Report II

Question: 2. In theory, preemptive shortest job first scheduling can always have better average turnaround time than non-preemptive shortest job first.

My answer: False

Correct Answer: True

Why I chose "False":

I got confused by the word **"always."** I was thinking about real-world issues, like unknown burst times, preemption overhead, and fairness, etc, where preemptive SJF doesn't always perform better. I also remembered that when all jobs arrive at once, preemption makes no difference, so I mistakenly thought it meant "not always better" indeed.

Why it's True:

Under the classic assumptions (job sizes known, zero context-switch cost, single server), SRPT minimizes mean flow/sojourn/turnaround time among all policies. If jobs arrive over time, SRPT can preempt a long job when a shorter one arrives, strictly improving (or at least never worsening) average turnaround versus non-preemptive SJF. When all jobs arrive at time 0, SRPT collapses to SJF, so they tie, still not worse. Hence "always better (or equal)" in theory. [1]

Pointers:

- SRPT has "long been known to be optimal for minimizing mean response time," formal proofs in queueing/scheduling theory. [2]
- Schrage's classic proof of SRPT optimality; subsequent analyses extend the result. [3]
- Lecture notes/text resources noting SJF is optimal (non-preemptive) when all jobs are available at time 0; with arrivals, SRPT preemptive variant dominates. [4]

References:

[1] L. Schrage, "A proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, vol. 16, no. 3, pp. 687–690, 1968.

[2] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge, MA: Cambridge University Press, 2013.

[3] P. Brucker, *Scheduling Algorithms*, 5th ed. Berlin, Germany: Springer, 2007.

[4] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Hoboken, NJ: Wiley, 2018.

Week 10 Test (Total Marks Capped at 40)

PART A. MULTIPLE-CHOICE (20 marks, single answer, please put answers in [ ])

[A] 1. Concurrent computing can be considered that computing tasks are done simultaneously _____.
A. from observer's point of view B. from system's point of view
[D] 2. Which of the following computing paradigm support parallel computing tasks? _____.
A. distributed computing B. clustering computing C. grid computing D. all of the above
[C] 3. Which of the following process state transition cannot happen? _____.
A. ready->running B. running->ready C. blocked->running D. running->blocked E. none of the above
[A] 4. In Unix, which system call can create a new process? _____.
A. fork B. create C. exec D. both A and C
[C] 5. Which of the following system calls does not return control to the calling point after finished? _____.
A. fork B. wait C. exec D none of the above
[D] 6. Which statement correctly corresponds to the turnaround time of a job? _____.
A. the total waiting time for a process to finish the execution B. the total time spent in the ready queue
C. the total time spent in the running queue D. the total time from the submission till the completion
[B] 7. Which of the following scheduling strategy is most likely to suffer from starvation? _____.
A. first come first serve B. preemptive shortest job first C. round robin D. multi-level feedback queue
[B] 8. Which one of the following is not a valid state of a thread?
A. running B. parsing C. ready D. blocked
[C] 9. Which of the following does a thread not own a private copy? _____.
A. program counter B. stack C. heap D. stack pointer E. none of the above
[D] 10. The number of the threads in the pool can be decided on factors such as: _____.
A. number of CPUs B. amount of physical memory C. expected number of client requests D. all of the above
[B] 11. A unit we do not want to interrupt is regarded as: _____.
A. single B. atomic C. static D. none of the above
[D] 12. Which of the following is required by a good lock? _____.
A. can prevent multiple threads from entering a critical section B. low overhead added by using the lock
C. ensure each thread contending the lock gets a fair shot at acquiring it once it is free D. all of the above
[D] 13. Which is the least important issue of implementing a lock by disabling interrupts? _____.
A. requires too much trust in applications B. does not work on multiple processors
C. causes other important interrupts ignored D. very inefficient when implemented
[B] 14. The main disadvantage of spinlocks is that: _____.
A. they are not sufficient for many process B. they require busy waiting
C. they are unreliable sometimes D. they are too complex to implement for programmers
[C] 15. Which hardware function or method can help implement a fair spin lock? _____.
A. test-and-set B. compare-and-swap C. fetch-and-add D. all of A,B,C E. none of A,B,C
[A] 16. If processes P1, P2, and P3 are all requesting resource R, but the operating system allows P1 and P2 to access R repeatedly while P3 continues to wait for R, that is called: _____.
A. starvation B. livelock C. deadlock D. busy waiting
[C] 17. Which of the following structure is the main part of a condition variable? _____.
A. a boolean variable B. an integer variable C. a queue D. a stack
[A] 18. Which of the following needs a mutex/lock reference to be passed in? _____.
A. pthread_cond_wait() B. pthread_cond_signal() C. both D. neither
[D] 19. Which of the following is not used to help with concurrent control? _____.
A. lock B. condition variable C. semaphore D. livelock
[D] 20. In the Dining Philosopher problem, changing the order of one philosopher's fork-pickup actions, is to tackle which condition to prevent deadlock? _____.
A. mutual exclusion B. hold-and-wait C. no pre-emption D. circular wait E. none of the above

PART B. TRUE-FALSE (T/F) QUESTIONS (10 marks, please put answers in [ ])

[T] 1. Round robin falls under the category of preemptive scheduling.
[F] 2. In theory, preemptive shortest job first scheduling can always have better average turnaround time than non-preemptive shortest job first.
[F] 3. In multi-level feedback queue, jobs of the same priority will be scheduled by first come first serve.
[T] 4. In Java, ReentrantLock can provide concurrency functionalities that synchronization blocks cannot provide.
[F] 5. If one thread of a process is holding the lock on a mutex, and another thread of the process attempts to lock the mutex, the whole process is blocked.
[F] 6. If there is no thread waiting on a condition variable, when pthread_cond_signal() is called, the signal will be remembered, and the next thread to wait on the condition variable will not block.
[F] 7. When using lock & condition variable for concurrency control, it is recommended to always signal() before unlock(&mutex), because, otherwise, it may cause deadlocks.
We can use resource graph to help detect deadlocks.
[T] 8. If a resource allocation graph has no cycles, there will be no deadlock.
[F] 9. If a resource allocation graph contains a cycle, there will be a deadlock.
[T] 10. Reboot is considered as an option to recover from deadlocks if deadlocks are rare.

PART C. SHORT-ANSWER QUESTIONS (10 marks, please put answers in ____)

1. What is the advantage of a shorter scheduling quantum? _____better responsiveness and latency for interactive jobs, improve fairness (reducing likelihood of long-running processes seizing the CPU)_____
2. What is the advantage of a longer scheduling quantum? _____improve CPU efficiency by reducing overhead from frequent context switches, better CPU throughput for CPU-bound jobs_____
3. In Java, if a class *A* has a static variable *b*, how can you ensure a thread can safely assess the static variable? Method 1__Synchronize on the class object__Method 2__Use a thread-safe primitive (declare volatile, using ReentrantLock to guard, or using Atomic* for updates)_____
4. We often see code like “while (flag == 0) Pthread_cond_wait(&cond, &mutex)”. Here why it is better to use “while”, rather than “if (flag == 0) Pthread_cond_wait(&cond, &mutex)”?___ handle spurious wakeups and re-check the predicate; multiple waiters/signals require revalidation after waking___
5. A semaphore is initialised as 2. Now it is value is -1. This means how many threads are waiting? __1___
6. What concurrency primitive tools can you use to fix atomicity violation bugs and order violation bugs?
Atomicity violation _____mutex or lock_____Order violation____condition variable or semaphore_____
7. Assume the following code will run to complete (ignoring syntax errors),
how many processes will be generated? ___24__ how many lines will be printed? ___25____

```
int main(void) {
        printf(“this is one line\n”);
        pid_t  pid = fork();
        pid = fork();
        pid = fork();
        if ( pid == 0)
                fork();
        fork();
        printf(“this is one line\n”);
}
```

PART D. SUDOKU (1 mark) – optional

3	2	6	4	2	7	1	9	8
8	5	9	6	5	1	4	7	3
1	7	4	9	8	3	5	2	6
4	3	7	1	9	5	8	6	2
9	8	5	2	4	6	3	1	7
6	1	2	3	7	8	9	5	4
2	6	3	5	1	4	7	8	9
5	4	8	7	6	9	2	3	1
7	9	1	8	3	2	6	4	5