Questions

1. What is one of the primary challenges of converting single-threaded code to multi-threaded code?
   - ☐ A) Reducing code size
   - ☐ B) Ensuring compatibility with existing programs
   - ☐ C) Simplifying function calls
   - ☐ D) Increasing execution speed
2. In the context of multi-threading, what is a global variable?
   - ☐ A) A variable that is accessible only within a single thread
   - ☐ B) A variable that can be modified by any thread but may cause conflicts
   - ☐ C) A variable that is never modified
   - ☐ D) A variable that is local to the main function
3. What issue arises with the shared `errno` variable in a multi-threaded environment?
   - ☐ A) It cannot be accessed by multiple threads
   - ☐ B) It can hold incorrect error codes when threads switch
   - ☐ C) It is always set to zero
   - ☐ D) It is automatically reset by the kernel
4. What is one proposed solution to avoid conflicts with global variables in multi-threaded code?
   - ☐ A) Use only local variables
   - ☐ B) Prohibit the use of global variables entirely
   - ☐ C) Give each thread its own global variable
   - ☐ D) Store all variables in a database
5. Which of the following is an example of a custom library procedure to manage thread-local global variables?
   - ☐ A) Allocate_global()
   - ☐ B) Create_global("bufptr")
   - ☐ C) Local_global("bufptr")
   - ☐ D) Thread_global("bufptr")
6. What is the purpose of the `Set_global("bufptr", &buf)` call?
   - ☐ A) It deletes the global variable
   - ☐ B) It stores a pointer's value in a thread-local global variable
   - ☐ C) It creates a new thread
   - ☐ D) It reads the value of a global variable
7. Why can many library procedures be problematic when transitioning to multi-threaded environments?
   - ☐ A) They require too much memory
   - ☐ B) They are not reentrant and can lead to crashes
   - ☐ C) They execute too slowly
   - ☐ D) They are difficult to understand
8. How can non-reentrant library procedures be handled in multi-threading?
   - ☐ A) By eliminating them from the codebase
   - ☐ B) By adding a wrapper code to manage access

☐ C) By converting them to reentrant procedures
☐ D) By using global variables only

9. What problem can arise when multiple threads call `alarm` simultaneously?
   ☐ A) The program will crash
   ☐ B) Only the first thread's alarm will trigger
   ☐ C) The kernel does not recognize individual threads for alarms
   ☐ D) All threads will receive an alarm signal
10. In a multi-threaded environment, how does the kernel handle stack overflow detection?
   ☐ A) It increases the size of the stack for each thread
   ☐ B) It does not recognize stack overflows per thread
   ☐ C) It automatically terminates the process
   ☐ D) It ignores stack size entirely
11. What is a key consideration when using global variables in multi-threaded applications?
   ☐ A) They should be used frequently
   ☐ B) They must always be static
   ☐ C) They should have restricted access to prevent conflicts
   ☐ D) They are only needed in the main thread
12. What does the term "thread-local global variables" refer to?
   ☐ A) Global variables that can be accessed by all threads
   ☐ B) Variables that are unique to each thread
   ☐ C) Variables that are stored in user space
   ☐ D) Shared variables between processes
13. What is the consequence of a thread switching during a non-reentrant library call?
   ☐ A) The system will run more efficiently
   ☐ B) It can lead to the use of invalid pointers
   ☐ C) The thread will be terminated
   ☐ D) The library call will execute successfully
14. Why is it complex to transform an existing system into a multi-threaded one?
   ☐ A) It requires hardware upgrades
   ☐ B) It demands a complete redesign of the system
   ☐ C) It cannot be done without rewriting all code
   ☐ D) It leads to performance loss
15. What is the conclusion regarding the challenges of converting single-threaded to multi-threaded systems?
   ☐ A) It is always a straightforward process
   ☐ B) It raises many compatibility issues but is manageable
   ☐ C) It requires extensive documentation
   ☐ D) It can be avoided entirely

Answers

1. B) Ensuring compatibility with existing programs
2. B) A variable that can be modified by any thread but may cause conflicts
3. B) It can hold incorrect error codes when threads switch
4. C) Give each thread its own global variable
5. B) Create_global("bufptr")
6. B) It stores a pointer's value in a thread-local global variable
7. B) They are not reentrant and can lead to crashes
8. B) By adding a wrapper code to manage access
9. C) The kernel does not recognize individual threads for alarms
10. B) It does not recognize stack overflows per thread
11. C) They should have restricted access to prevent conflicts
12. B) Variables that are unique to each thread
13. B) It can lead to the use of invalid pointers
14. B) It demands a complete redesign of the system
15. B) It raises many compatibility issues but is manageable