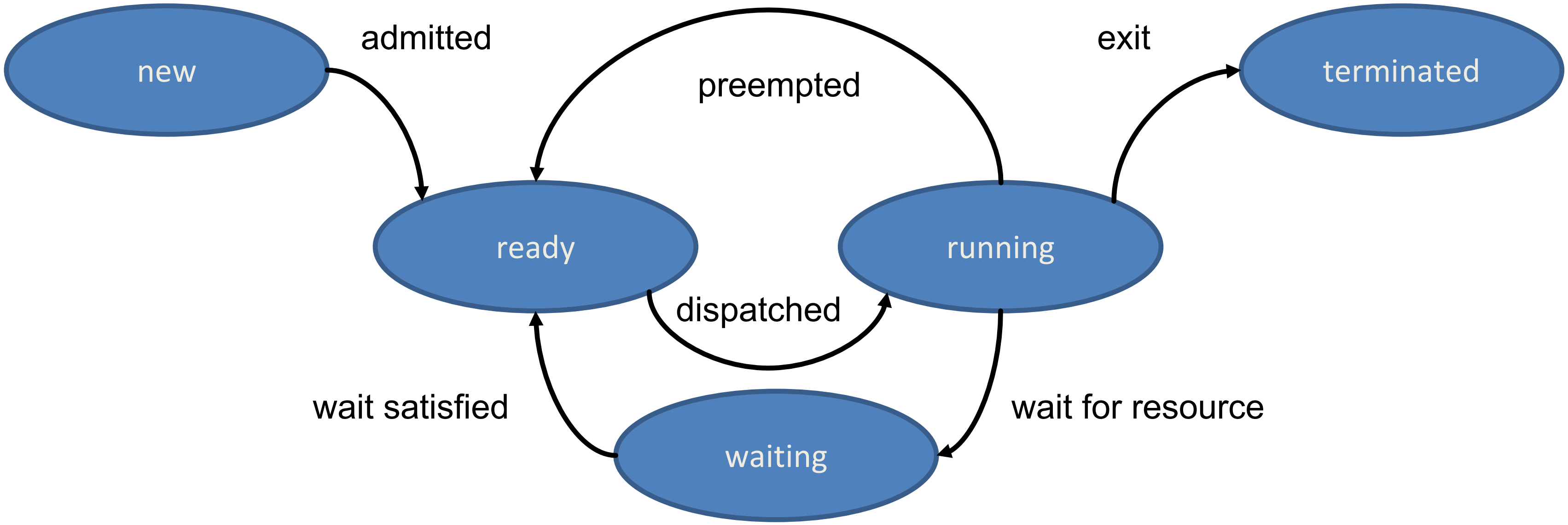
- Also called multitasking
- Logical extension of multiprogramming
- CPU executes multiple processes by switching among them
- Switches occur frequently enough so that users can interact with each program while it is running
- On a multiprocessor, processes can also run truly concurrently, taking advantage of additional processors

Single core

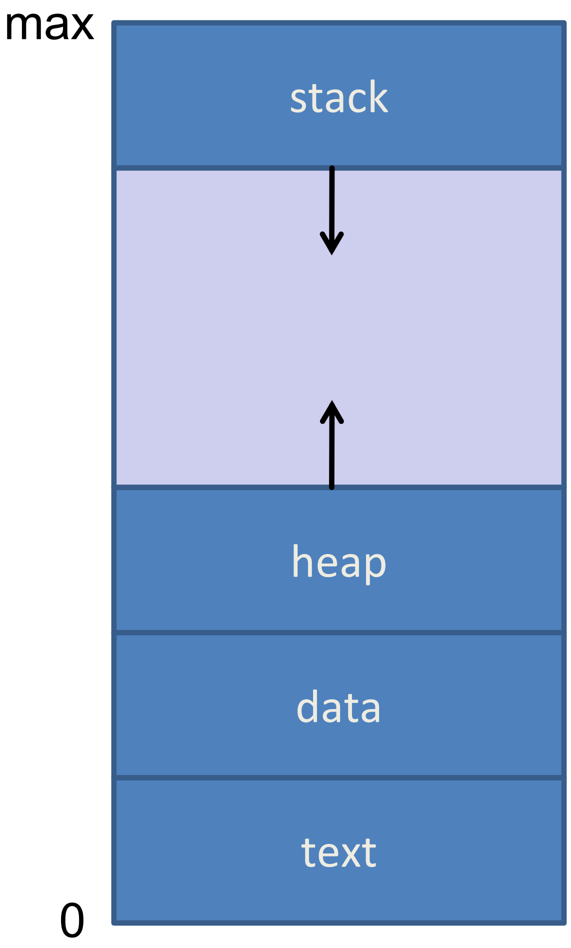


Process

- A process is a program in execution
- States of a process



Processes in memory



Temporary data: function parameters, return addresses, local variables

Dynamically allocated memory

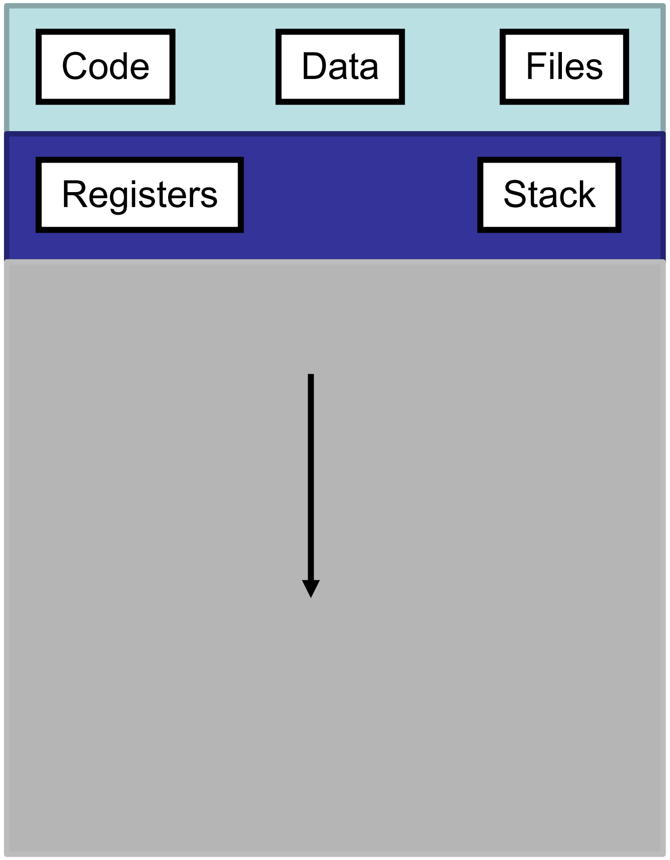
Global variables

Program code

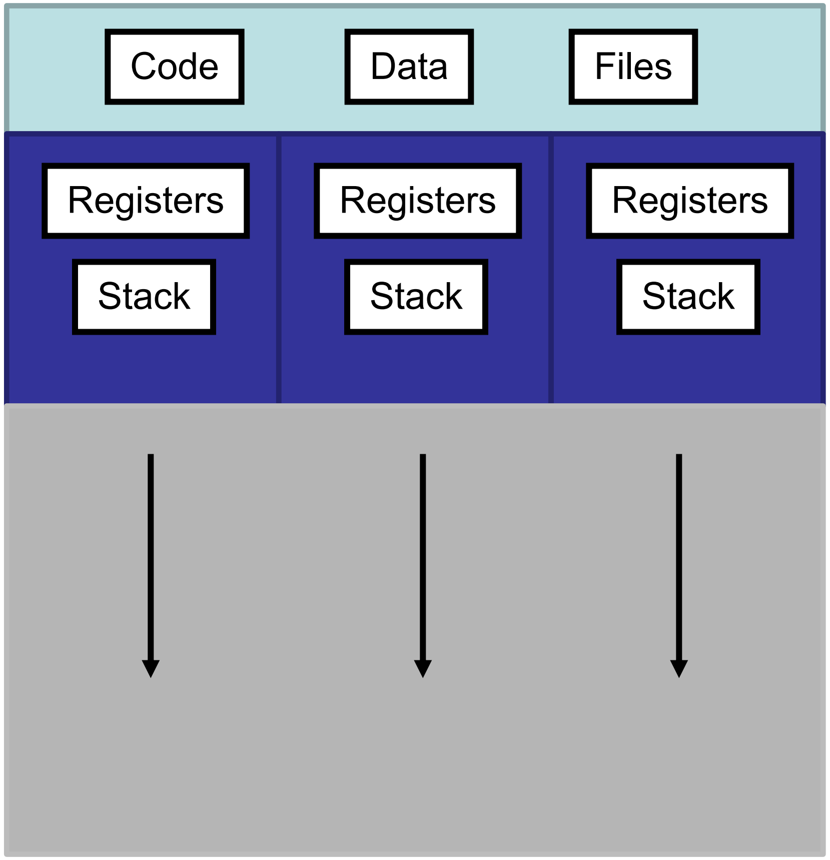
Thread

- Basic unit of CPU utilization
  - Flow of control within a process
- A thread includes
  - Thread ID
  - Program counter
  - Register set
  - Stack
- Shares resources with other threads belonging to the same process
  - Text (i.e., code) section
  - Data section (i.e., address space)
  - Other operating system resources
  - E.g., open files, signals

Single-threaded vs. multi-threaded

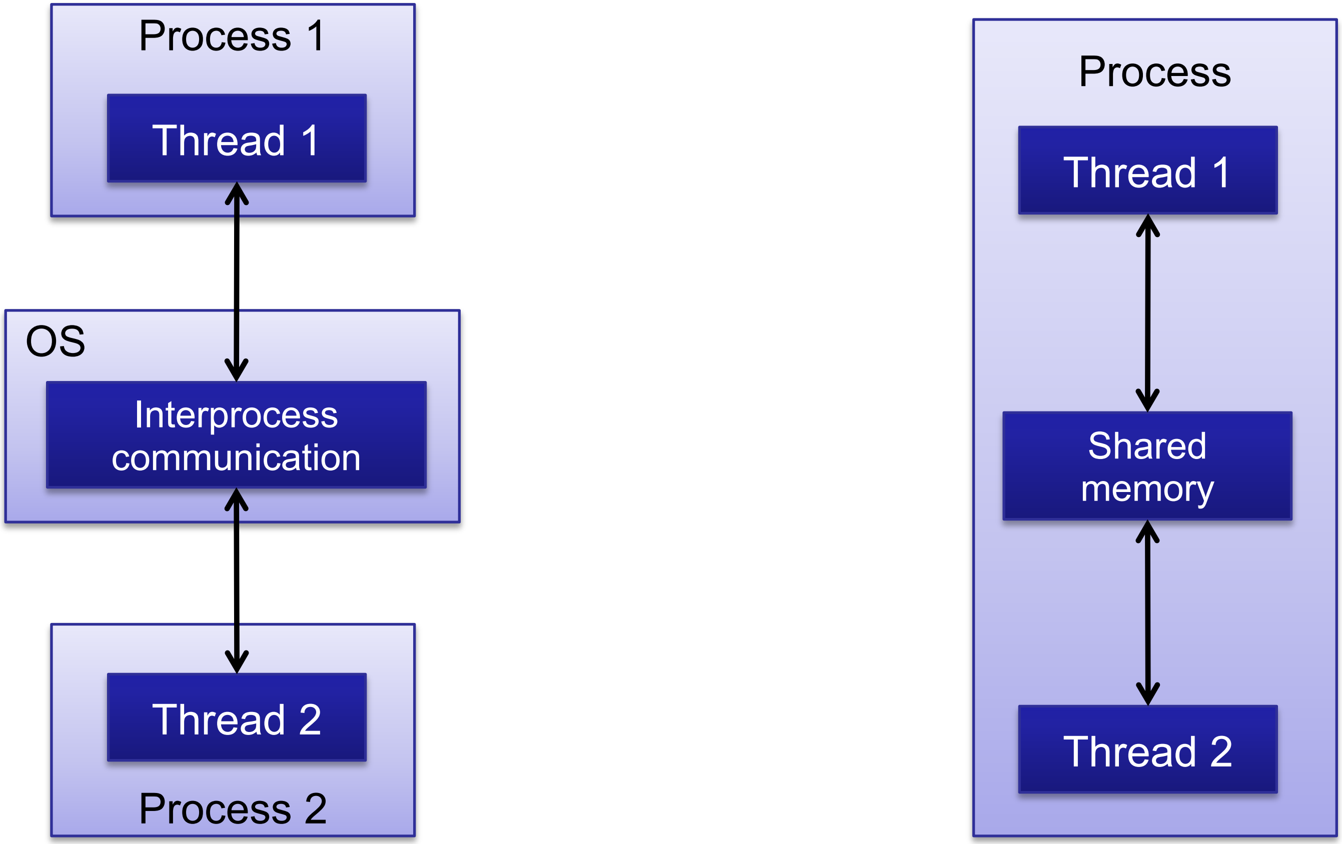


Single-threaded

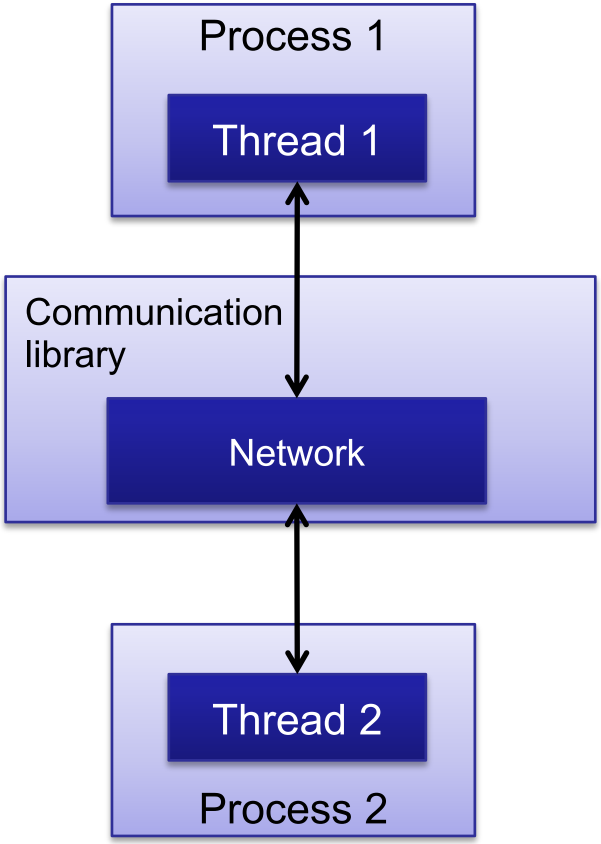


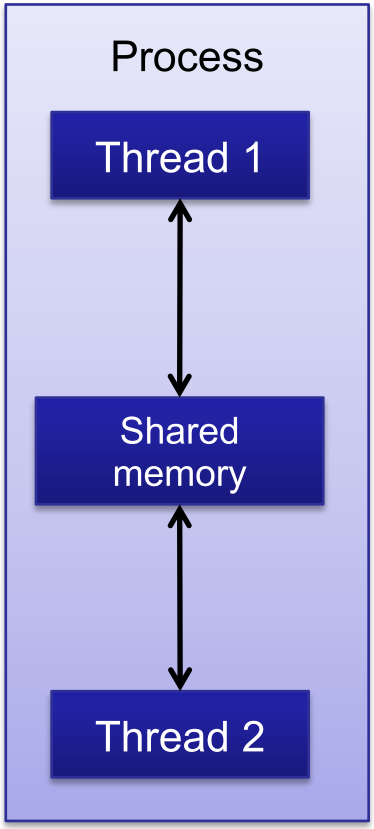
Multi-threaded

Concurrency – processes vs. threads



Concurrency – processes vs. threads (2)





Concurrency – processes vs. threads (3)

Processes

- Communication explicit
- Often requires replication of data
- Address spaces protected
- Parallelization usually implies profound redesign
- Writing debuggers is harder
- More scalable

Threads

- Convenient communication via shared variables
- More space efficient - sharing of code and data
- Context switch cheaper
- Incremental parallelization easier
- Harder to debug – race conditions

On-chip multithreading

- All modern, pipelined CPUs suffer from the following problem
  - When a memory reference misses L1 or L2 caches, it takes a long time until the requested word is loaded into the cache
- On-chip multithreading allows the CPU to manage multiple threads to mask these stalls
- If one thread is stalled, the CPU can run another thread and keep the hardware busy
- Four hardware threads per core often sufficient to hide latency

Fine-grained vs. coarse-grained multithreading

TECHNISCHE UNIVERSITÄT DARMSTADT

3 threads

A1	A2			A3	A4	A5			A6	A7	A8
B1			B2			B3	B4	B5	B6	B7	B8
C1	C2	C3	C4			C5	C6			C7	C8

Fine grained – threads run round-robin

A1	B1	C1	A2	B2	C2	A3	B3	C3	A4	B4	C4

Coarse grained – switch occurs only upon stall

A1	A2		B1		C1	C2	C3	C4		A3	A4

Fine-grained vs. coarse-grained multithreading

- Optimization of course-grained multithreading
  - Switch on any instruction that might cause a stall
  - Avoids wasted cycles
- Both options require bookkeeping
  - Fine-grained MT – attach thread ID to an instruction before it enters the pipeline
  - Coarse-grained MT – also possible to let the pipeline run dry before starting the next thread

Five ways of improving CPU performance

- Increase the clock speed
- Put two cores on a chip
- Add functional units
- Make pipeline longer
- Use on-chip multithreading

On-chip multithreading support can improve performance over-proportionally in comparison to the required extra chip area

Hyperthreading in the Intel Core i7

- Two threads (or processes) can run at once on the same core
- Looks from far like a dual processor in which both CPUs share a common cache and main memory
- However, many hardware resources are shared between threads
- Advantage – enables true concurrency within the same core
- Disadvantage – resource contention (e.g., cache) may lower throughput

Hyperthreading – resource sharing strategies

- Duplication
  - E.g., program counter, table that maps architectural onto physical registers
- Partitioning
  - E.g., slots in queue between stages of a functional pipeline
  - May lead to underutilized resources
- Full sharing
  - More flexible than partitioning but danger of starvation
- Threshold sharing
  - A thread can acquire resources dynamically up to a maximum
  - Compromise between fixed partitioning and full sharing

Summary

- A process is a program in execution
- Threads are light-weight processes that can share code and a global address space
  - Advantages – responsiveness, utilization of multiprocessors
  - Disadvantages – synchronization overhead, programming complexity
- Hardware threads can hide latency
- Hyperthreading can boost performance