7 Concurrent and Distributed Systems (RNW)

This question is with respect to an operating system that supports multi-threaded processes using the POSIX threads (pthreads) API. Assume that each call to printf prints its output atomically, that thread scheduling is non-deterministic, and that threads are allocated unique and sequential integer IDs starting with 0.

(a) Some program state is per-process, and some is per-thread. How many instances of each of the following will a 2-thread process have: virtual address space, executable program, register file, scheduling state (e.g., RUN, SLEEP), and stack? [5 marks]

(b) A programmer adds printfs to a concurrent program to debug a race condition, but the symptoms vanish. Explain why this might have happened. [2 marks]

(c) thrprint accepts as arguments the current thread’s unique ID and a debug message to print. If each thread calls thrprint exactly once on start, how many possible interleavings are there with n threads? [2 marks]

\[
\text{void thrprint(int threadid, char *message) \{ }
\text{printf("Thread \%d: \%s\n", threadid, message);} \\
\text{\}}
\]

(d) ordered_thrprint attempts to print debug messages ordered by thread ID. Describe three ways in which the synchronisation in this implementation is incorrect, and provide a corrected pseudocode implementation. [6 marks]

\[
\text{int next_thread_id = 0; // Next ID to print } \\
\text{pthread_mutex_t ordering_mtx; // Lock protecting next ID } \\
\text{pthread_cond_t ordering_cv; // next_thread_id has changed} \\
\text{void ordered_thrprint(int thread_id, char *message) \{ } \\
\text{pthread_mutex_lock(ordering_mtx);} \\
\text{if (thread_id \! = \! next_thread_id) \{ } \\
\text{pthread_cond_wait(ordering_cv, ordering_mtx);} \\
\text{\}} \\
\text{next_thread_id = next_thread_id + 1;} \\
\text{pthread_mutex_unlock(ordering_mtx);} \\
\text{printf("Thread \%d: \%s\n", thread_id, message);} \\
\text{\}}
\]

(e) This approach to implementing ordered_thrprint suffers a substantial performance problem: if lower-numbered threads are slow in starting, then higher-numbered threads will also be delayed. Describe an alternative strategy, paying particular attention to synchronisation, that maintains ordered output while allowing greater concurrency. [5 marks]