1 CheatSheet: Concurrency & Parallel Programming INTERVIEW

- PDF Link: cheatsheet-concurrency-A4.pdf, Category: interview
- Blog URL: `https://cheatsheet.dennyzhang.com/cheatsheet-concurrency-A4`
- Related posts: CheatSheet: Leetcode For Code Interview, CheatSheet: Well-Known Papers For IT Industry, #denny-cheatsheets

File me Issues or star this repo.

1.1 Concurrency Concepts

Num	Name	Summary
1	Inter-process communication	Pipe; Signal; Shared memory; MQ; socket; RPC
2	Synchronization primitives	mutex, semaphore
3	Atomic operations	Test-and-set; `GET_ADD`; Redis INCR; CPU CAS;
4	Spinlocks	Locks which spin on mutex. Continuously poll until condition gets met
5	Sleeping locks	Put threads to wait queue.
6	Critical section	The code between the lock and unlock calls to the mutex
7	Mutex	MUTual EXclusion
8	Semaphores	It solves the problem of lost wakeup calls. Semaphores: binary and counting
9	Conditional variables	A queue of threads, associated with a monitor
10	futex	A fast userspace mutex
11	Starvation(Lived Lock)	When a thread waits for an indefinite period of time to get the required resource
12	Recursive Mutexes	Re-entrant lock
13	Reader/Writer Mutexes	
14	Dead lock	
15	Memory barrier	
16	Callbacks	
17	Per-CPU locking	
18	Asynchronous I/O	
19	Thread Design Patterns	Thread pool, Peer and Pipeline
20	Actor model vs CSP model	
21	Reference	Link: Java Concurrency and Multithreading Tutorial

1.2 Concurrent Implementation In Common Languages

Num	Name	
1	Java concurrent source code	Github: jdk7u-jdk/src/.../concurrent
2	Golang concurrent source code	Github: golang/go/tree/master/src/sync

1.3 Well-known Concurrent Problems

Num	Name	
1	ABA problem: loss updates	
2	Readers-writers problem	Read/write access the same shared resource at one time
3	Producer-consumer problem	a.k.a the bounded-buffer problem
4	Dining philosophers problem	Leetcode: The Dining Philosophers
5	Cigarette smokers problem	Assume a cigarette requires 3 ingredients: tobacco, paper, and matches
6	Sleeping barber problem	Keep a barber working when there are customers, resting when none
7	Guarded suspension	

1.4 Top 10 Concurrency Design Problems

Num	Name	
1	How to implement a spinlock	Github: link
2	How to implement a mutex	Github: link
3	How to implement a condition variable	Github: link
4	How to implement a reader-writer locker	Github: link
5	How to implement a bounded blocking queue	Github: link
6	Create two threads cooridnated by mutex in C	Github: code-example/threads/thread$_{\text{mutex.c}}$
7	IPC: use shared memory without kernel copy	Github: code-example/shared-memory
8	Support in-memory kv store transactions	Github: link
9	Design a thread-safe Hashmap	
10	Delayed task scheduling	
11	Implement a lock-free queue with multiple readers/writers	Github: link
12	Implement a api rate limiter with token bucket algorithm	

1.5 Top 10 Concurrency Coding Problems

Num	Problem	Summary
1	Semaphores to maintain the order	Leetcode: Building H2O
2	Web Crawler Multithreaded	LeetCode: Web Crawler Multithreaded
3	Print Zero Even Odd	Leetcode: Print Zero Even Odd
4	Map/Reduce: scheduler + workers	Leetcode: Fizz Buzz Multithreaded
5	Design Bounded Blocking Queue	Leetcode: Design Bounded Blocking Queue
6	Avoid deadlock and starvation	Leetcode: The Dining Philosophers
7	Claim ownerhip of a single resource	LeetCode: Traffic Light Controlled Intersection

1.6 POSIX thread C library

Num	Summary	Function
1	Create a thread	`pthread_create(&handler, &attr, func, arg);`
2	Exit a thread	`pthread_exit(exit_status);`
3	Cancel a thread	`pthread_cancel(handle);`
4	Parent wait threads to finish	`pthread_join(handle, &exit_status);`
5	Parent detach a thread	`pthread_detach(handle);`
6	mutex lock	`pthread_mutex_lock(&mylock);`
7	mutex unlock	`pthread_mutex_unlock(&mylock);`

1.7 More Resources

`https://www.linkedin.com/pulse/locks-mutex-semaphore-deadlock-starvation-mohammad-fares/`

- Github: angrave/SystemProgramming
- Wikipedia: Concurrent computing
- Link: Multithreaded Programming (POSIX pthreads Tutorial)
- Link: The Secret To 10 Million Concurrent Connections -The Kernel Is The Problem, Not The Solution