Advanced Operating Systems

Pre-Lab Exercise

Submitted By:
Manish Choudhary
GTID: 902982487

Question 1: The main function contains calls to exit() (line 66) and pthread_exit() (line 80). How will the effect of these two calls differ when they are executed?

Answer 1:

Exit() function terminates the process along with all the threads while pthread_exit() kills the calling thread only. So, calling pthread_exit() from main would allow the other threads to continue but exit() will terminate the process i.e. all the threads would be killed and resources would be freed.

In the code, if we are not able to create the producer or consumer thread, we would exit terminating all the existing threads. But, if we are not able to join these threads from main, we want the other threads to continue. So, exiting using pthread_exit() from the initial thread "main" terminates only this thread and the process terminates only when the last thread in it terminates.

Unlike **exit**, **pthread_exit** does not clean up system resources shared among threads.

Question 2: The main function calls pthread_join() (line 77) with the parameter thread_return. Where does the value stored in thread_return come from when the consumer_thread is joined?

Answer 2:

*int pthread_join(pthread_t thread, void **return_value);*

The return_value is the address of a void pointer which would contain the value returned by the thread (on which join has been called by passing its thread ID as the first argument to function) on termination. The return_value can be passed to any successful join either with ***return (void *) value*** or with ***pthread_exit(void *value).***

In the code, the consumer thread returns the type casted value of count ***return (void*) count;*** which gets stored in thread_return.

Question 3: Where does the value stored in thread_return come from if the joined thread terminated by calling pthread_exit instead of finishing normally?

Answer 3:

*void pthread_exit(void *retval);*

pthread_exit() can also return the value to the thread that called join on this thread. So, even if we terminate the thread with pthread_exit(), the count value can be returned in thread_return by calling ***pthread_exit((void *)count);***

Even returning from another thread implicitly calls the ***pthread_exit*** subroutine. The return value has the same role as the *status* parameter of the ***pthread_exit*** subroutine.

Question 4: On the same call to pthread_join() (line 77), what will it do if the thread being joined (consumer_thread, in this case) finishes before the main thread reaches the that line of code (line 77)?

Answer 4:

If the target thread is already terminated, the pthread_join subroutine would return immediately. When main executes a thread join and consumer has already terminated, main continues as if no such thread join has ever executed (i.e., join has no effect).

Question 5: In this program, the main thread calls pthread_join() on the threads it created. Could a different thread call pthread_join() on those threads instead? Could a thread call pthread_join() on the main thread (assuming it knew the main thread's thread ID - i.e. pthread_t)?

Answer 5:

Yes, any thread can call pthread_join() on any other joinable thread. So, pthread_join() can be called on the threads (created by main) by any other thread. For example, producer thread can call pthread_join() on the consumer thread created by the main thread.

Yes, pthread_join() can be called on the main thread by some other thread.

Question 6: The consumer_routine function calls sched_yield() (line 180) when there are no items in the queue. Why does it call sched_yield() instead of just continuing to check the queue for an item until one arrives?

Answer 6:

Checking the queue for an item again and again would waste CPU cycles and resources. Calling sched_yield() would make the calling thread to relinquish the CPU and would put it at the end of the ready queue. This would allow the other threads/processes to perform some useful work.