CSC112 Lab1: Printing Vowels and Consonants

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.

Note: Since Windows has limited support for the POSIX API, esp. not the process fork() call, please work on a Linux machine, or install the Windows Subsystem for Linux (WSL): <https://learn.microsoft.com/en-us/windows/wsl/install>. If you use VS Code, then please install VS Code WSL extension <https://code.visualstudio.com/docs/remote/wsl>.

# Task 1: Use pthreads and 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 pthreads to print all vowels before all consonants

Your second task is to modify the program to use pthreads to achieve the program behavior of printing 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. Second, you need to make sure that the thread that runs print\_vowels() runs and finishes before the thread that runs print\_consonants(). This can be done by calling pthread\_create() to create the first thread that runs print\_vowels(), and pthread\_join() to wait for it to finish, before calling pthread\_create() to create the second thread that runs print\_consonants(), and pthread\_join() to wait for it to finish. (You do not need mutex protection for each function, since the two threads run sequentially, not concurrently.) Please refer to the following code snippet:

#include <pthread.h>

...

pthread\_create(&vow\_thread, NULL, print\_vowels, NULL);

pthread\_join(vow\_thread, NULL);

pthread\_create(&cons\_thread, NULL, print\_consonants, NULL);

pthread\_join(cons\_thread, NULL);

Try the following, does it work?

pthread\_create(&vow\_thread, NULL, print\_vowels, NULL);

pthread\_create(&cons\_thread, NULL, print\_consonants, NULL);

pthread\_join(vow\_thread, NULL);

pthread\_join(cons\_thread, NULL);

References:

Thread functions in C/C++

<https://www.geeksforgeeks.org/thread-functions-in-c-c/>

How to create and join threads in C (pthreads).

<https://www.youtube.com/watch?v=uA8X5zNOGw8>

# Task 3: Use child processes to print all vowels before all consonants (only works on Linux)

Your third task is to modify the program to use process fork() and join() to achieve the program behavior of printing out all vowels before all consonants. This can be done by calling fork() to create the first child process that runs print\_vowels(), and waitpid() to wait for it to finish, before calling fork() to create the second child process that runs print\_consonants(), and waitpid() to wait for it to finish. (You do not need mutex protection for each function, since the two processes do not share memory, and they run sequentially.) Please refer to the following code snippet:

#include <sys/wait.h> //This does not work on Windows

...

pid\_t vowel\_pid = fork();

waitpid(vowel\_pid, NULL, 0);

pid\_t cons\_pid = fork();

waitpid(cons\_pid, NULL, 0);

Try the following, does it work?

pid\_t vowel\_pid = fork();

pid\_t cons\_pid = fork();

waitpid(vowel\_pid, NULL, 0);

waitpid(cons\_pid, NULL, 0);

References:

Slide 13 “wait()” in “Lecture 2 Processes and Threads”

fork() in C

<https://www.geeksforgeeks.org/fork-system-call/>

# What to submit

Please submit the following on Canvas:

1. Task 1: A C program named Task1.c 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. Task 2: A C program named Task2.c that uses pthreads to print all vowels before all consonants.
3. Task 3: A C program named Task3.c that uses child processes to print all vowels before all consonants.
4. A short PDF report explaining the code you have written and the execution results.