

# Parallel Programming

## Processes and Threads

Prof. Paolo Bientinesi

HPAC, RWTH Aachen  
pauldj@aices.rwth-aachen.de

WS18/19



**High Performance and  
Automatic Computing**

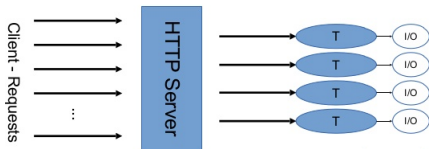
## References

- Modern Operating Systems, 4th Edition.  
*Andrew S. Tanenbaum, Herbert Bos.*  
Chapters 1.5, 2.1, and 2.2.

Only if you want to know more. For this course, these slides suffice.

# Parallel programming

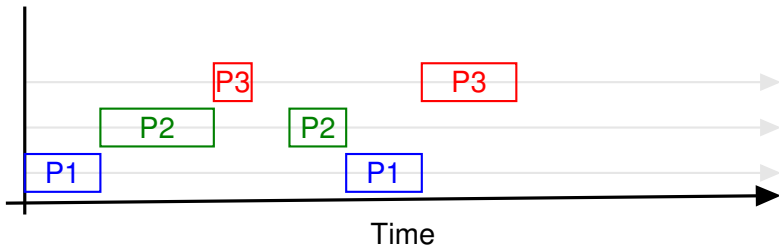
- Ability of expressing the fact that tasks/actions/instructions/programs/... take place/are performed/are executed/run/... at the same time/simultaneously/in parallel/concurrently.
- Tasks are *in progress* at the same time
- Either on a single processor or on more than one.
- Typical examples: operating system, web server, multiple programs running in your desktop, ...



# Concurrency

## Time-sharing or Multitasking systems

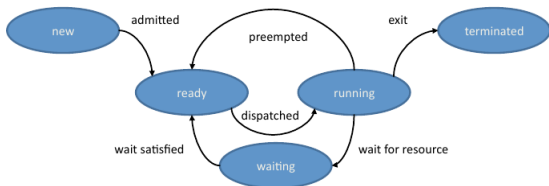
- CPU executes multiple processes by switching among them
- Switching occurs often enough for users to interact seamlessly with each program while running



- In multi-core/multi-computers, processes may actually run in parallel.

# Process

- A process is an instance of a program in execution
- States of a process
  - New: the process is being created
  - Ready: waiting to be assigned to a processor
  - Running: instructions are being executed
  - Waiting: waiting for some event to occur (e.g., I/O completion)
  - Terminated: has finished execution



# Thread

---

- Basic unit of CPU utilization
  - 1 process  $\rightarrow \geq 1$  threads (usually: 1-2 threads)
- A thread includes
  - Thread ID
  - Program counter
  - Register set
  - Stack
- Threads within the same process share resource
  - Text section
  - Data section
  - Other OS resources (open files, ...)

# Processes vs Threads

---

- Processes

- Independent
- Have separate address spaces
- Heavier: carry more information
- Creation, context switching, . . . : expensive
- Communicate via system-provided inter-process communication mechanisms

- Threads

- Exist within a process
- Share address spaces
- Lighter (faster creation, context switching, . . .)
- Communicate via shared variables

# Single Program Multiple Data

---

- Most common programming model
- The same program is executed on multiple processors
- **Different control flow based on the process/thread ID**

```
if (process_id == 0)
    do_something()
else
    do_something_else()
```



- OpenMP
  - Not a programming language.  
Mainly annotations to the (existing, sequential) code.
  - “High”-level interface
  - Emphasis on high-performance computing
- IEEE POSIX Threads (PThreads)
  - Standard UNIX threading API. Also used in Windows.
  - Over 60 functions: `pthread_create`, `pthread_join`, `pthread_exit`, etc.

- Steered by the OpenMP ARB (industry, academia)
- Assumes shared memory
- Allows definition of shared/private variables
- Language extensions
  - Compiler directives
  - Library of routines
  - Runtime for the creation and management of threads

```
#pragma omp parallel for
for( i = 0; i < n; i++ )
    z[i] = a * x[i] + y[i]
```

- Currently available for C/C++ and Fortran
- [www.openmp.org](http://www.openmp.org)

# Message Passing

---

- Multiple processes – not necessarily different nodes
- Each with its own private address spaces
- Access to (remote) data of other processes via explicit communication: send/receive

```
if (process_id == SENDER)
    send_to(RECEIVER, data);

if (process_id == RECEIVER)
    recv_from(SENDER, data);
```

- Well-suited for distributed memory
- MPI (*Message Passing Interface*) is the *de-facto* standard

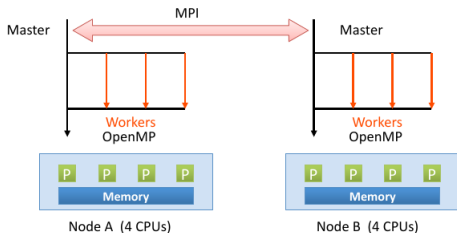
# Message Passing Interface (MPI)

---

- Specified and managed by the MPI Forum
  - Library offers a collection of communication primitives
  - Language bindings for C and Fortran
  - [www.mpi-forum.org](http://www.mpi-forum.org)
- Relatively low-level programming model
  - Data distribution and communication must be done manually
  - Primitives are easy to use, but designing parallel programs is hard
- Communication modes
  - Point-to-point (messages between two processors)
  - Collective (messages among groups of processors)
    - $1 \rightarrow n$  (e.g., broadcast)
    - $n \rightarrow 1$  (e.g., reduce)
    - $n \rightarrow n$  (e.g., allreduce )

# Hybrid programming

- Multiple processes, each spawning a number of threads
  - Inter-process communication via message passing (MPI)
  - Intra-process (thread) communication via shared memory
- Especially well-suited for hybrid architectures. For instance:
  - one process per shared-memory node, and
  - one thread per core



# Summary

---

- Process: instance of a program in execution
  - Container: code, data, process control block
  - Independent, heavier than threads
  - Communicate via inter-process mechanisms
- Threads: Unit of execution within a process
  - Share code and global address space
  - Private stack, lightweight
  - Communicate via shared variables
- Single Process Multiple Data (SPMD) paradigm
  - Message Passing via MPI
  - Shared-memory multithreading via OpenMP