Midterm2

Due No due date **Points** 11.7 **Questions** 7 **Available** Mar 21 at 12:45pm - Mar 21 at 1:53pm 1 hour and 8 minutes
Time Limit 45 Minutes

This quiz was locked Mar 21 at 1:53pm.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	32 minutes	3.22 out of 11.7

Score for this quiz: **3.22** out of 11.7
Submitted Mar 21 at 1:25pm
This attempt took 32 minutes.

Question 1

0.6 / 1.6 pts

The four conditions of deadlocks are (names starts with capital for only the first letter):

1. Lack of resources
2. Circular programing
3.
4.

Since they are absolute conditions, i.e., they have to be met for a deadlock to happen, one can "prevent" deadlock by failing just one of these four conditions. For example, for dinning lawyer problems of 5 lawyers, providing at least 1 additional chopstick(s) may prevent the deadlock. This approach fails condition # 1 . We may also require all lawyers to get "either all or none" chopsticks to prevent deadlock by failing condition # 4 .

Answer 1:

You Answered: Lack of resources

Correct Answer: Bounded resources

Answer 2:

You Answered: Circular programing

Correct Answer: No preemption

Answer 3:

You Answered: (You left this blank)

Correct Answer: Wait while holding

Answer 4:

You Answered

(You left this blank)

Correct Answer

Circular waiting

Answer 5:

You Answered

absolute

Correct Answer

necessary

Answer 6:

Correct!

1

Answer 7:

Correct!

1

Answer 8:

You Answered

4

Correct Answer

3

Question 2

0.8 / 0.8 pts

The output of the following code is

6

6

```
// Program 1
main() {
        int val = 5;
        int pid;

        if (pid = fork())
            wait(pid);
        val++;
        printf("%d\n", val);
        return val;
}
```

Answer 1:

Correct!

6

Answer 2:

Correct!

6

Question 3

0.3 / 2.1 pts

Now suppose that we will implement a RWLock class as follows using lock variables and condition variables. The requirements are that (i) readers do not conflict readers; (ii) writers conflict with both readers and writers; (iii) if there is any writer waiting to write, readers will have to yield.

Note: Please **make sure** to leave one blank space before and after each operator such as ==, %, +, / , etc.

```
class RWLock {
  private:
  Lock lock;
  CV readGo;
  CV writeGo;

  int activeReaders;
  int activeWriters;
  int waitingReaders;
  int waitingWriters;
  bool readShouldWait();
  bool writeShouldWait();
  public:
  RWLock();
  ~RWLock();
  void startRead();
  void doneRead();
  void startWrite();
  void doneWrite();
};
```

(1) Please complete the following function for **readShouldWait()**.

```
bool RWLock::readShouldWait() {
    return (activeWriters>0 || [waitingWriters>0] );
}
```

(2) Please complete the following function for **startWrite()**.

```
bool RWLock:: startWrite() {
    [void startWrite()]
    [activeWriters] ++;
    while(writeShouldWait()) {
        [waitingWriters]
    }
    [ ]
    activeWriters++;
    lock.release();
}
```

(3) Please complete the following function for **doneRead()**.

```
bool RWLock:: doneRead() {
    lock.acquire();
    [ ]
    if([ ] && waitingWriters>0)
        writeGo.signal(&lock);
    lock.release();
}
```

Answer 1:

Correct!

waitingWriters>0

Correct Answer

waitingWriters > 0

Answer 2:

You Answered: void startWrite()

Correct Answer: lock.acquire();

Answer 3:

You Answered: activeWriters

Correct Answer: waitingWriters

Answer 4:

You Answered: waitingWriters

Correct Answer: writeGo.wait(&lock);

Answer 5:

You Answered: (You left this blank)

Correct Answer: waitingWriters--;

Correct Answer: waitingWriters --;

Answer 6:

You Answered: (You left this blank)

Correct Answer: activeReaders--;

Correct Answer: activeReaders --;

Answer 7:

You Answered: (You left this blank)

Correct Answer: activeReaders==0

Correct Answer: activeReaders == 0

Question 4

0.3 / 1.2 pts

Please fill in the missing part of the following pseudo code for bounded buffer problem, assuming Mesa Semantics for condition variable. Buffer is an array/vector named "buf". Please make sure to leave one blank space before and after each operator such as ==, %, +, / , etc.

```
get() {
    lock.acquire();
    while( front>tail ) {
        signal .wait(&lock);
    }
    item = front-tail ;
    front++;
    full.signal(&lock);
    lock. release ();

    return item;
}
```

Initially: front = tail = 0; MAX is buffer capacity and **empty/full** are condition variables.

Answer 1:

You Answered: front>tail

Correct Answer: front == tail

Correct Answer: tail == front

Answer 2:

You Answered: signal

Correct Answer: empty

Answer 3:

You Answered: front-tail

Correct Answer: buf[front % MAX]

Correct Answer: buf[front % MAX]

Correct Answer: buf[front%MAX]

Answer 4:

Correct!: release

Question 5

0.92 / 3 pts

Suppose that three threads A, B, and C in a system are competing for a total of 8 pages of memory. A, B, and C needs 4, 5, and 5 pages to complete respectively. A, B, and C currently holds 3, 2, and 2 pages respectively. They take turns to request the remaining pages they need. **Now, B is the next thread to continue.** Show the detailed steps such that with **banker's algorithm**, all threads eventually get the pages they need and release all pages they hold.

For each of the blanks, enter either the number of pages currently being allocated for a specific thread. **If a thread is blocked due to an unsafe state (from banker's algorithm), enter W**.

Note: only one action at a time, e.g., release, request and acquire, wait. There is no cell intentionally left as blank.

Process							**Allocation**
A	3	3	2	4	0	0	0
B	2	w	w	w	3	3	4
C	2	2	w	w	3	3	w

Answer 1:

Correct! 3

Answer 2:

You Answered 2

Correct Answer 3

Answer 3:

Correct! 4

Answer 4:

Correct! 0

Answer 5:

Correct! 0

Answer 6:

Correct! 0

Answer 7:

You Answered (You left this blank)

Correct Answer 0

Answer 8:

You Answered (You left this blank)

Correct Answer 0

Answer 9:

You Answered (You left this blank)

Correct Answer 0

Answer 10:

You Answered (You left this blank)

Correct Answer 0

Answer 11:

You Answered (You left this blank)

Correct Answer 0

Answer 12:

You Answered (You left this blank)

Correct Answer 0

Answer 13:

You Answered (You left this blank)

Correct Answer 0

Answer 14:

Correct! W

Answer 15:

Correct! W

Answer 16:

Correct! W

Answer 17:

You Answered 3

Correct Answer W

Answer 18:

Correct! 3

Answer 19:

You Answered 4

Correct Answer 3

Answer 20:

You Answered (You left this blank)

Correct Answer 4

Answer 21:

You Answered (You left this blank)

Correct Answer 4

Answer 22:

You Answered (You left this blank)

Correct Answer 5

Answer 23:

You Answered (You left this blank)

Correct Answer 0

Answer 24:

You Answered (You left this blank)

Correct Answer 0

Answer 25:

You Answered (You left this blank)

Correct Answer 0

Answer 26:

You Answered (You left this blank)

Correct Answer 0

Answer 27:

Correct! 2

Answer 28:

Correct! W

Answer 29:

Correct! W

Answer 30:

You Answered 3

Correct Answer W

Answer 31:

You Answered 3

Correct Answer W

Answer 32:

You Answered w

Correct Answer 3

Answer 33:

You Answered (You left this blank)

Correct Answer 3

Answer 34:

You Answered (You left this blank)

Correct Answer W

Answer 35:

You Answered (You left this blank)

Correct Answer W

Answer 36:

You Answered (You left this blank)

Correct Answer W

Answer 37:

You Answered (You left this blank)

Correct Answer 4

Answer 38:

You Answered

(You left this blank)

Correct Answer

5

Answer 39:

You Answered

(You left this blank)

Correct Answer

0

Question 6

0.3 / 0.6 pts

```
Line#   // Thread A              // Thread B
1       lock1.acquire();         lock1.acquire();
2       ...                      ...
3       lock2.acquire();         lock2.acquire();
4       while(need to wait) {    ...
5           cv.wait(&lock2);     cv.signal();
6       }                        lock2.release();
7       ...                      ...
8       lock2.release();         lock1.release();
9       ...
10      lock1.release();
```

For above pseudo code, assuming that Thread A obtained lock1 and then lock2 successfully, deadlock will happen when Thread A just executed Line # 5 and is now busy waiting for Thread B to execute Line # 6 , which never happens since Thread A holds the lock.

Answer 1:

Correct!

5

Answer 2:

You Answered

6

Correct Answer

5

Question 7

0 / 2.4 pts

Please finish the following thread programming. Please use the simple Threads API from the textbook.

```
#define NTHREADS 10

static thread_t threads 100 ;

void go(int n) {
    cout << "child thread running!" << endl;
```

NTHREADS (100 + n); // terminate the thread

}

int main() {

for(int i = 0;i < n ;++i)

(& go , t threads ,); // create ith thread with go function and pass i as parameter

}

for(int i = 0;i < n ;++i)

int exitValue = ( ); // wait for ith thread to finish

cout << exitValue << endl;

}

return 0;

}

Answer 1:

You Answered

100

Correct Answer

[NTHREADS]

Correct Answer

NTHREADS

Answer 2:

You Answered

NTHREADS

Correct Answer

thread_exit

Answer 3:

You Answered

n

Correct Answer

NTHREADS

Answer 4:

You Answered

(You left this blank)

Correct Answer

thread_create

Answer 5:

You Answered

go

Correct Answer

threads[i]

Answer 6:

You Answered

t threads

Correct Answer &go

Answer 7:

You Answered (You left this blank)

Correct Answer i

Answer 8:

You Answered n

Correct Answer NTHREADS

Answer 9:

You Answered (You left this blank)

Correct Answer thread_join

Answer 10:

You Answered (You left this blank)

Correct Answer threads[i]

Quiz Score: **3.22** out of 11.7