

**Faculty of Engineering and Technology**

**Electrical and Computer Engineering Department**

**Operating system**

**Project Report**

**Prepared by:**

Ghassan Qandeel 1212397

Leelyan Karajah 1201191

**Instructor :** Dr. Bashar Tahayna

**Section:** 1

**Date**: september 5,2023

**Code Overview:**

This report explains a C program designed to simulate thread-level scheduling using the round-robin algorithm on a computer with multiple processing cores. The purpose is to help us understand how tasks (threads) are scheduled and executed on these cores.

**Thread and Core:**

* Threads represent individual tasks. Each thread has an ID and keeps track of how much work (remaining time) it has left.
* Cores represent the computer's processing units. They can be either busy (working on a task) or idle (not working).

**Setting Up:**

* We define some basic parameters:
  + There are 5 "processes" (groups of tasks).
  + Each process can have up to 3 tasks (threads).
  + The computer has 3 CPU cores.
  + Each task can work for up to 4 units of time before pausing.

**Executing Tasks:**

* The program performs the following steps:
  1. It prepares the CPU cores, making them initially idle.
  2. Tasks (threads) are created and assigned to processes, with random execution times (between 1 and 10 units of time).
  3. The program simulates thread scheduling and execution:
     + Threads are scheduled onto CPU cores, following the round-robin approach.
     + Cores execute threads for a predetermined time (time quantum), simulating their work.
     + When a task (thread) finishes its work, it's marked as completed, and the core becomes idle again.
  4. This simulation continues until all tasks are completed.

**Cleaning Up:**

* After the simulation, the program frees up the memory allocated for the tasks to avoid wasting resources.

**The Code**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <stdbool.h>**

**// Thread structure**

**typedef struct {**

**int process\_id;**

**int thread\_id;**

**int remaining\_time;**

**} Thread;**

**// Core structure**

**typedef struct {**

**bool busy;**

**Thread\* running\_thread;**

**} Core;**

**int main() {**

**// Define parameters**

**const int num\_processes = 5;**

**const int max\_threads\_per\_process = 3;**

**const int num\_cores = 3;**

**const int time\_quantum = 4;**

**// Create cores**

**Core cores[num\_cores];**

**for (int i = 0; i < num\_cores; ++i) {**

**cores[i].busy = false;**

**cores[i].running\_thread = NULL;**

**}**

**// Create and initialize threads**

**Thread\* threads[num\_processes][max\_threads\_per\_process];**

**for (int i = 0; i < num\_processes; ++i) {**

**for (int j = 0; j < max\_threads\_per\_process; ++j) {**

**threads[i][j] = (Thread\*)malloc(sizeof(Thread));**

**threads[i][j]->process\_id = i;**

**threads[i][j]->thread\_id = j;**

**threads[i][j]->remaining\_time = rand() % 10 + 1; // Random execution time (1-10)**

**}**

**}**

**// Simulation loop**

**int current\_time = 0;**

**int current\_core = 0;**

**int total\_remaining\_threads = num\_processes \* max\_threads\_per\_process;**

**while (total\_remaining\_threads > 0) {**

**// Schedule threads on cores**

**for (int i = 0; i < num\_processes; ++i) {**

**for (int j = 0; j < max\_threads\_per\_process; ++j) {**

**Thread\* thread = threads[i][j];**

**if (thread->remaining\_time > 0) {**

**if (!cores[current\_core].busy) {**

**cores[current\_core].busy = true;**

**cores[current\_core].running\_thread = thread;**

**current\_core = (current\_core + 1) % num\_cores;**

**}**

**}**

**}**

**}**

**// Execute threads on cores**

**for (int i = 0; i < num\_cores; ++i) {**

**if (cores[i].busy) {**

**Thread\* running\_thread = cores[i].running\_thread;**

**printf("Time %d: Executing Process %d Thread %d\n", current\_time,**

**running\_thread->process\_id, running\_thread->thread\_id);**

**running\_thread->remaining\_time -= time\_quantum;**

**if (running\_thread->remaining\_time <= 0) {**

**printf("Time %d: Process %d Thread %d completed\n\n", current\_time,**

**running\_thread->process\_id, running\_thread->thread\_id);**

**cores[i].busy = false;**

**cores[i].running\_thread = NULL;**

**total\_remaining\_threads--;**

**}**

**}**

**}**

**current\_time += time\_quantum;**

**}**

**// Clean up allocated memory**

**for (int i = 0; i < num\_processes; ++i) {**

**for (int j = 0; j < max\_threads\_per\_process; ++j) {**

**free(threads[i][j]);**

**}**

**}**

**return 0;**

**}**

# Conclusion

In conclusion, This C program is like a basic simulation of how a computer manages multiple tasks (threads) running on different CPU cores using the round-robin scheduling approach. It helps us understand the fundamental concept of scheduling tasks on a computer with multiple processing units.