CS241 SP15 Exam 4: Solution Key

Name: Unni, N. **UIN:** 664350897

Exam code: ACCBDB **NetID:** nunni2

SCROLL TO THE NEXT PAGE TO REVIEW YOUR ANSWERS

A VERSION OF THESE QUESTIONS MAY APPEAR IN A FUTURE QUIZ

Name: Unni, N netid: nunni2 uin:664350897

---

1. (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 process, X is `fork-exec` and Y is `waitpid`.

(B) I am a counting semaphore, X is `sem_wait` and Y is `sem_post`.

(C) I am a mutex, X is `pthread_mutex_lock` and Y is `pthread_mutex_unlock`.

(D) I am a critical section, X is `pthread_create` and Y is `pthread_join`.

(E) I am a condition variable, X is `pthread_cond_wait` and Y is `pthread_cond_signal`.

Correct answer: E.
Your answer: C.
0 out of **1** point received

---

2. (1 point.) Using an initial heap size of 2^{10} bytes (1KB) and a binary buddy-allocator, how many memory allocation requests of 68 bytes can be completed before the allocator requires additional heap memory?

(A) 15

(B) 13

(C) 16 or greater

(D) 9 or fewer

(E) 14

Correct answer: D.
Your answer: D.
1 out of **1** point received

Solution. Round allocation requests up to nearest 2^n i.e. 128 bytes (2^7)

3. (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 unlock call will never block

(B) The lock must be unlocked by two threads before it can be locked again.

(C) The same thread that locked the mutex must have unlocked it

(D) The lock can now be destroyed using `pthread_mutex_destroy`

(E) Another waiting thread that already called `pthread_mutex_lock`, can now lock the mutex and proceed

Correct answer: B.
Your answer: B.
1 out of **1** point received

4. (1 point.) Identify the two missing pieces to complete Dekker's N=2 solution.

```
raise my flag
[X]? flag is raised :
   if it's your turn to win :
      lower my flag
      [Y]?
      raise my flag
// Do Critical Section stuff
set your turn to win
lower my flag
```

(A) $X =$ `while my` and $Y =$ `wait while my turn`

(B) $X =$ `if your` and $Y =$ `wait while your turn`

(C) $X =$ `if your` and $Y =$ `wait until your turn`

(D) $X =$ `while your` and $Y =$ `wait while your turn`

(E) $X =$ `while my` and $Y =$ `wait until your turn`

Correct answer: D.
Your answer: D.
1 out of **1** point received

5. (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 progress but mutual exclusion is satisfied

(B) Does not satisfy mutual exclusion

(C) Does not satisfy bounded wait but mutual exclusion is satisfied

Correct answer: B.
Your answer: A.
0 out of **1** point received

6. (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 exchanges the contents of a data register and stack pointer and satisfies progress.

(C) it exchanges the contents of two data registers and is non-atomic.

(D) it exchanges the contents of a register and memory and is atomic.

(E) it inverts the bit pattern stored in one byte of memory and will never deadlock.

Correct answer: D.
Your answer: D.
1 out of **1** point received

7. (1 point.) Which response best describes the following code?

```
int acquired = 0;  /* shared between threads */

void lock() {
  while( acquired != 0) {/* busy wait*/};
  acquired = 1;
}
void unlock() { acquired = 0; }
```

(A) Incorrect lock implementation (suffers from a race condition) and does not satisfy Mutual Exclusion

(B) A correct implementation of a mutex lock

(C) Incorrect lock implementation (suffers from a race condition) and does not satisfy Bounded Wait

(D) Incorrect lock implementation (suffers from a race condition) and does not satisfy Progress (may deadlock)

(E) This implementation is equivalent to Peterson's solution

Correct answer: A.
Your answer: D.
0 out of **1** point received

Solution. Consider two threads calling `lock()` at the same time. Both would see a *acquired* value of zero (0). Both would continue.

8. (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) This cannot be implemented with a mutex lock, semaphore or condition variable.

(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) A mutex lock or condition variable are good choices but a semaphore is not a good choice.

(E) A condition variable is good choice but a mutex lock or semaphore are not good choices.

Correct answer: C.
Your answer: A.
0 out of **1** point received

Solution. Spring 2015: Typo in one of the incorrect responses (condition variable-¿semaphore), but this does not affect the best answer

9. (1 point.) Which one of the following is NOT TRUE?

(A) A condition variable is initialized with an integer counter.

(B) A pthread mutex lock can be easily replaced with a counting semaphore (albeit with a slight loss of performance)

(C) Waiting on a condition variable should be wrapped in a loop (in part due to spurious wake ups)

(D) A counting semaphore can be implemented with a mutex lock and condition variable

(E) Underflow and overflow of a queue data structure can be prevented using counting semaphores

Correct answer: A.
Your answer: A.
1 out of **1** point received

10. (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) Does not satisfy bounded wait but mutual exclusion is satisfied

(C) Does not satisfy mutual exclusion

(D) This is correct only for multi-threaded processes

(E) This is Turing's solution

Correct answer: A.
Your answer: A.
1 out of **1** point received

11. (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) This is Peterson's N=2 solution

(C) None of the other responses are correct

(D) Does not satisfy mutual exclusion

(E) Does not satisfy progress but satisfies mutual exclusion

Correct answer: B.
Your answer: A.
0 out of **1** point received

12. (1 point.) Solve my riddle! Four threads call my X function but only two 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 counting semaphore, X is `sem_wait` and Y is `sem_post` .

(B) I am a critical section, X is `pthread_create` and Y is `pthread_join`.

(C) I am a mutex, X is `pthread_mutex_lock` and Y is `pthread_mutex_unlock` .

(D) I am a condition variable, X is `pthread_cond_wait` and Y is `pthread_cond_signal`.

(E) I am a mutex, X is signal and Y is waitpid.

Correct answer: A.
Your answer: A.
1 out of **1** point received

13. (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) Programs should not use the contents of pthread_mutex_t directly

(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) `PTHREAD_MUTEX_INITIALIZER` can be used on memory allocated from the heap

Correct answer: E.
Your answer: E.
1 out of **1** point received

Solution. `PTHREAD_MUTEX_INITIALIZER` can only be used with static (global) variables.

```
pthread_mutex_t m =  PTHREAD_MUTEX_INITIALIZER;
int dostuff() {
  pthread_mutex_lock(&m);
  pthread_mutex_unlock(&m);
}

int dostuff2() {
  pthread_mutex_t m2;
  pthread_mutex_init(&m2, NULL);
  pthread_mutex_lock(&m2);
  // Do Critical Section stuff here
  pthread_mutex_unlock(&m2);
  pthread_mutex_lock(&m2);
}
```

14. (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) Multi-threaded performance is only guaranteed if threads do not sleep inside the critical section.

(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 atomic exchange assures waiting time is limited to less then N CPU instructions.

(E) 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.

Correct answer: E.
Your answer: E.
1 out of **1** point received

15. (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 `cond_post` increments the semaphore's internal counter.

(B) When the XCHG (exchange) or test-and-set CPU instruction reads and writes to main memory.

(C) When `pthread_mutex_lock` locks a mutex.

(D) When the post increment operator is used to increment an integer variable, `i++`.

Correct answer: D.
Your answer: D.
1 out of **1** point received

16. (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 Dekker. Later, a simple solution was published by Peterson.

(B) The first correct solution was published by von Neumann. Later, a simple solution was published by Ullman.

(C) The first correct solution was published by Turing. Later, a simple solution was published by Dijkstra.

(D) The first correct solution was published by Hopcroft. 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.

Correct answer: A.
Your answer: A.
1 out of **1** point received

17. (1 point.) Which of the following is FALSE for condition variables?

(A) Occasionally a call to `pthread_cond_wait()` may return even without any corresponding `pthread_cond_signal()` or `pthread_cond_broadcast()` call

(B) During `pthread_cond_wait()` the mutex is automatically unlocked and later relocked before returning.

(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.

Correct answer: D.
Your answer: B.
0 out of **1** point received

18. (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 process, X is `fork-exec` and Y is `waitpid`.

(B) I am a counting semaphore, X is `sem_wait` and Y is `sem_post`.

(C) I am a mutex, X is `pthread_mutex_lock` and Y is `pthread_mutex_unlock`.

(D) I am a condition variable, X is `pthread_cond_wait` and Y is `pthread_cond_signal`.

(E) I am a critical section, X is `pthread_create` and Y is `pthread_join`.

Correct answer: C.
Your answer: C.
1 out of **1** point received

19. (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 mutex lock is increased by two

(D) The result is undefined

Correct answer: B.
Your answer: B.
1 out of **1** point received

20. (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) Bounded Region

(B) Single-threaded Section

(C) Mutual Wait

(D) Critical Section

(E) Race Condition

Correct answer: D.
Your answer: D.
1 out of **1** point received

21. (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) Some programs are only single threaded.

(B) CPU speeds (instructions per second) are now faster than main-memory read-write access.

(C) For performance, the CPU and compiler may re-order instructions and cache reads may be stale.

(D) Memory reads can deadlock.

(E) It is not possible to implement critical section problem solutions in real software.

Correct answer: C.
Your answer: C.
1 out of **1** point received

22. (1 point.) Which response best describes "Mutual Exclusion"?

(A) Multi-threaded performance is only guaranteed if one thread sleeps during I/O actions of the second thread.

(B) Two threads may not perform I/O at the same time.

(C) Only one thread of execution may be executing code inside the critical section at a time.

(D) Two threads may not be executing the same line of code at the same time.

(E) Before sleeping or performing slow I/O during a critical section, threads must preemptively unlock the mutex.

Correct answer: C.
Your answer: C.
1 out of **1** point received

23. (1 point.) "If there are no threads inside the critical section, a thread should be able to enter immediately." is an example of ... (pick the most appropriate response)

(A) Circular Wait

(B) None of the other responses are correct

(C) Livelock

(D) Mutual Exclusion

(E) No preemption

Correct answer: B.
Your answer: B.
1 out of **1** point received

Solution. And: Progress

24. (1 point.) Which response is an example of "Deadlock"?

(A) When a mutex is transformed into an inconsistent state because it was initialized twice.

(B) When two processes cannot continue because they are both waiting for an event from each other.

(C) When a mutex is destroyed but another thread calls `pthread_mutex_lock` on the same mutex.

(D) When a mutex is transformed into an inconsistent state because it was destroyed and then re-initialized.

(E) When a mutex cannot be unlocked because it was locked from another thread.

Correct answer: B.
Your answer: B.
1 out of **1** point received

Summary of answers:

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