Vector Parallelization (parallel_for):

The parallel_for function divides a range [low, high) into smaller chunks based on the number of threads specified.

It creates an array of pthreads and a corresponding array of thread_args_vector structures to pass parameters to each thread.

The lambda function is passed to each thread, and each thread processes its assigned chunk of the range.

Error handling for thread creation and joining is included.

The total execution time is measured using the clock function.

Matrix Parallelization (parallel_for for matrices):

Similar to the vector version, but it operates on a 2D range specified by low1, high1, low2, and high2.

It creates an array of pthreads and a corresponding array of thread_args_matrix structures to pass parameters to each thread.

The lambda function takes two indices and processes the corresponding element in the 2D range.

Error handling for thread creation and joining is included.

The total execution time is measured using the clock function.

Lambda Demonstration (demonstration):

This function takes a lambda function and executes it.

Two lambda functions are demonstrated in the main function, showing how to capture variables and execute them.

Main Function (main):

Sets up a couple of lambda functions (lambda1 and lambda2) to demonstrate capturing and execution.

Calls the user_main function and captures its return code.

Executes the second lambda function.

Returns the return code obtained from user_main.

- Contribution Percentage:

50% Details:

➔ Made a basic structure of the code, laying down the logic and components.

➔ Focused on the code implementation, from defining global variables to drifting the initial versions of the primary functions.

2. Arav: ● Contribution Percentage: 50% Details:

➔ Worked intensively on error analysis.

➔ Dedicated efforts towards the optimization of the code, making it efficient, and more user friendly.

Final Breakdown:

- Anish: 50%
- Arav: 50%

GITHUB : https://github.com/ianishdev/OS-Assignment.git