Quizlet

Operating Systems - Concurrency

Study online at quizlet.com/_2a4w8m

atomically	instruction execution is not interrupted
coarse-grained locking	one big lock over the critical sections
compare and swap	adds if statement to test and set so doesn't change value unless unlocked
condition variables	used to communicate between threads
critical section	area of access to shared resource

critical section lock example	pthread_mutex_t lock;
	pthread_mutex_lock(&lock);
	x = x + 1; // or whatever your critical section is
	pthread_mutex_unlock(&lock);

deadlock	two competing threads are waiting for the other to finish and then never finish
disabling interrupts for locking	Pro: simple Cons: 1. High trust in threads 2. Doesn't work on multiprocessor 3. Miss interrupts
evaluating locks	1. Mutual Exclusion 2. Fairness -> no thread starved 3. Performance -> lock overhead good?
fine-grained locking	many locks over critical sections allowing for many threads to be running at once
indeterminate	one or more race conditions w/ various outputs
initialization of condition variables	pthread_cond_t cond = PTHREAD_COND_INITIALIZER
initialization of locks	pthread_lock_t = PTHREAD_MUTEX_INITIALIZER
initialization of mutex lock	- pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
locks	provide mutual exclusion to critical sections
multithreaded	more than one point of execution
process control block	saves state of each process
pthread_cond_signal	pthread_cond_signal(pthread_cond_t* cond)
pthread_cond_wait	pthread_cond_wait(pthread_cond_t **cond, pthread_lock_t** lock)
pthread_create	adds processes to scheduling
pthread_create()'s arguments	pthread_create(pthread_t **p, pthread_attr attributes, void start_routine, void args**);
pthread_join	function that waits for processes to finish
pthread_join()'s arguments	pthread_join(pthread_t p, void value_ptr);
pthread_mutex_destroy	used when done with lock
pthread_mutex_init	dynamic lock initialization
pthread_mutex_lock	function where if placed before critical section: locks it
pthread_mutex_t	mutual exclusion lock type
pthread_mutex_unlock	unlocks section of code, placed after critical section
pthread_t	thread variable type (pthread_t p1, p2;)
race condition	multiple threads enter critical section at similar times
test and set (atomic exchange)	both the test and set are done in one atomic operatic

TestAndSet (code)	int ??????(int **old_ptr, int** new_ptr){
	int old = old_ptr;
	*old_ptr = new;
	return old;
	}

thread	single running process
thread control block	saves state of each thread - one stack per thread - places threads registers/variables/etc in local storage