CSC112 Lab1: Multi-Threaded Programming

You are given a C program “pthread vows cons.c” that creates two threads (`vow` and `cons`) that print words starting with vowels and consonants, respectively, while maintaining the original order of input words. It uses the syscall `sched\_yield()` to allow threads to take turns.

Compile the program using command line (or VS Studio Code IDE):

gcc -pthread -o threadtest threadtest.c

Run the program by supplying a list of words as arguments, for example:

./threadtest apple banana orange grape kiwi umbrella

The output looks like this:

Vowel: apple

Consonant: banana

Vowel: orange

Consonant: grape

Consonant: kiwi

Vowel: umbrella

The program works as follows:

1. Input Handling: The program takes command-line arguments as input. Each argument is treated as a word.

2. Thread Creation: The `vow` thread prints words starting with vowels. The `cons` thread prints words starting with consonants.

3. Turn-Based Synchronization:

- A shared variable `turn` determines which thread should process the current word:

- `turn == 0`: Vowel thread's turn.

- `turn == 1`: Consonant thread's turn.

- Each thread checks whether it's its turn and processes the word accordingly. If it's not its turn, it calls sched\_yield()to yield control to the other thread.

4. Word Processing:

- Each thread checks whether the current word starts with a vowel or consonant and prints it accordingly.

- The `current\_index` ensures that both threads process words in sequence.

The system call sched\_yield()lets the calling thread voluntarily give up control to the OS scheduler, in order to give other active threads a chance to run, The two threads use sched\_yield()to take turns in printing out the vows and cons. The program preserves the original word sequence in the argument list because both threads operate on a shared index `current\_index` and take turns based on the `turn` variable. No mutexes or explicit locking mechanisms are used.

# Task 1: Use the mutex API to print words in original order

Your first task is to modify the program to use the mutex API instead of sched\_yield() while keeping the same program behavior (of printing words starting with vowels and consonants, respectively while maintaining the original order of input words), i.e.,

pthread\_mutex\_t lock;

pthread\_mutex\_lock(&lock);

pthread\_mutex\_unlock(&lock);

Hints:

1. You still need the shared variable `current\_index`, but not the shared variable `turn`.
2. You need to call pthread\_mutex\_lock(&lock)and pthread\_mutex\_unlock(&lock)within functions print\_vowels() and print\_consonants()to ensure mutual exclusion between them, since they are not yielding to each other voluntarily.

# Task 2: Use the mutex API to print all vowels before all consonants

Your second task is to modify the program to still use the mutex API, but change the program behavior to print out all vowels before all consonants. First, you need to modify the functions print\_vowels()and print\_consonants()to each iterate through all the input arguments and print out all vowels and consonants, respectively, using mutex to ensure the atomic execution of each function. Second, you need to make sure that the thread that runs print\_vowels() runs and finishes before the thread that runs print\_consonants(). If you are running Linux, this can be achieved by using the fixed-priority real-time scheduling policy defined in the POSIX standard. SCHED\_FIFO, and assigning higher priority to the pthread that runs print\_vowels()than the pthread that runs print\_consonants().However, if you are running Windows, SCHED\_FIFO does not work because the Windows implementation of POSIX threads (such as Pthreads-w32) does not support real-time scheduling policies like SCHED\_FIFO. Instead, it only supports the SCHED\_OTHER policy, which corresponds to the default, non-real-time scheduling behavior in Windows. So a simple hack is to call the Windows function Sleep(100) after starting the thread that runs print\_vowels(), and before starting the thread that runs print\_consonants().

# What to submit

Please submit the following on Canvas:

1. A C program that uses the mutex API, and prints words starting with vowels and consonants, respectively while maintaining the original order of input words , and screenshots for running it against some inputs.
2. Another C program that uses the mutex API, and prints out all vowels before all consonants, using either the SCHED\_FIFO policy on Linux, or the Sleep() function on Windows.
3. A short PDF report explaining the code you have written and the execution results.