

CS241 SP15 Exam 4: Solution Key

Name: Geng, Y.

UIN: 656934543

Exam code: BACABE

NetID: yangeng2

SCROLL TO THE NEXT PAGE TO REVIEW YOUR ANSWERS

A VERSION OF THESE QUESTIONS MAY APPEAR IN A FUTURE QUIZ

1. (1 point.) Two threads call `pthread_mutex_lock` on the same mutex. Which one of the following best describes what happens next?

- (A) The mutex lock is decreased by two
- (B) One thread will continue, the other thread must wait until the mutex is unlocked
- (C) The result is undefined
- (D) The mutex lock is increased by two

2. (1 point.) Identify the two missing pieces to complete Dekker's N=2 solution.

```
raise my flag
[X]? flag is raised :
  [Y] :
    lower my flag
    wait while your turn
    raise my flag
// Do Critical Section stuff
set your turn to win
lower my flag
```

- (A) $X = \text{while my}$ and $Y = \text{wait if it's your turn to win}$
- (B) $X = \text{while my}$ and $Y = \text{wait if it's my turn to win}$
- (C) $X = \text{if your}$ and $Y = \text{if it's my turn to win}$
- (D) $X = \text{while your}$ and $Y = \text{if it's your turn to win}$
- (E) $X = \text{if your}$ and $Y = \text{if it's your turn to win}$

3. (1 point.) Solve my riddle! Five threads call my X function but only three may continue; the other two threads must wait! Later my Y function is called once more and one of the two waiting threads is allowed to continue. What am I and what is X and Y ?

- (A) I am a condition variable, X is `pthread_cond_wait` and Y is `pthread_cond_signal`.
- (B) I am a counting semaphore, X is `wait` and Y is `post`.
- (C) I am a critical section, X is `start` and Y is `end`.
- (D) I am a mutex, X is `fork-exec` and Y is `waitpid`.
- (E) I am a mutex, X is `pthread_mutex_lock` and Y is `pthread_mutex_unlock`.

4. (1 point.) Which of the following is FALSE for condition variables?
- (A) During `pthread_cond_wait()` the mutex is automatically unlocked and later relocked before returning.
 - (B) Occasionally a call to `pthread_cond_wait()` may return even without any corresponding `pthread_cond_signal()` or `pthread_cond_broadcast()` call
 - (C) Condition variables use a helper mutex lock which must be locked before calling `pthread_cond_wait()` .
 - (D) Condition variables are initialized with a user-supplied condition callback function that returns 0(wait) or 1(continue).
 - (E) A thread can wake up one or all threads that are waiting on a condition variable.

5. (1 point.) Which response best describes “Bounded Wait”?
- (A) Before sleeping or performing slow I/O during a critical section, threads must preemptively unlock the mutex.
 - (B) If a thread is waiting to enter the critical section (CS), then atomic exchange assures waiting time is limited to less than N CPU instructions.
 - (C) A thread inside the critical section may only sleep for a finite number of milliseconds before continuing.
 - (D) If a thread is waiting to enter the critical section (CS), then other threads may only enter the CS first, a finite number of times.
 - (E) Multi-threaded performance is only guaranteed if threads do not sleep inside the critical section.

6. (1 point.) Solve my riddle! Whenever a thread calls my X function it must always wait! Later when my Y function is called then one waiting thread (if there is one) is released and allowed to continue. What am I and what is X and Y ?
Hint: I occasionally release blocked threads for no reason!

- (A) I am a counting semaphore, X is `sem_wait` and Y is `sem_post`.
- (B) I am a mutex, X is `pthread_mutex_lock` and Y is `pthread_mutex_unlock`.
- (C) I am a critical section, X is `pthread_create` and Y is `pthread_join`.
- (D) I am a condition variable, X is `pthread_cond_wait` and Y is `pthread_cond_signal`.
- (E) I am a process, X is `fork-exec` and Y is `waitpid`.

7. (1 point.) Which response is an example of “Deadlock”?
- (A) When two processes cannot continue because they are both waiting for an event from each other.
 - (B) When a mutex is destroyed but another thread calls `pthread_mutex_lock` on the same mutex.
 - (C) When a mutex cannot be unlocked because it was locked from another thread.
 - (D) When a mutex is transformed into an inconsistent state because it was initialized twice.
 - (E) When a mutex is transformed into an inconsistent state because it was destroyed and then re-initialized.

8. (1 point.) Which response best describes “Progress”?
- (A) The CPU clock may not be reset to an earlier time.
 - (B) In a multi-threaded program, a thread must preemptively unlock the mutex if it is required by another thread.
 - (C) If there are no threads inside the critical section, a thread should be able to enter immediately.
 - (D) Two threads may not perform terminal I/O at the same time.
 - (E) Only one thread may lock a mutex lock a time.

9. (1 point.) Solve my riddle! Whenever a thread calls my X function it should later call my Y function. If two or more threads call X then I shall declare a winner and the other(s) will have to wait! What am I and what is X and Y ?

- (A) I am a critical section, X is `pthread_create` and Y is `pthread_join`.
- (B) I am a process, X is `fork-exec` and Y is `waitpid`.
- (C) I am a mutex, X is `pthread_mutex_lock` and Y is `pthread_mutex_unlock`.
- (D) I am a counting semaphore, X is `sem_wait` and Y is `sem_post`.
- (E) I am a condition variable, X is `pthread_cond_wait` and Y is `pthread_cond_signal`.

10. (1 point.) Complete the following by choosing the best response. On modern processors, implementations of mutex locks on multi-core machines require CPU support. The relevant characteristics of a suitable CPU instruction are:

- (A) it exchanges the contents of a data register and PC register and satisfies bounded waiting.
- (B) it inverts the bit pattern stored in one byte of memory and will never deadlock.
- (C) it exchanges the contents of a register and memory and is atomic.
- (D) it exchanges the contents of two data registers and is non-atomic.
- (E) it exchanges the contents of a data register and stack pointer and satisfies progress.

11. (1 point.) The computation thread must wait until the array is full. The array is filled by another thread that will also release the waiting thread once the array is ready. Which response best describes which synchronization primitive to use to complete this task?

- (A) A mutex lock or condition variable are good choices but a semaphore is not a good choice.
- (B) A mutex lock or semaphore are good choices but a condition variable is not a good choice.
- (C) A counting semaphore or condition variable are good choices but a mutex lock is not a good choice.
- (D) This cannot be implemented with a mutex lock, semaphore or condition variable.
- (E) A condition variable is good choice but a mutex lock or semaphore are not good choices.

12. (1 point.) “When two threads concurrently execute code inside a critical section”, is an example of.
- (A) Preemptive multi-threaded process
 - (B) Failed mutual exclusion
 - (C) Multi-threaded acceleration
 - (D) Livelock
 - (E) Progress

13. (1 point.) Which one of the following is FALSE for a mutex of type `pthread_mutex_t`, that was locked and then unlocked ?

- (A) The same thread that locked the mutex must have unlocked it
- (B) The lock can now be destroyed using `pthread_mutex_destroy`
- (C) The unlock call will never block
- (D) Another waiting thread that already called `pthread_mutex_lock`, can now lock the mutex and proceed
- (E) The lock must be unlocked by two threads before it can be locked again.

14. (1 point.) Which one of the following is a NOT an example an atomic operation (behaves as if it is a single uninterruptible operation)?

- (A) When the post increment operator is used to increment an integer variable, `i++`.
- (B) When `cond_post` increments the semaphore's internal counter.
- (C) When the XCHG (exchange) or test-and-set CPU instruction reads and writes to main memory.
- (D) When `pthread_mutex_lock` locks a mutex.

15. (1 point.) Which response best describes the following code? Assume the value of `pthread_t` is the id of the calling thread.

```
pthread_t turn; /* shared between threads */

void lock(pthread_t id) {
    turn = id;
    while( turn == id) { /* now busy wait until the other thread calls lock*/};
}
void unlock() { }
```

- (A) Incorrect lock implementation (suffers from a race condition) and does not satisfy Bounded Wait
- (B) A correct implementation of a mutex lock
- (C) This implementation is equivalent to Peterson's solution
- (D) Incorrect lock implementation (suffers from a race condition) and does not satisfy Progress (may deadlock)
- (E) Incorrect lock implementation (suffers from a race condition) and does not satisfy Mutual Exclusion

16. (1 point.) Complete the following by choosing the best response. Simple implementations of correct solutions to the critical section problem may fail on some architectures because ...

- (A) CPU speeds (instructions per second) are now faster than main-memory read-write access.
- (B) Some programs are only single threaded.
- (C) Memory reads can deadlock.
- (D) For performance, the CPU and compiler may re-order instructions and cache reads may be stale.
- (E) It is not possible to implement critical section problem solutions in real software.

17. (1 point.) Complete the following. Which best describes two well known solutions to the *The Critical Section Problem*?

- (A) The first correct solution was published by von Neumann. Later, a simple solution was published by Ullman.
- (B) The first correct solution was published by Hopcroft. Later, a simple solution was published by Dijkstra.
- (C) The first correct solution was published by Dekker. Later, a simple solution was published by Peterson.
- (D) The first correct solution was published by Turing. Later, a simple solution was published by Dijkstra.
- (E) The first correct solution was published by Peterson. Later, a simple solution was published by Dekker.

18. (1 point.) Which response best describes the following attempt to solve the Critical Section Problem for two processes (or threads)? Assume both flags are initially down.

```
wait while my flag is up
raise your flag
// Perform critical section activities
lower your flag
```

- (A) Does not satisfy mutual exclusion
- (B) Does not satisfy progress but mutual exclusion is satisfied
- (C) Does not satisfy bounded wait but mutual exclusion is satisfied

19. (1 point.) Which response best describes the following code to ‘solve’ the Critical Section Problem? Assume both flags are initially down.

```
raise your flag
lower my flag
wait until my flag is up
// Perform critical section activities
raise my flag
lower your flag
```

- (A) Does not satisfy progress but mutual exclusion is satisfied
- (B) This is correct only for multi-threaded processes
- (C) Does not satisfy mutual exclusion
- (D) Does not satisfy bounded wait but mutual exclusion is satisfied
- (E) This is Turing’s solution

20. (1 point.) Which one of the following is NOT TRUE?
- (A) A program may `fork()` after initializing a mutex but by default the mutex is not shared between processes
 - (B) `PTHREAD_MUTEX_INITIALIZER` can be used on memory allocated from the heap
 - (C) Not calling `pthread_mutex_destroy` can lead to resource leaks because the mutex may include a pointer to a system-based synchronization primitive
 - (D) `pthread_mutex_init` is an alternative function to initialize a mutex
 - (E) Programs should not use the contents of `pthread_mutex_t` directly

21. (1 point.) Which one of the following is NOT TRUE?
- (A) Waiting on a condition variable should be wrapped in a loop (in part due to spurious wake ups)
 - (B) A counting semaphore can be implemented with a mutex lock and condition variable
 - (C) A condition variable is initialized with an integer counter.
 - (D) Underflow and overflow of a queue data structure can be prevented using counting semaphores
 - (E) A pthread mutex lock can be easily replaced with a counting semaphore (albeit with a slight loss of performance)

22. (1 point.) Which response best describes the code below? Each process or thread has it's own flag plus there is a shared-variable named `turn`.

```
raise my flag
Set turn to you
wait while (your flag is raised and it's your turn)
// Do Critical Section stuff
lower my flag
```

- (A) Does not satisfy bounded wait but satisfies mutual exclusion
- (B) None of the other responses are correct
- (C) Does not satisfy mutual exclusion
- (D) Does not satisfy progress but satisfies mutual exclusion
- (E) This is Peterson's N=2 solution

23. (1 point.) Which response best fills in the three blanks: When a program calls `pthread_create`, a new ? and ? are created ? the current process.

- (A) thread ; stack ; inside
- (B) process ; stack ; outside
- (C) thread ; heap ; outside
- (D) process ; thread ; by forking
- (E) thread ; heap ; inside

24. (1 point.) Solve my riddle! For the correct functioning of my program, only one thread of execution may concurrently execute inside this region of code because it accesses a shared resource. What is the name given to this region of code?

- (A) Race Condition
- (B) Bounded Region
- (C) Mutual Wait
- (D) Critical Section
- (E) Single-threaded Section

Summary of answers:

Question	Correct Answer	Your Answer	Points
1	B	B	1
2	D	D	1
3	B	B	1
4	D	D	1
5	D	D	1
6	D	D	1
7	A	A	1
8	C	C	1
9	C	C	1
10	C	C	1
11	C	E	0
12	B	B	1
13	E	E	1
14	A	A	1
15	D	D	1
16	D	D	1
17	C	C	1
18	A	A	1
19	A	A	1
20	B	B	1
21	C	C	1
22	E	E	1
23	A	A	1
24	D	D	1
Total			23