

# Building Blocks

---

Operating Systems, Processes, Threads



# Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en\\_US](http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US)

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

# Outline

- What does an Operating System (OS) do?
  - OS types in HPC
  - The Command Line
- Processes
- Threads
  - Threads on accelerators
- OS performance optimisation
  - Why is the OS bad for performance?
  - Approaches to improving OS performance

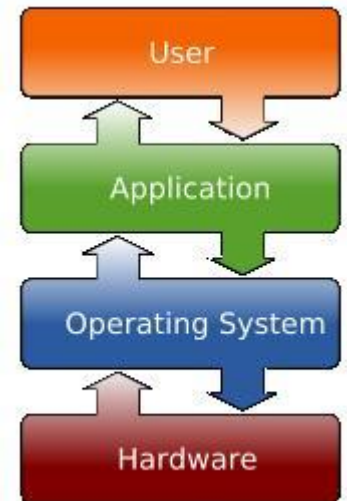
# Operating Systems

What do they do? Which ones are used for HPC?



# Operating System (OS)

- The OS is responsible for orchestrating access to the hardware by applications.
  - Which applications are running at any one time?
  - How is the memory allocated and de-allocated?
  - How is the file-system accessed?
  - Who has authority to access which resources?
- Running applications are controlled through the concepts of *processes* and *threads*.
  - an applications / program is a single process...
  - ...which may have multiple threads



# OS's for HPC

- HPC systems have always used Unix
  - vendors (DEC, SUN, Cray, IBM, SGI, ...) all wrote their own version
- Now dominated by Linux (of various flavours)
  - Most HPC vendors modify a commercial Linux distro (RedHat or SUSE) and tailor to their own system.
  - Many commodity clusters run a free Linux distro (CentOS is particularly popular).
- Only IBM Power systems still use vendor Unix (AIX)
  - 3 HPC systems in the June 2016 Top500 do not use Linux
- Windows really not used for HPC
  - No systems in the June 2016 Top500 list use Windows

# The Command Line

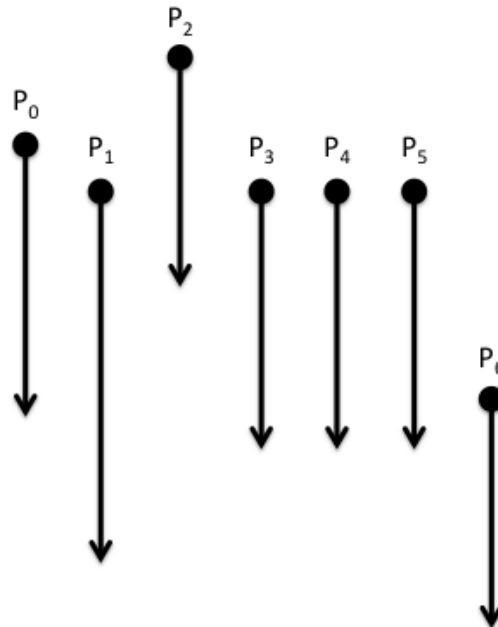
- HPC sector is dominated by Linux
- Interaction almost always through Linux command line.
  - e.g. which two files or folders are taking up the most space?  
`user@hpcsystem> du -sm * | sort -n | tail -2`
  - often a reasonably large barrier to new people adopting HPC.
- For any serious use of HPC you will have to learn to use the command line.
  - often also useful for using command line on your own laptop/PC
- Should also learn basic operation of in-terminal text editor
  - vi/vim is generally available
  - emacs is another popular choice

# Processes



# Processes

- Each application is a separate *process* in the OS
  - a process has its own memory space which is not accessible by other running process.
  - processes are ring-fenced from each other: if web browser crashes, it can't scribble over document stored in the memory your word processor
- Each process is scheduled to run by the OS



# OS and multicore

- “*Multicore parallelism – manually specified by the user*”
  - what’s the use of a multicore laptop if I run non-parallel code?
- OS’s have *always* scheduled multiple processes
  - regularly check which process is running
  - give another process a chance to run for a while
  - rapid process switching gives *illusion* applications run concurrently even on a single core
- With a multicore processor
  - multiple processes can *really* run at the same time

# Process Scheduling

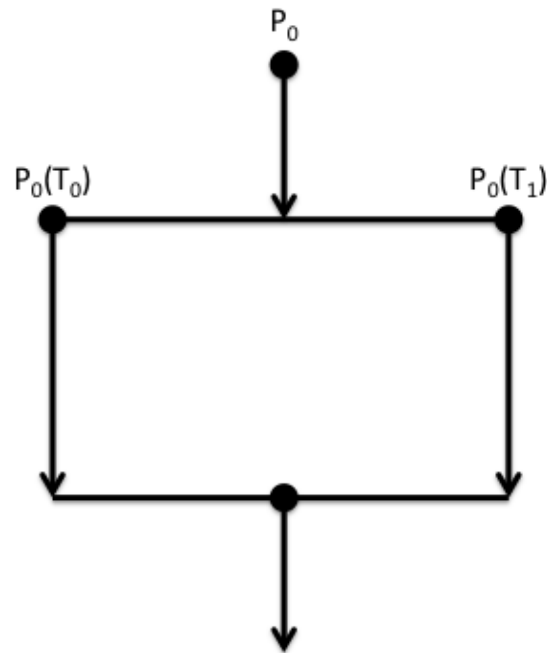
- The OS has responsibility for interrupting a process and granting the core to another process
  - Which process is selected is determined by the *scheduling policy*
  - Interrupt happens at regular intervals (every 0.01 seconds is typical)
  - Process selected should have processing work to do
- On a quad core processor, OS schedules 4 processes at once
- Some hardware supports multiple processes per core
  - Known as *Symmetric Multi-threading* (SMT)
  - Usually appears to the OS as an additional core to use for scheduling
- Process scheduling can be a hindrance to performance
  - in HPC, typically want a single user process per core

# Threads

Sharing memory

# Threads

- For many applications each process has a single *thread*...
  - ... but a single process can contain multiple threads
  - each thread is like a child process contained *within* parent process



# Threads (cont.)

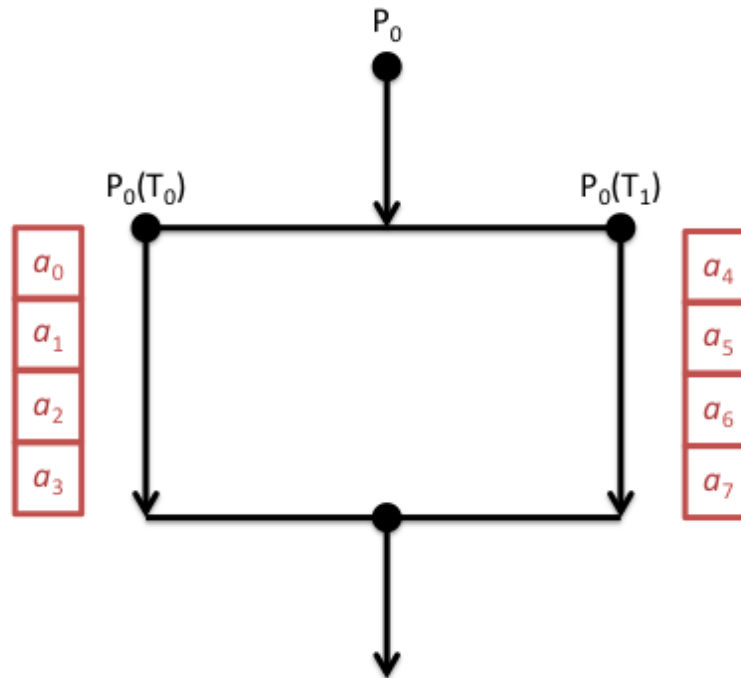
- All threads in a process have access to the same memory
  - the memory of the parent process
- Threads are a useful programming model pre-dating multicore
  - e.g. a computer game (a process) creates asynchronous threads
    - one thread controls the spaceship
    - another controls the missile
    - another deals with keyboard input
    - ...
  - but all threads update the same game memory, e.g. the screen
- OS scheduling policy is aware of threads
  - ensures all of the game operations progress
  - switching between threads usually quicker than between processes

# Threads and multicore

- With multiple cores
  - multiple threads can operate at the same time on the same data to speed up applications
- Cannot scale beyond the number of cores managed by the operating system
  - to share memory, threads must belong to the same parent process
- In HPC terms cannot scale beyond a single *node*
  - using multiple nodes requires multiple processes
  - this requires inter-process communication – see later

# Shared-memory concepts

- Process has an array of size eight
  - each thread operates on half the data; potential for 2x speedup





# Threads and Accelerators

- The Accelerator programming model generally requires a huge number of threads to provide efficient usage
  - Oversubscription of the accelerator by threads is encouraged
  - Hardware supports fast switching of execution of threads
    - switch off a thread when it is waiting for data from memory
    - switch on a thread that is ready to do computation
    - try and hide memory latency
  - As GPGPUs can have 1000's of computing elements, oversubscription can be difficult!
- Threading is becoming more and more important on modern HPC machines

# OS Optimisation

How do vendors get performance?

# Different Compute node OS

- Interactive (front-end) nodes usually run a full OS
  - You interact with these directly and-so the OS provides a full suite of tools and functionality that you would expect
- Compute nodes often run an optimised OS to improve performance
  - A greatly stripped down system with the bare minimum functionality required to run codes
  - As you don't interact directly with the compute nodes then they just need to support the execution of codes
  - Example is Cray's Compute Node Linux which is based on SUSE Linux (Enterprise)

# OS Optimisations

- Remove features that are not needed (e.g. USB support)
  - The best optimization is not to have it in the first place!
- Restrict scheduling flexibility and increase interrupt period
  - To avoid interrupting user programs
- Bind processes and threads to specific cores
- Remove support for virtual memory (paging)
  - Which provides the illusion of more RAM than is physically available by writing out to disk. However this is very slow and hence avoided with HPC.
  - Therefore the amount of RAM present is a strict memory limit that you can't go beyond

# Summary

# Summary

- OS orchestrates access to hardware resources by software
  - HPC sector dominated by Linux
  - Access via command line rather than GUI
- Each running application is a separate process
  - Has its own memory space and can be scheduled by OS
  - Usually placed on a particular core and not moved
- Running applications can have one or more threads
  - Threads share memory and OS switching is faster than for processes
- OS Optimisation
  - Remove unnecessary features
  - Increase user-level control of placement