**Thread concept: clone, threads of java**

**Subject - Unix Operating System**

**Name – Hemant Sharma**

**PRN – 22610001 Class – TYIT**

**Assignment No – 4(e)**

**Title-** Write a program using p-thread library of Linux. Create three threads to take odd, even and prime respectively and print their average respectively.

**Objectives:**

1. To learn about threading in Linux/Unix and Java and difference between them.
2. Use of system call/library to write effective programs.

**Theory:**

* **POSIX Threads (pthreads)**: POSIX threads (also called pthread) is a standard for managing multiple threads in Unix-like operating systems. It provides a set of functions to create, synchronize, and manage threads. In this program, we use pthread\_create to start a new thread and pthread\_join to wait for the thread to complete.
* **Thread Synchronization**: The pthread\_join function ensures that the main thread waits for the worker threads (odd, even, and prime number threads) to complete their execution before the program exits.
* **Thread Efficiency**: By using threads to calculate the averages concurrently, we can achieve better performance compared to a single-threaded approach, particularly for larger ranges or more complex operations.

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <stdbool.h>

#define N 10  // Number of elements to consider

// Shared data structure for storing results

typedef struct {

    int sum;

    int count;

} ThreadData;

ThreadData oddData = {0, 0};

ThreadData evenData = {0, 0};

ThreadData primeData = {0, 0};

// Function to check if a number is prime

bool is\_prime(int num) {

    if (num < 2) return false;

    for (int i = 2; i \* i <= num; i++) {

        if (num % i == 0) return false;

    }

    return true;

}

// Thread function to calculate odd number average

void\* odd\_numbers(void\* arg) {

    for (int i = 1; i <= N; i += 2) {

        oddData.sum += i;

        oddData.count++;

    }

    printf("Odd Average: %.2f\n", (float)oddData.sum / oddData.count);

    return NULL;

}

// Thread function to calculate even number average

void\* even\_numbers(void\* arg) {

    for (int i = 2; i <= N; i += 2) {

        evenData.sum += i;

        evenData.count++;

    }

    printf("Even Average: %.2f\n", (float)evenData.sum / evenData.count);

    return NULL;

}

// Thread function to calculate prime number average

void\* prime\_numbers(void\* arg) {

    for (int i = 2; i <= N; i++) {

        if (is\_prime(i)) {

            primeData.sum += i;

            primeData.count++;

        }

    }

    printf("Prime Average: %.2f\n", (float)primeData.sum / primeData.count);

    return NULL;

}

int main() {

    pthread\_t thread1, thread2, thread3;

    // Create threads

    pthread\_create(&thread1, NULL, odd\_numbers, NULL);

    pthread\_create(&thread2, NULL, even\_numbers, NULL);

    pthread\_create(&thread3, NULL, prime\_numbers, NULL);

    // Wait for threads to finish

    pthread\_join(thread1, NULL);

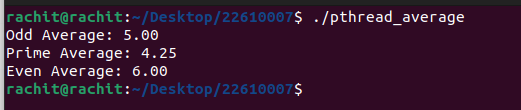
    pthread\_join(thread2, NULL);

    pthread\_join(thread3, NULL);

    return 0;

}

**Output:**

****

**Conclusion:**

This program demonstrates the use of POSIX threads (pthreads) to concurrently calculate the averages of odd, even, and prime numbers in a given range. By utilizing threads, we can achieve parallelism and improve performance for larger tasks. In this case, while the range is small, the program shows how threading can be applied to perform multiple computations simultaneously, enhancing efficiency and organization of code.