Threads
How can arguments be passed to threads?

A. As an argument to pthread_create
B. As an argument to pthread_start
C. Using a thread-specific data interface
D. More than one of the above
E. Threads must work blind, without knowledge of their fellows slaving away in an indifferent world
How can arguments be passed to threads?

A. As an argument to pthread_create
B. As an argument to pthread_start
C. Using a thread-specific data interface
D. More than one of the above
E. Threads must work blind, without knowledge of their fellows slaving away in an indifferent world
What is the goal of having a fixed locking order?

A. Ordering locks helps prevent deadlock
B. Allows finer-grained locks to be protected by more coarse-grained locks
C. Increases efficiency by helping the library achieve locality
D. The order provides a hierarchy so the locks know which is more important; they like to boss each other around
E. None of the above
What is the goal of having a fixed locking order?

A. Ordering locks helps prevent deadlock
B. Allows finer-grained locks to be protected by more coarse-grained locks
C. Increases efficiency by helping the library achieve locality
D. The order provides a hierarchy so the locks know which is more important; they like to boss each other around
E. None of the above
Which of the following is a good alternative to a fixed locking order?

A. Using a single mutex instead of several
B. Allowing the system to occasionally break mutexes
C. A backoff strategy
D. Allowing deadlock and looking for another job
E. Not exactly one of the above
Which of the following is a good alternative to a fixed locking order?

A. Using a single mutex instead of several
B. Allowing the system to occasionally break mutexes
C. A backoff strategy
D. Allowing deadlock and looking for another job
E. Not exactly one of the above
When using a backoff strategy, which of the following should the thread do when it fails to get a mutex?

A. Wait a random amount of time before trying again
B. Block until another thread releases the mutex it failed to get
C. Switch the order in which it requests the mutexes
D. Release the other mutexes it holds
E. Not exactly one of the above
When using a backoff strategy, which of the following should the thread do when it fails to get a mutex?

A. Wait a random amount of time before trying again
B. Block until another thread releases the mutex it failed to get
C. Switch the order in which it requests the mutexes
D. Release the other mutexes it holds
E. Not exactly one of the above
What is a condition variable?

A. Fancy name for a boolean variable (aka “flag”)
B. A variable that is either empty or full. Reading an empty variable blocks. Reading a full one makes it empty. Opposite for writing
C. A variable that is tested just prior to entering a critical section
D. A variable that threads can wait on, blocking until signaled by another thread
E. None of the above
What is a condition variable?

A. Fancy name for a boolean variable (aka “flag”)
B. A variable that is either empty or full. Reading an empty variable blocks. Reading a full one makes it empty. Opposite for writing
C. A variable that is tested just prior to entering a critical section
D. A variable that threads can wait on, blocking until signaled by another thread
E. None of the above
void insert(Item x) {
    pthread_mutex_lock(&lock);
    while(buffer_is_full()) {
        pthread_cond_wait(&nonfull, &lock);
    }
    add_to_buffer(x);
    pthread_cond_signal(&nonempty);
    pthread_mutex_unlock(&lock);
}

Item remove() {
    pthread_mutex_lock(&lock);
    while(buffer_is_empty()) {
        pthread_cond_wait(&nonempty, &lock);
    }
    Item x = remove_from_buffer();
    pthread_cond_signal(&nonfull);
    pthread_mutex_unlock(&lock);
    return x;
}

To the left is pseudocode for bounded buffer. Why is the lock used to protect the condition variable calls (wait and signal) and not just the buffer-related ones?

A. So buffer state isn’t changed immediately before the calls
B. It must protect wait because there isn’t a way to not protect it while still protecting the loop conditions; protecting signal is optional
C. It must protect the signal calls, but only protects wait because it must protect the statement on either side
D. It doesn’t have to be (unnecessary, but not harmful)
E. It shouldn’t be (actually harmful)
void insert(Item x) {
    pthread_mutex_lock(&lock);
    while(buffer_is_full()) {
        pthread_cond_wait(&nonfull, &lock);
    }
    add_to_buffer(x);
    pthread_cond_signal(&nonempty);
    pthread_mutex_unlock(&lock);
}

Item remove() {
    pthread_mutex_lock(&lock);
    while(buffer_is_empty()) {
        pthread_cond_wait(&nonempty, &lock);
    }
    Item x = remove_from_buffer();
    pthread_cond_signal(&nonfull);
    pthread_mutex_unlock(&lock);
    return x;
}

To the left is pseudocode for bounded buffer. Why is the lock used to protect the condition variable calls (wait and signal) and not just the buffer-related ones?

A. So buffer state isn’t changed immediately before the calls
B. It must protect wait because there isn’t a way to not protect it while still protecting the loop conditions; protecting signal is optional
C. It must protect the signal calls, but only protects wait because it must protect the statement on either side
D. It doesn’t have to be (unnecessary, but not harmful)
E. It shouldn’t be (actually harmful)
void insert(Item x) {
    pthread_mutex_lock(&lock);
    while(buffer_is_full()) {
        pthread_cond_wait(&nonfull, &lock);
    }
    add_to_buffer(x);
    pthread_cond_signal(&nonempty);
    pthread_mutex_unlock(&lock);
}

Item remove() {
    pthread_mutex_lock(&lock);
    while(buffer_is_empty()) {
        pthread_cond_wait(&nonempty, &lock);
    }
    Item x = remove_from_buffer();
    pthread_cond_signal(&nonfull);
    pthread_mutex_unlock(&lock);
    return x;
}
void insert(Item x) {
    pthread_mutex_lock(&lock);
    while(buffer_is_full()) {
        pthread_cond_wait(&nonfull, &lock);
    }
    add_to_buffer(x);
    pthread_cond_signal(&nonempty);
    pthread_mutex_unlock(&lock);
}

Item remove() {
    pthread_mutex_lock(&lock);
    while(buffer_is_empty()) {
        pthread_cond_wait(&nonempty, &lock);
    }
    Item x = remove_from_buffer();
    pthread_cond_signal(&nonfull);
    pthread_mutex_unlock(&lock);
    return x;
}