- Chapter 4

- process creation is heavy-weight while thread creation is light-weight

- Threads can simplify code and increase effciency.

- Examples for tasks can be implemented by separate threads
  - ↳ Update display, Fetch data, spell checking, Answer a network request

- Client (1)request → Server (2)→ thread (3)↻ (2) create new thread to service the request

  (Asynchronous) (3) resume listening for additional client requests

- Synchronous threading: the parent waits for the forked children to finish
  - ↳ lots of data sharing between threads

    ex: merge sort

- Asynchronous threading: the parent does not wait, all run concurrently
  - ↳ lots of duplication of data, but no communication.

- Threading benefits:
  - ↳ 1) Responsiveness: allow continued execution of part of process is blocked
    2) Resource Sharing: easier than shared memory or message passing
    3) Economy: 1) cheaper than process creation
       2) thread switching lower overhead than context switch
    4) Scalability: run threads on different cores, in parallel

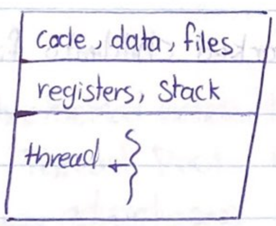
- Parallelism: a system can perform more than one task simultaneously
  - ↳ not possible on single core

- Concurrency: Support more than one task making progress
  - ↳ Single core → Scheduler providing concurrency

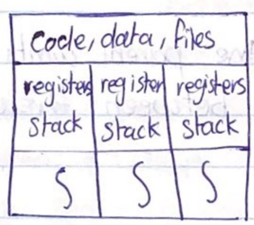
- types of Parallelism:
  - 1) data Parallelism: distributes subsets of the same data across multiple cores with same operation on each
  - 2) Task parallelism: distributing threads across cores, each thread perform unique operation.

threads
# of threads grows, so does achitectural support for threading / CPU has cores and Hardware th

- Multicore programming challenges:
  1) Task identification (which part to be Parallel/concurr
  2) Balance (equally valued tasks)
  3) Data splitting
  4) Data dependency
  5) Testing and debugging



Single-thread process



multi thread process

- Amdahl's law: $\text{Speed up} \leq \frac{1}{S + \frac{(1-S)}{N}}$ S: Serial portion, N: processing cores
- as $N \to \infty$, $\text{Speed up} \to \frac{1}{S}$
  - ↳ Serial portion has disproportionate effect on the performance
- user threads: managment done by user-level threads library
  - ↳ like Pthreads in POSIX, Windows threads, Java threads
- kernal threads: Supported by the kernal
  - ↳ treated as virtual processor

- Mapping user threads to kernal threads
  - ↳ Many-to-One , One-to-One , Many-to-Many

- Many-to-One: many user-level threads mapped to single kernal thread
  - ↳ 1) One thread blocking causes all to block
  - 2) Multiple threads may not run in parallel because only one may be in the kernal at a time (in muticore system)
  - Like Solaris Green , GNU Portable threads.

- One to One: Eeach user-level thread maps to kernal thread
  - ↳ 1) Creating a user thread creates a kernal thread
  - 2) More concurrency than many to one
  - 3) # of threads / process sometimes restricted due to over head
  - Like Windows , Linux , Solaris 9 and later.

- Many to Many: many user threads mapped to many kernal threads
  - ↳ Allows the OS to create sufficient # of kernal threads
  - Like Windows with the ThreadFiber package , Solaris prior to version 9

- Two-level model: it allows a user thread to be bound to kernal thread
  - Like IRIX, HP-UX, Tru64 UNIX, Solaris 8 and ~~elem~~ earlier

- Thread library: provids programmer with API for creating/managing threads
  - ↳ Implementedting way: 1) Library entirely in user space
  - 2) Kernal-level Library supported by the OS

- Pthread: Provided either as user or kernal level
  - ↳ POSIX Standard ~~API~~ (IEEE 1003.1c) defining an API for thread creation and synchronization
  - → a specification for thread behavior not implementation

- Java thread: managed by the JVM
  - ↳ may be created by: 1) Extending thread class
    2) Implement the runnable interface

- Program correctness more difficult with explicit threads

  why implicit threading? أهم سؤال

- Creation and management of threads done by compilers and run-time libraries rather than programmers, This is named as implicit threading

- three methods on implicit threading: 1) Thread Pool
  2) OpenMP
  3) Grand Central Dispatch

Other method include Microsoft Threading Building Block (TBB), Java.util.concurrent package

- Thread Pool: Create number of threads in a pool where they await work
  - ↳ advantages: 1) faster to service a request with an existing thread than create a new one
    2) Numbers of threads in the application are bound to the size of the Pool.
    3) Separating task to be performed from mechanics of creating task allows different strategies for running task (Ex: a task scheduled to run periodically)

- OpenMP: Set compiler directives and API for C, C++, FORTRAN
  - ↳ 1) Provide support for parallel programming in shared memory environments
    2) Identifies parallel regions - blocks of code that can run in parallel.

  [#pragma omp parallel] Creates as many threads are there are processing cores

- Grand Central dispatch: Apple tech for Max OS X and IOS
  - ↳ Extensions of C, C++, API, run-time library that allows developers to identify sections of codes to run in parallel (OpenMP)

- Chapter 4- cont...

- Block: is a self-contained unit of work ^{ Some code };
  - ↳ are scheduled by being placed in dispatch queue.
  - , when it is removed from the queue, it is assigned to available thread from the thread pool.

- dispatch queue
  - ↳ 1) Serial (FIFO), each process has it's own serial queue known as (main queue)
  - 2) Concurrent (FIFO), but several blocks may be removed at once. with priority: low, default, hight

- Signals in UNIX are used to notify a process that a particular event has occured.
- Signal handler is used to process signals
  - ↳ Signals are: 1) generated by particular event
    - 2) delivered to a process
    - 3) handled by one of two signal handlers (default, user-defined

- Every signal has default handler runs by the kernal
  - ↳ can be override by user-defined signal handler
  - → For ~~sig~~ single-thread; signal delivered to process.

- For multithreaded processes:
  - ↳ 1) signals is ~~sa~~ delivered to the thread which it applies
  - 2) signal is delivered to every thread in the process.
  - 3) signal is delivered to a certain thread in the process.
  - 4) Assign a specific thread to receive ~~#~~ all signals for the process.

- thread to be canceled is target thread.
- Asynchronous cancellation: the target is terminated immediately.
- Deferred cancellation: allows the target to check periodically if it should be cancelled.

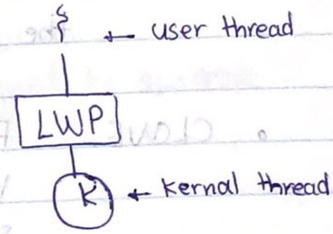
```
pthread_create(&tid, 0, worker, NULL); / pthread_cancel(tid);
pthread_t tid;
```

- thread cancellation requests cancellation
  - ↳ actual cancellation depends on thread state.
- If thead has cancellation disabled, cancellation remains pending until thread enables it.
- Default type is deferred.
  - → Cancellation only occurs when thread reaches cancellation point
    - pthread_testcancel()
    - then cleanup handler is invoked.
- Thread cancellation is handled through signals in Linux.
- Thread Local Storage (TLS): each thread have its own copy of data
  - ↳ 1) useful when you do not have control over thread creation (ex: when using thread pool)
  - 2) Different from local variables → TLS visible across func invocations; Local variables visible only during single func invocation
  - 3) Similar to static data (TLS is unique to each thread)

(Scheduler activation)

- an interminate data structure between user and kernal threads is used (lightweight process (LWP)) in both M:M and two-level models

when requiring communication to maintain the appropriate number of kernal threads allocated to the application



→ it Appears to be virtual process on which process can schedule user threat to run.

+ Each LWP is attached to kernal thread.

- Scheduler activations provide upcalls

→ upcalls: a communication mechanism from the kernal to the upcall handler in the thread library.

→ it allows an application to maintain the correct number kernal threads.

- Windows threads
  1) implements the 1:1 mapping, kernal level
  2) Each thread contains: - thread id
     - Register set
     - Seperate user and kernal stack
     - Private data storage area used by run-time and dynamic link libraries

  Context of the thread ⇐ (Register set, Seperate user and kernal stack, Private data storage area)

- Primary data structures in windows threads includ:
  1) ETHREAD (executive thread block) ~~includes pointer to KTHREAD~~
  2) KTHREAD (kernal thread block)
  3) TEB (thread environment block) - thread id, user-mode stack, TLS in user space.

  (KTHREAD) → Scheduling and synchronization info, kernal-mode stack, pointer to TEB in kernal space.

  (ETHREAD) → includes pointer to process to which thread belongs to KTHREAD in kernal space.

- Linux threads (or tasks)
  - clone() for thread creation
    - allows a child task to share the address space of the parent task (process)

- CLONE_ FS: File-Sys information is shared
  - VM: the same memory space are shared
  - SIGHAND: signal handlers are shared
  - FIIES: set of open files is shared

- struct task_struct points to process data structures (shared or unique)