Write a C program to simulate Real-Time CPU Scheduling algorithms: Rate Monotonic

```
#include <stdio.h>

struct Task {
    int id;
    int execution_time;
    int period;
};

void rate_monotonic(struct Task tasks[], int n, int hyper_period) {
    printf("Timeline (Rate-Monotonic Scheduling):\n");

    for (int time = 0; time < hyper_period; time++) {
        int task_executed = -1;

        // Task arrivals
        for (int i = 0; i < n; i++) {
            if (time % tasks[i].period == 0) {
                printf("Task %d arrives at time %d\n", tasks[i].id, time);
            }
        }

        // Select task with shortest period that still needs to execute
        for (int i = 0; i < n; i++) {
            if (time % tasks[i].period < tasks[i].execution_time) {
                task_executed = tasks[i].id;
                printf("At time %d: Executing Task %d\n", time, task_executed);
                break;
            }
```

```
        }

        // If no task was executed
        if (task_executed == -1) {
            printf("At time %d: CPU Idle\n", time);
        }
    }
}

int main() {
    struct Task tasks[] = {
        {1, 1, 3},
        {2, 2, 5}
    };

    int n = sizeof(tasks) / sizeof(tasks[0]);
    int hyper_period = 15;

    rate_monotonic(tasks, n, hyper_period);
    return 0;
}
```

OUTPUT:

```
Timeline (Rate-Monotonic Scheduling):
Task 1 arrives at time 0
Task 2 arrives at time 0
At time 0: Executing Task 1
At time 1: Executing Task 2
At time 2: CPU Idle
Task 1 arrives at time 3
At time 3: Executing Task 1
At time 4: CPU Idle
Task 2 arrives at time 5
At time 5: Executing Task 2
Task 1 arrives at time 6
At time 6: Executing Task 1
At time 7: CPU Idle
At time 8: CPU Idle
Task 1 arrives at time 9
At time 9: Executing Task 1
Task 2 arrives at time 10
At time 10: Executing Task 2
At time 11: Executing Task 2
Task 1 arrives at time 12
At time 12: Executing Task 1
At time 13: CPU Idle
At time 14: CPU Idle
```