POSIX Threads Quiz

CS/CE-232/324 - Operating Systems Lab

Fall 2024

6/10

Instructions

- Each question carries equal marks.
- Color the circle corresponding to the correct answer.
- There is only one correct answer for each question.
- Answer all questions.

Name: Breela Dasin

ID: 08283

✓ 1. (2 points) When multiple threads are created within a process, which of the following memory segments is NOT shared among them?

- ○ Heap segment.
- ○ Initialized data segment.
- ○ Uninitialized data segment.
- ● Thread-specific `errno` variable.

✓ 2. (2 points) What happens when the main thread exits before other threads complete their execution?

- ○ The program crashes.
- ○ Other threads continue running independently.
- ● Other threads are automatically joined and terminated.
- ○ Other threads become zombie processes.

✗ 3. (2 points) Which of the following best describes why threads are sometimes called "light-weight processes"?

- ○ Because they are cheaper to create and make data sharing easier compared to processes.
- ○ Because they run faster than regular processes.
- ○ Because they use less memory than processes.
- ● Because they can only perform simple tasks.

✗ 4. (2 points) What is the primary purpose of the `pthread_join()` function?

- ○ To create a new thread.
- ○ To terminate a thread.
- ● To pass arguments to a thread.
- ○ To wait for a thread to terminate before continuing execution.

✓ 5. (2 points) In a multi-threaded program on a multi-processor system, how do threads execute?

- ○ Always sequentially.
- ● In parallel.
- ○ Only when the main thread is sleeping.
- ○ Only one thread per processor.

Synchronization using condition variables

CS/CE-232/324 - Operating Systems Lab

Fall 2024

Instructions

- Each question carries equal marks.
- Color the circle corresponding to the correct answer.
- There is only one correct answer for each question.
- Answer all questions.

10/10

Name: Bracha Qasim

ID: 08283

✓1. (2 points) What is the primary difference between a mutex and a condition variable?

- ○ Mutexes are faster than condition variables
- ○ Condition variables can only be used with one thread at a time
- ○ Mutexes can only be used for producer-consumer problems
- ● Mutexes enforce mutual exclusion while condition variables allow threads to communicate state changes

✓2. (2 points) In the `pthread_cond_wait()` function, what happens internally in the correct order?

- ○ Block thread, relock mutex, unlock mutex
- ● Unlock mutex, block thread, relock mutex
- ○ Block thread, unlock mutex, relock mutex
- ○ Relock mutex, unlock mutex, block thread

✓3. (2 points) Why must a condition variable always be used with a mutex?

- ○ To improve performance
- ○ Because it's a POSIX requirement
- ● The mutex provides mutual exclusion for accessing shared variables while condition variables signal state changes
- ○ Condition variables cannot function without mutexes due to hardware limitations

✓4. (2 points) When implementing a producer-consumer solution with a buffer of size N, when should the producer thread wait?

- ● When the buffer is full
- ○ When the buffer is empty
- ○ When the consumer is reading
- ○ When the mutex is locked

✓5. (2 points) What operation is used to inform waiting threads that a condition has changed?

- ○ `pthread_cond_wait()`
- ○ `pthread_create()`
- ○ `pthread_mutex_lock()`
- ● `pthread_cond_signal()`

12

POSIX Thread Synchronization

CS/CE-232/324 - Operating Systems Lab

Fall 2024

10/10

Instructions

- Each question carries equal marks.
- Color the circle corresponding to the correct answer.
- There is only one correct answer for each question.
- Answer all questions.

Name: Ashbah Faisal

ID: 08271

```c
/*race.c*/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define INC_SIZE 10000000  // 10 million

static volatile int glob = 0;

void* access_global(void* args) {
    char* p   = (char*)args;
    int   loc = 0;

    printf("%s:thread started\n", p);

    for (int i = 0; i < INC_SIZE; ++i) {
        loc = glob;
        loc++;
        glob = loc;
    }

    printf("%s:threadending\n", p);
}

int main(int argc, char* argv[]) {
    pthread_t t1, t2;  // thread identifier

    // create a new thread that runs hello_arg with argument hello
    printf("main: creating threads\n");
    pthread_create(&t1, NULL, access_global, "T0");
    pthread_create(&t2, NULL, access_global, "T1");
    printf("main: created threads\n");

    // wait until the thread completes
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("main: joined threads\n");
    printf("main: glob = %d\n", glob);

    return 0;
}
```

Listing 1: Accessing global data (race.c)

1. (2 points) What is the primary cause of race conditions in multithreaded programs with shared variables?
   - ○ Excessive memory usage by threads
   - ○ Operating system scheduler bugs
   - ○ Network latency between threads
   - ◉ Concurrent modification of shared resources without synchronization

2. (2 points) In Listing 1 provide, why does increasing `INC_SIZE` to 100 million make the issue more apparent?
   - ○ It causes the program to crash
   - ○ It makes the program run faster
   - ◉ It gives more opportunities for thread interruption and scheduling conflicts
   - ○ It reduces the chance of race conditions

3. (2 points) What happens when a thread calls `pthread_mutex_lock()` on a mutex that is already locked by another thread?
   - ◉ It blocks until the mutex is unlocked
   - ○ It returns an error code
   - ○ It crashes the program
   - ○ It automatically unlocks the mutex

4. (2 points) Which scenario could lead to a deadlock when using mutexes?
   - ○ A thread attempting to unlock a mutex it doesn't own
   - ○ A thread calling `pthread_mutex_lock()` on an already unlocked mutex
   - ○ Multiple threads calling `pthread_mutex_unlock()` simultaneously
   - ◉ A thread calling `pthread_mutex_lock()` on a mutex it already holds

5. (2 points) What is the correct sequence for accessing a shared resource using mutexes?
   - ○ Create mutex → Lock mutex → Use resource → Destroy mutex
   - ◉ Lock mutex → Use resource → Unlock mutex
   - ○ Lock mutex → Use resource → Unlock mutex → Lock mutex again
   - ○ Create thread → Lock mutex → Use resource → Unlock mutex

Using Signals to communicate between processes

CS/CE-232/324 - Operating Systems Lab

Fall 2024

4/10

Instructions

- Each question carries equal marks.
- Color the circle corresponding to the correct answer.
- There is only one correct answer for each question.
- Answer all questions.

Name: Bresha Qasim

ID: 08283

✓ 1. (2 points) What is the primary purpose of signals in Unix-like operating systems?

- ○ To provide a user-friendly interface.
- ● To enable communication between two running processes.
- ○ To optimize memory allocation.
- ○ To establish network connections.

✗ 2. (2 points) What happens when a process receives a signal by default?

- ○ The process ignores the signal.
- ○ The process catches the signal and executes a specific handler.
- ○ The process performs the default action associated with the signal.
- ● The operating system terminates the process.

✓ 3. (2 points) Which of the following signals cannot be ignored by a process?

- ○ `SIGINT` and `SIGKILL`
- ● `SIGKILL` and `SIGSTOP`
- ○ `SIGINT` and `SIGSTOP`
- ○ `SIGTERM` and `SIGKILL`

✗ 4. (2 points) What does the `signal()` function do when the action is set to `SIG_IGN`?

- ○ Executes the default action for the signal.
- ○ Registers a custom signal handler for the signal.
- ○ Ignores the signal completely.
- ● Terminates the process.

✗ 5. (2 points) What should the signature of a valid signal handler function be?

- ○ `int fn(int signal)`
- ○ `void fn(int signal)`
- ● `void fn()`
- ○ `int fn()`