Parallel Programming

Processes and Threads

Diego Fabregat-Traver and Prof. Paolo Bientinesi

HPAC, RWTH Aachen
`fabregat@aices.rwth-aachen.de`

WS16/17

References

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

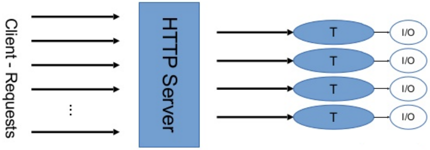
  Only if you want to know more. This slides are more than enough for this course!

Outline

1. Concurrency
2. Processes
3. Threads
4. MPI
5. OpenMP

Concurrency

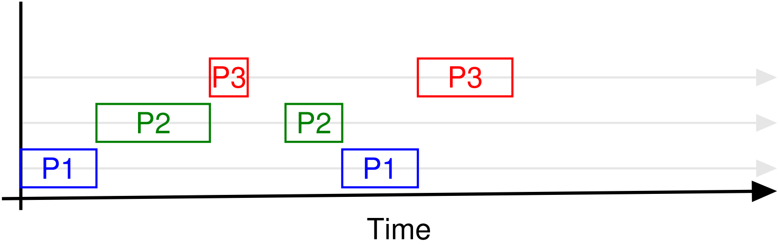
- A property of computing systems in which several tasks are executing simultaneously
- Tasks are *in progress* at the same time
- Maybe running on one single processor, maybe on more than one
- Typical examples: web server, multiple programs running in your desktop, ...



Concurrency

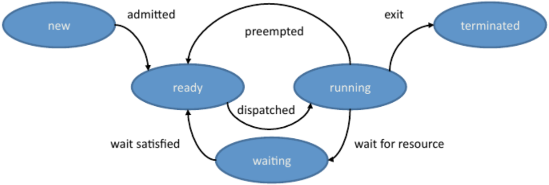
Time-sharing or Multitasking systems

- CPU executes multiple processes by switching among them
- Switching occurs often enough for users to interact with each program while running
- In multi-core / multi-computer, processes may indeed be running in parallel.



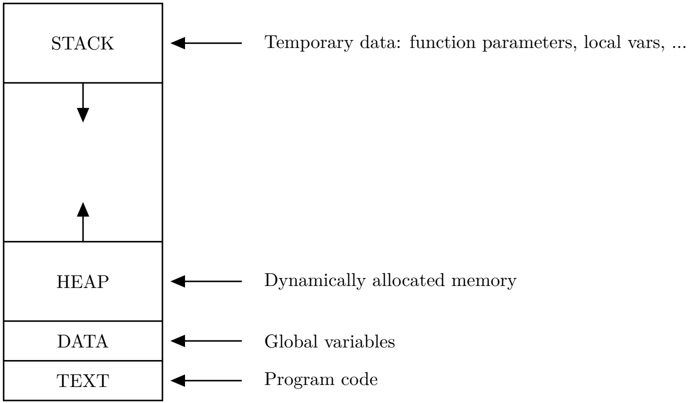
Process

- 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



Process

- Associated address space
  - Program itself (text section)
  - Program's data (data section)
  - Stack, heap


STACK
Temporary data: function parameters, local vars, ...
HEAP
Dynamically allocated memory
DATA
Global variables
TEXT
Program code

Process

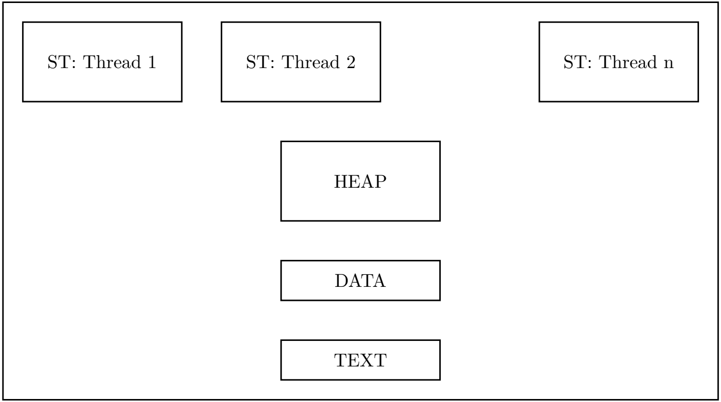
- Process control block
  - Process ID
  - Process status
  - CPU registers (PC, ...)
  - Open files, memory management, ...
- Stores context to ensure a process can continue its execution properly after switching by restoring this context.

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 within the same process
  - Text section
  - Data section
  - Other OS resources (open files, ...)

Thread

Single-threaded vs multi-threaded



Processes vs Threads

- Processes:
  - Independent
  - Have separate address spaces
  - Heavier (carry more information)
  - Creation, context switching, ... is more expensive
  - Communicate via system-provided inter-process communication mechanisms
- Threads:
  - Exist within a process
  - Share address spaces
  - Lighter (faster creation, context switchin, ...)
  - Communicate via shared variables

Programming models

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()
```

Message Passing

- Multiple processes (not necessarily running on different nodes)
- Each with its own private address spaces
- Access to (remote) data of other processes via sending/receiving messages (explicit communication)

```
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/C++ and Fortran
  - `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)
    - $1 \to n$ (e.g., broadcast)
    - $n \to 1$ (e.g., reduce)
    - $n \to n$ (e.g., allreduce)

Shared memory

- OpenMP
  - Higher level interface based on:
    - compiler directives
    - library routines
    - runtime
  - 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.

OpenMP

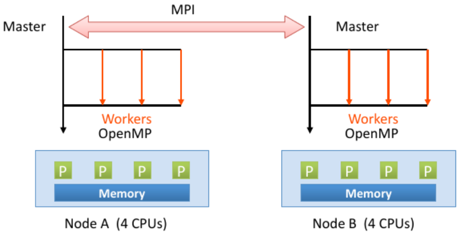
- Specified and managed by the OpenMP ARB
- Assumes shared memory
- Allows definition of shared/private variables
- Language extensions based on:
  - Compiler directives
  - Library of routines
  - Runtime for the creation and management of threads

```
#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`

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