Concurrency

processes are executed sequentially but are swapped quickly. in reality technically only one process is executed ata time

Parallelism
multiple cores allows for actual multitasking. As the number of threads grows, so does architectural support for threading. cpus have cores as well as hyper threading

Data parallelsim

distributes subsets of the same data accross multiple cores that perform the same operations on each

task parallelism

distributing threads across cores, each thread does a unqiue operation

Hyperthreading

intels term fro simultaneous multithreading (SMT)

- a process where a cpu splits each of its physical cores into vitual cores
    - a hardware inovation that allows more than one thread to run on each core

Architectural Innovations

- registers have been duplicated (each virtual processor gets its on set of registers)
- cache has been divided into 2 haves
- the execution units have been duplicated

Limitations

- components like the alu and floating point unit (fpu) are typically shared among logical processrs
- if data in registers is swapped rapidly, this slows us down

Amdahls law

identifies performance gains from adding additional cores to an application that has both serial and parallel components

“Pasted image 20240202102437.png” could not be found.

as n -> infinity, speed approaches 1/S... the serial portion of an application has a disproportionate effect on performance gained by adding more cores

User threads

managed by user-level thread library

- high level api
  - has a user-level scheduler
- each process has its own user levl thread scheduler -> process specific scheduler
  - the kernel is generally unaware of the individual user levle threads
  - lack of kernel support -> limited system resource utilization
- lightwheight due to no kernal intervention
- flexibile due to user level thread management

kernel threads

supported by kernel

- better parallelism from the support of the kernel
- generally more robust
- heavier due to deep kernel involvement
- posix pthreads, windows threads, java threads

Mapping between user and kernel threads

“Pasted image 20240202103320.png” could not be found.

reason for mapping is for use of both systems. It allows us to determine which proces will serve what. allowing us to properlly schedule

- without mapping, the entire processs including all its userlevel threads would be blocked, because the kernel is responsible for scheduling processes not user level threads
- a user level thread making a block call would not block the entire process
- mapping ult to klt allows multiple user-level threads to run simultaneously

Mapping procedure

- initialize some code in the user level threads libraary with some starting program
- upon creation of user level therad, the library interacts with the kernel to request the creation of a corresponding kernel levle thread
- the user level threads library determines how user level threads map to kernel level threads

Mapping models

“Pasted image 20240202104121.png” could not be found.

“Pasted image 20240202104239.png” could not be found.

“Pasted image 20240202104515.png” could not be found.

Implicit threading

growing in popularity, as numbers of threads increase, program correctness is more difficult with explicit thtreads

- makes it hard to manage concurrency

Thread Pools

create a number of threads in a thread pool where they await work

- adds a slight speed benefit as threads are created before they need to be executed
- overhead for creating threads is the tradefoff

OpenMp (Open multi processing)

“Pasted image 20240202110007.png” could not be found.

Grand central dispatch

“Pasted image 20240202110025.png” could not be found.