chapter 4

- thread → basic unit of CPU utilization context
  - ↳ Private: threa ID, stack, register set, Program counter
  - ↳ shared with other threads of the same Process: code, data, resources (memory and I/O)
- single threaded Process ≡ heavy weight
- use processes instead of threads ⇒ overhead and time consuming
- in RPC the server creates a thread for each request

Adv → responsiveness: allows a Program to ~~[illegible]~~ continue running even if Part of is blocked or running a lengthy operation ⇒ UI

↳ resource sharing: have several threads within the same address space sharing code, data and resources

↳ economy: context switch between threads is faster than Processes

↳ scalibility: threads may be running in Parallel on processing cores

* Multicore Programming

- systems with multiple cores across or within CPU chips ⇒ Multicore

Multicore Programming → improves $\boxed{\text{concurrency}}$ ≠ Parallelism

↳ more efficient use of multiple cores

- it's Possible to have concurrency without Parallelism.
- Parallelism → data Parallelism: distributing data across cores executing the same operation
  - ↳ task // : distributing different operations across cores
- Amdahl's law

S: the Portion of the application that must be Performed serially

N: number of Processing cores

- $N \to \infty \Rightarrow$ speed up $\leq \frac{1}{S}$

$$\Rightarrow \text{speed up} \leq \frac{1}{S + \frac{(1-S)}{N}}$$

Scanned by CamScanner

Date No

* Programming challenges

- application Programmers must design multithreaded Programs
- O.S. designers must write algorithms to use multiple cores
  1- Identifying tasks: find areas of app that can be divided into seperate concurrent tasks
  2- Balance: find valuable tasks and assigning cores to them
  3- Data splitting: between tasks running on different cores
  4- Data dependency: synchronize task so that one can use the result of others
  5- Testing & debugging: controling execution Paths

* Multithreading Models

threads < kernel threads / user "

- Many-to-one model → thread management done by thread library in user-space => efficient
  - ↳ the entire process will block if a thread makes a blocking system call
  - ↳ multiple threads may not run in Parallel
  - ↳ used in few systems (Solaris & Green threads)
- one-to-one model → more concurrency than many-to-one
  - disadv ↳ overhead by creating kernel threads => restrict # of threads
  - ↳ linux and windows family
- Many-to-many model → Multiplexing
  - ↳ # of threads depends on app and machine
  - ↳ more concurrency than many-to-one (sufficient number of threads)
  - ↳ user-thread blocking solved by creating new kernel threads
  - ↳ user thread bound to kernel thread => two-level model

allowing multitasking to be done at the user-level

Scanned by CamScanner

* Thread state → spawn

↳ blocked : waiting for an event

↳ unblocked : ↳ done. thread is moved to the ready queue

↳ finish : thread's register context and stacks are deallocated

* Thread local storage (TLS)

- allows each thread to have it's own copy of data
- useful when you don't have control over thread's creation (thread Pool)
- different from local variables and similar to static data
- visible across function invocations
- unique to each thread

* Thread cancelation → terminating a thread before it has completed

target thread

1. asynchronous : one thread immediately terminates target thread
2. deferred : target thread Periodically checks wether it should be terminated. → ~~the~~ flag checking

→ may not free a necessary system_wide resource.

* Linux threads → tasks instead of threads

↳ thread creation using clone() system call

↳ flags control behaviour over address space of Parent

Scanned by CamScanner