# HaclOSsim FreeRTOS on QEMU-ARM simulator

ALBERTO AMEGLIO, GIOVANNI LUCA DI BELLA, ENRICO DI STASIO, GIUSEPPE SALVEMINI, COSIMO VERGARI

# FreeRTOS a real-time operating system

## Main characteristics include of FreeRTOS:

- Lightweight design and it is open-source nature
- Portability: Adapts seamlessly to diverse hardware architectures.
- Task Scheduling: FreeRTOS employs a priority-based queue to prioritize critical tasks, preempting less urgent ones
- Memory Management: Employs dynamic allocation and memory pools.
- Interrupt Management

## What is QEMU and why would we want to simulate with it?

- Reduced development costs as physical hardware is not required.
- Increased testing flexibility, allowing the execution of multiple scenarios.
- Early debugging and problem identification.
- Safe experimentation without the risks associated with hardware manipulation.

# A quick Tutorial:

1. First we need to download QEMU and the ARM architecture extension
2. We need to download FreeRTOS
3. We need a debugger for ARM, in this case we have selected the ARM GNU toolchain
4. We need a compiler make and cmake

qemu-system-arm -machine mps2-an385 -cpu cortex-m3 -kernel build/gcc/output/RTOSDemo.out -serial stdio -s –S

# FIRST COME FIRST SERVED

**Non-Preemptive**

- A task can't be interrupted until it finishes

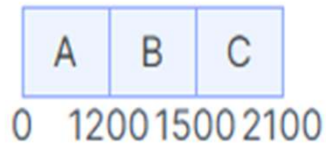
**Same Priorities**

- Tasks have the same priority

**Easy to Implement**

- It's one of the easiest algorithms to implement as it only requires a FIFO queue to manage the processes

## Gantt Chart




| Job | Arrival Time | Burst Time | Finish Time | Turnaround Time | Waiting Time |
|---|---|---|---|---|---|
| A | 0 | 1200 | 1200 | 1200 | 0 |
| B | 0 | 300 | 1500 | 1500 | 1200 |
| C | 0 | 600 | 2100 | 2100 | 1500 |
| | | | Average | 4800 / 3 = 1600 | 2700 / 3 = 900 |

# SHORTEST JOB FIRST

**Non-Preemptive**

- A task can't be interrupted until it finishes

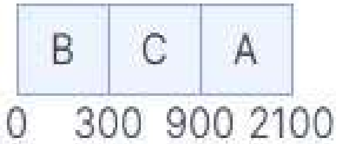
**Minimizes waiting time**

- Tends to minimize the average waiting time of the processes in the ready queue by executing first the tasks with the shortest burst time

**Challenging**

- Accurately predicting the burst time of each process is not trivial

## Gantt Chart




| Job | Arrival Time | Burst Time | Finish Time | Turnaround Time | Waiting Time |
|---|---|---|---|---|---|
| A | 0 | 1200 | 2100 | 2100 | 900 |
| B | 0 | 300 | 300 | 300 | 0 |
| C | 0 | 600 | 900 | 900 | 300 |
| | | | Average | 3300 / 3 = 1100 | 1200 / 3 = 400 |

# ROUND ROBIN

**Preemptive**

- A task can be interrupted and put in ready state

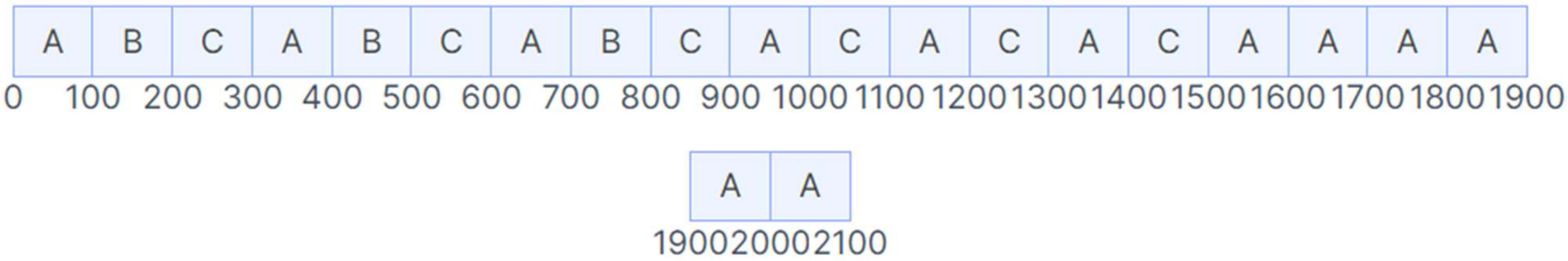
**Time Sharing**

- Each task can run for a time quantum, then it must be preempteed

**Fairness**

- Equal allocation of CPU time to all tasks

## Gantt Chart




| Job | Arrival Time | Burst Time | Finish Time | Turnaround Time | Waiting Time |
|---|---|---|---|---|---|
| A | 0 | 1200 | 2100 | 2100 | 900 |
| B | 0 | 300 | 800 | 800 | 500 |
| C | 0 | 600 | 1500 | 1500 | 900 |
| | | | Average | 4400 / 3 = 1466.667 | 2300 / 3 = 766.667 |

## STATISTICS: FCFS, SJF, RR

| | FCFS | SJF | ROUND ROBIN |
|---|---|---|---|
| AVERAGE WAITING TIME | 900 ms | 400ms | 766 ms |
| AVERAGE TURNAROUND TIME | 1600 ms | 1100 ms | 1466 ms |
| AVERAGE RESPONSE TIME | 900 ms | 400 ms | 100 ms |

# TIMELINE SCHEDULING

### Non-Preemptive

- A task can’t be interrupted until it finishes

### Major and Minor cycles

- The Execution time is divided in minor cycles. Each of them executes different tasks sequentially.

### Enhanced Resource Utilization

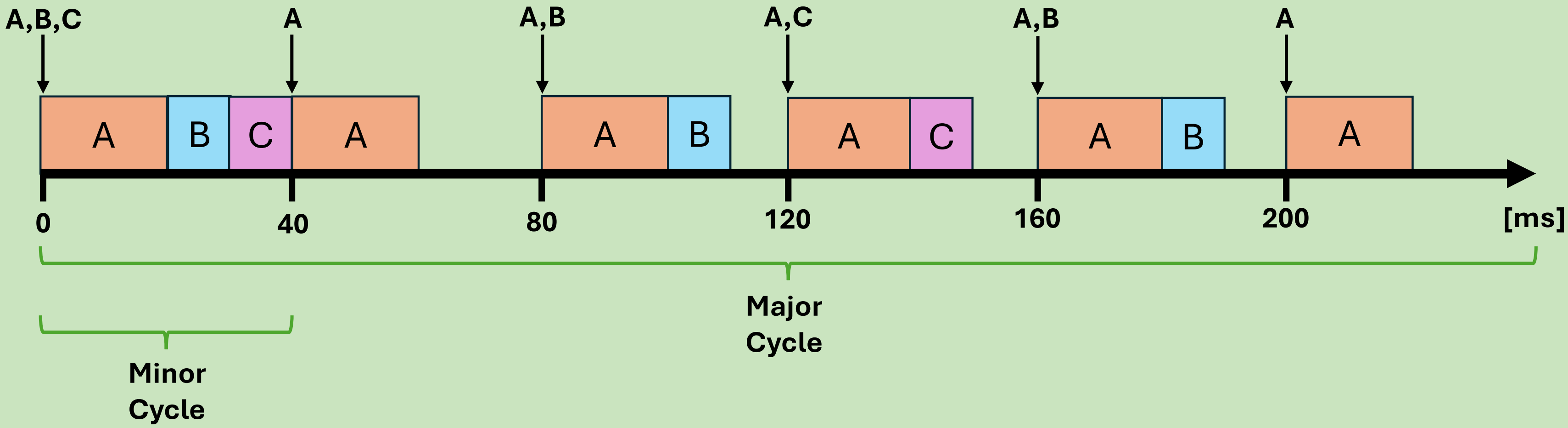
- By statically allocating time slots for task execution it can optimize resource utilization.

# Gantt Chart

| Task | T [ms] | WCET [ms] |
|---|---|---|
| A | 40 | 20 |
| B | 80 | 10 |
| C | 120 | 10 |

Cpu utilization factor

$$U_{cpu} = \frac{\sum C_i}{Total\_time} = \frac{6 * 20 + 3 * 10 + 2 * 10}{240} = \frac{170}{240} \approx 0{,}708$$



# RATE MONOTONIC

## Preemptive

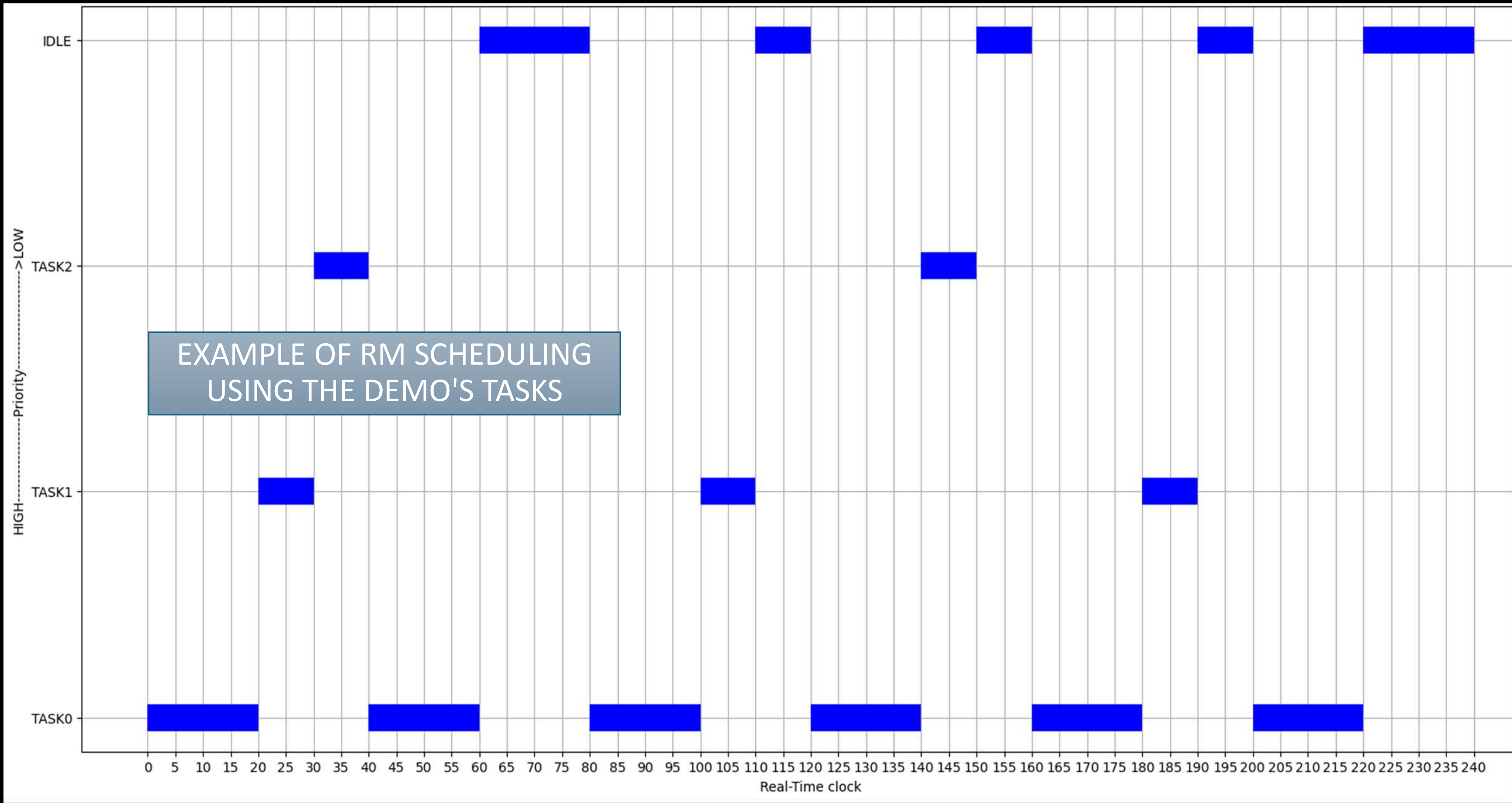
- A task can be interrupted and put in ready state

## Fixed Priorities

- Tasks'priorities are calculated as the inverse of their periods. Shorter period means higher priority

## Feasibilty

- Feasible for a given set of tasks if every instance of any task can be completed before it reaches its deadline (the time at which a new instance of itself is created)


EXAMPLE OF RM SCHEDULING
USING THE DEMO'S TASKS
IDLE
TASK2
TASK1
TASK0
HIGH-----------Priority----------->LOW
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240
Real-Time clock

# STATISTICS: TIMELINE SCHEDULING, RM

| | TIMELINE SCHEDULING | RATE MONOTONIC |
|---|---|---|
| CPU UTILIZATION FACTOR | 0.71 | 0.36 |