Unit Exam: Concurrency

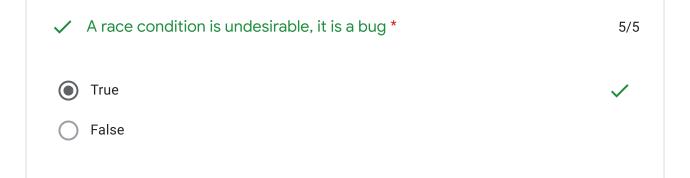
Total points 110/110

Take this exam alone. It is closed book, closed notes, but feel free to use a single 8.5" x 11" sheet of paper (both sides) of notes. Take the exam only once.

The respondent's email (parth2@pdx.edu) was recorded on submission of this form.

✓ All concurrent programs are parallel programs *	5/5
O True	
False	~

While one thread holds a mutex, other threads may lock or unlock that mutex *	5/5
True	
False	✓



	Concurrent Execution	Not Necessarily Concurrent Execution	Score	
a C program initializes the pthreads library	0		1/1	~
a program calls functions recursively	0		1/1	✓
a DBMS executes multiple queries at the same time	•	0	1/1	✓
while a process is executing, the operating system de-schedules the running process to handle an I/O interrupt		0	1/1	✓
a timer interrupt occurs while a user-level process executes within a loop		0	1/1	✓
a java method is declared as "synchronized"	0		1/1	✓
a web server handles multiple HTTP requests at the same time	•	0	1/1	✓

~	What do we call a concurrent read (or write) of an unprotected shared data value? *	5/5
0	race condition	
0	deadlock	
•	data race	✓
0	prioritized	
0	concurrent	
0	order violation	
/	the C language statement "num++;" is an atomic operation *	5/5
0	True	
•	False	✓

~	Which fundamental operation is often used to implement mutex locks or Intel CPUs? *	า 5/5
0	TestAndTestAndSet	
0	FetchAndAdd	
0	TestAndSet	
0	CompareAndSwap	
0	LoadLink	
•	xchg	✓
~	When pthread code calls signal() on a condition variable, which of the threads waiting on that condition variable will wake up and run as a resul of the signal() *	5/5 t
~	threads waiting on that condition variable will wake up and run as a resul	-
0	threads waiting on that condition variable will wake up and run as a resul of the signal() *	-
	threads waiting on that condition variable will wake up and run as a result of the signal() * none of the threads waiting on the condition variable	-
<!--</th--><td>threads waiting on that condition variable will wake up and run as a result of the signal() * none of the threads waiting on the condition variable only the thread that most recently began to wait on the condition variable</td><td>-</td>	threads waiting on that condition variable will wake up and run as a result of the signal() * none of the threads waiting on the condition variable only the thread that most recently began to wait on the condition variable	-

~	According to the lecture/slides, how many condition variables are needed for a bug-free implementation of the Producer/Consumer pattern? *	5/5
0	as many as the total number of threads in the system	
•	2	/
0	1	
0	none	
0	as many as the number of slots in the shared buffer	
✓	If multiple threads are waiting on a condition variable, and one other thread calls signal() on that condition variable, then which of the waiting thread(s) will be awoken? *	5/5
0	at least one, maybe more	
0	the thread that started to wait most recently	
•	one will be chosen randomly	/
0	the thread that started to wait earliest	
✓	When you need your threads to take turns then you should use condition variables *	5/5
	True	/
0	False	

Use of mutex is optional when using condition variables *	5/5
TrueFalse	✓
When a thread returns from a wait() call it will be holding the asso mutex *	ciated 5/5
TrueFalse	~
✓ Data races are the most common type of concurrency bug *	5/5
True	✓
○ False	
Fix times for concurrency bugs tend to vary more than for other to bugs. *	types of 5/5
TrueFalse	✓
O raise	

	Atomicity Violation bugs generally can be fixed with proper use of mutex ocks. *	< 5/5
T	True	✓
_ F	False	

	Concurrency Bug	Other type of bug	Score	
Order Violation		\bigcirc	1/1	✓
Segmentation Fault	\circ		1/1	~
Data Race		\bigcirc	1/1	✓
Illegal Instruction	0	•	1/1	~
beetle	\circ		1/1	✓
Atomicity Violation		0	1/1	~
Starvation		\circ	1/1	✓
Infinite Recursion	0		1/1	~
Deadlock		\circ	1/1	✓
Null Pointer Dereference			1/1	./

	Essential for deadlock	Not essential for deadlock	Score	
circular chain of locks	•	0	1/1	✓
paging	0	•	1/1	✓
condition variables	0	•	1/1	~
mutual exclusion		\circ	1/1	✓
no pre-emption		0	1/1	✓
offending Bruce's dog	0	•	1/1	~
recursion	0		1/1	✓
hold one lock while waiting for another		0	1/1	✓
(a lock) while	e trying to acqu	ogress because a th uire something else oncurrency bug? *		
Order Violatio	on			
Atomicity Vio				

!

Deadlock

✓ Database Servers sometimes use effects of deadlocks. *	to reduce the negative 5/5
deadlock detection	✓
random ordering of locks	
onon-preemptible locks	
mutual exclusion locks	
hold-and-wait	

This form was created inside of Portland State University.

Google Forms