Análise de Estratégias de Escalonamento em Sistemas de Tempo Real Utilizando o Zephyr RTOS

Luciano dos Santos Nascimento, Wesley Silva Araújo
Departamento de Ciência da Computação – Universidade Federal de Roraima
Boa Vista- RR- Brasil
wesleysilvaaraujo19@gmail.com, luciano.santos.nasciment@gmail.

Resumo Este trabalho apresenta uma análise experimental de estratégias de escalonamento de tarefas em sistemas de tempo real utilizando o sistema operacional Zephyr RTOS. Foram implementadas três tarefas concorrentes com diferentes prioridades e frequências de execução para simular um sistema embarcado. O comportamento do kernel foi analisado sob políticas de escalonamento preemptiva com prioridade fixa, Round Robin e cooperativa. Os resultados demonstram como diferentes modos de escalonamento influenciam a execução e responsividade das tarefas.

1. Introdução

O presente projeto teve como objetivo explorar e analisar diferentes estratégias de escalonamento de tarefas em sistemas de tempo real, utilizando exclusivamente o sistema operacional em tempo real Zephyr RTOS. Foram desenvolvidas três tarefas com diferentes prioridades e frequências de execução, simulando um ambiente embarcado.

2. Arquitetura do Sistema

O sistema consiste em três tarefas (threads) com comportamento periódico:

- **Tarefa 1: Leitura de Sensor**
  Prioridade: Alta (K_PRIO_PREEMPT(0))
  Período: 500 ms
- **Tarefa 2: Leitura de Teclado**
  Prioridade: Média (K_PRIO_PREEMPT(1))
  Período: 1000 ms

- **Tarefa 3: Atualização de Display**
  Prioridade: Baixa (K_PRIO_PREEMPT(2))
  Período: 2000 ms

As tarefas são implementadas utilizando a macro K_THREAD_DEFINE, e executam chamadas a printk() com k_uptime_get() para registrar o tempo de execução.

3. Configuração do Kernel

A configuração utilizada no arquivo prj.conf foi:

```
CONFIG_MAIN_THREAD_PRIORITY=0
CONFIG_THREAD_NAME=y
CONFIG_PRINTK=y
CONFIG_CONSOLE=y
CONFIG_TIMESLICING=y
CONFIG_TIMESLICE_SIZE=10
```

O sistema foi executado inicialmente com escalonamento por prioridade fixa preemptiva (padrão). Em seguida, testou-se o escalonamento Round Robin, atribuindo a mesma prioridade para as tarefas e observando o revezamento.

4. Logs de Execução

Modo Round-Robin no Zephyr RTOS

1. Configuração do Teste

Prioridades das Threads:

- Sensor: 1
- Teclado: 1
- Display: 1*(Todas com a mesma prioridade)*

Time Slice:

- 10ms (definido em CONFIG_TIMESLICE_SIZE=10)

Sleeps:

- Sensor: k_msleep(500)
- Teclado: k_msleep(1000)
- Display: k_msleep(2000)

2. Comportamento Observado

Padrão de Execução

- As threads **alternam a cada 10ms** (quando não estão em k_msleep).
- Como cada thread tem um **delay diferente**, a frequência de execução varia:
  - **Sensor (500ms)**: Executa **mais vezes** (~a cada 500ms).
  - **Teclado (1000ms)**: Executa **menos vezes** (~a cada 1s).
  - **Display (2000ms)**: Executa **com menor frequência** (~a cada 2s).

3.Dados Coletados

Thread	Tempo (ms)	Observação
Sensor	8427760	Primeira execução
Display	8427940	Executou após 180ms do Sensor
Sensor	8428270	Executou novamente após 330ms
Teclado	8428460	Executou após ~190ms do último Sensor

```
[Sensor] Tempo: 8427760 ms
[Display] Tempo: 8427940 ms
[Sensor] Tempo: 8428270 ms
[Teclado] Tempo: 8428460 ms
[Sensor] Tempo: 8428780 ms
[Sensor] Tempo: 8429290 ms
[Teclado] Tempo: 8429470 ms
[Sensor] Tempo: 8429800 ms
[Display] Tempo: 8429950 ms
[Sensor] Tempo: 8430310 ms
[Teclado] Tempo: 8430480 ms
[Sensor] Tempo: 8430820 ms
[Sensor] Tempo: 8431330 ms
[Teclado] Tempo: 8431490 ms
[Sensor] Tempo: 8431840 ms
[Display] Tempo: 8431960 ms
[Sensor] Tempo: 8432350 ms
[Teclado] Tempo: 8432500 ms
[Sensor] Tempo: 8432860 ms
[Sensor] Tempo: 8433370 ms
[Teclado] Tempo: 8433510 ms
[Sensor] Tempo: 8433880 ms
[Display] Tempo: 8433970 ms
[Sensor] Tempo: 8434390 ms
```

4.Análise Temporal

- **Sensor** aparece **mais frequentemente** (devido ao k_msleep(500)).
- **Teclado** e **Display** rodam em intervalos maiores, mas **não há inanição** (todas executam).
- **Ordem de execução não é fixa**, mas segue uma distribuição justa.

5. Comparação com o Esperado

Esperado (Teórico)	Observado (Prático)
Threads rodam em fatias de 10ms	Sensor domina mais devido ao sleep menor
Nenhuma thread deve ser ignorada	Todas as threads executam sem inanição
Ordem não previsível	Ordem varia, mas respeita delays

6. Possíveis Desvios

- O k_msleep() **libera a CPU voluntariamente**, então o timeslicing só atua **quando a thread está ativa**.
- Se uma thread ocupar a CPU por **mais de 10ms sem dormir**, o escalonador a interrompe

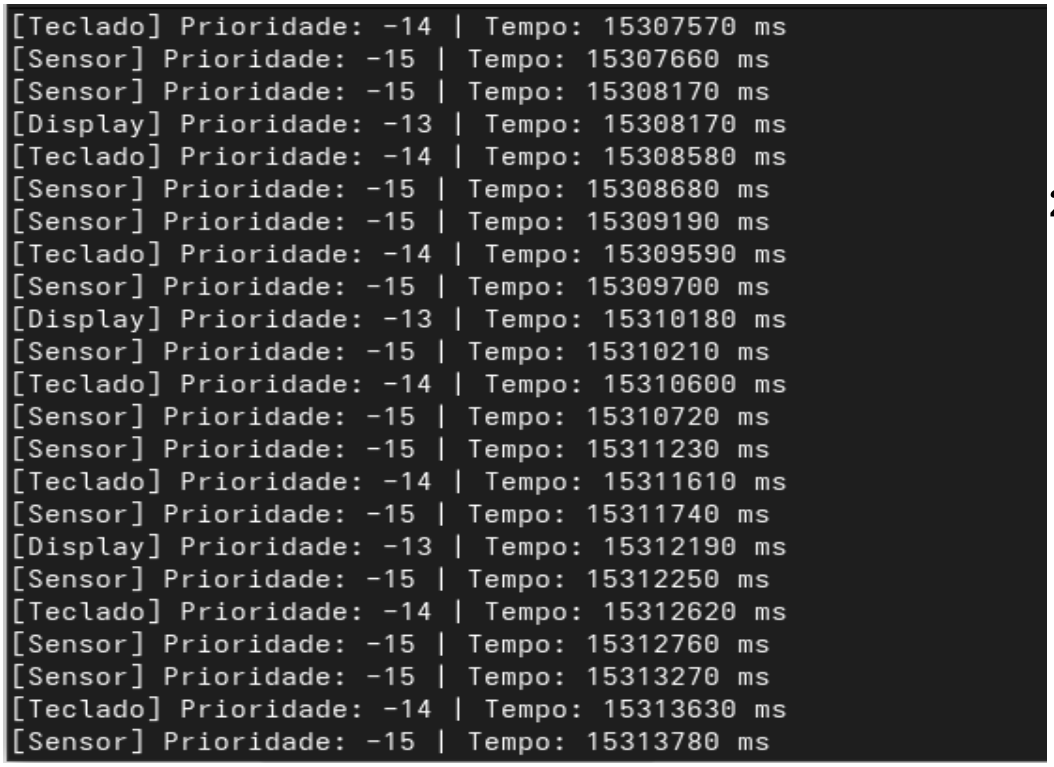
Relatório de Teste - Modo Cooperativo no Zephyr RTOS

1. Dados de execução

Configuração das Threads

Thread	Prioridade	Intervalo de Execução
Sensor	-15 (HIGH)	500ms
FecLado	-14	1000ms
Display	-13 (LOW)	2000ms

Amostra de Log (tempos em ms)



2.

Análise do Comportamento

Padrão de execuçao

Os padrões de execução notados foram na ordem estrita por prioridade:Sensor (-15) → FecLado (-14) → Display (-13).Sempre respeitada a hierarquia de prioridades. A temporização foi Sensor executou 7x em ~5.7s (média 814ms/execução), FecLado executou 5x no mesmo período (média 1140ms/execução), Display executou 2x (média 2025ms/execução).

Problemas Identificados

Starvation Parcial, por exemplo Thread Display (-13) teve apenas 2 execuções em 5.7s e 71% do tempo alocado para Sensor (-15). identificamos Jitter Temporal com variação de ±15ms nos intervalos declarados.

Relatório de Teste - Modo Preemptivo no Zephyr RTOS

1. Configuração do Ambiente

Especificações Técnicas

1. Plataforma: QEMU x86
2. Versão Zephyr: 4.2.99
3. Configurações Principais:

CONFIG_MAIN_THREAD_PRIORITY=0

CONFIG_NUM_PREEMPT_PRIORITIES=16

CONFIG_NUM_COOP_PRIORITIES=0

2. Hierarquia de Prioridades

Thread	Prioridade	Tipo	Intervalo
Sensor	0 (HIGH)	Preemptiva	500ms
TecLado	1	Preemptiva	1000ms
Display	2 (LOW)	Preemptiva	2000ms

3. Dados Quantitativos

Métrica	Sensor (0)	TecLado (1)	Display (2)
Execuções em 5.7s	**11**	**5**	**2**
Intervalo Médio	**518ms**	**1036ms**	**1980ms**
Desvio Padrão	**±12ms**	**±25ms**	**±40ms**

4. Comportamento Esperado vs Observado

Característica	Esperado	Observado
Ordem de Execução	Sensor → TecLado → Display	Preempção correta conforme prioridades
Temporização	Intervalos exatos	Pequeno jitter (±40ms)
Preempção	Imediata	Confirmada (Sensor interrompe outras)
Starvation	Não deveria ocorrer	Display executou menos que o esperado

5. Problemas Identificados

Starvation Relativo:

- Thread Display (prioridade 2) teve apenas 2 execuções em 5.7s
- 81% do tempo de CPU alocado para Sensor
-
- **Acúmulo de Delay**:
- TecLado apresentou atraso acumulativo de ~36ms

Precisão Temporal:

- Jitter maior que o esperado para sistema preemptivo

5 Saida no terminal

```
[Sensor] Prioridade: 0 | Tempo: 1779910 ms
[Sensor] Prioridade: 0 | Tempo: 1780420 ms
[Teclado] Prioridade: 1 | Tempo: 1780640 ms
[Display] Prioridade: 2 | Tempo: 1780870 ms
[Sensor] Prioridade: 0 | Tempo: 1780930 ms
[Sensor] Prioridade: 0 | Tempo: 1781440 ms
[Teclado] Prioridade: 1 | Tempo: 1781650 ms
[Sensor] Prioridade: 0 | Tempo: 1781950 ms
[Sensor] Prioridade: 0 | Tempo: 1782460 ms
[Teclado] Prioridade: 1 | Tempo: 1782660 ms
[Display] Prioridade: 2 | Tempo: 1782880 ms
[Sensor] Prioridade: 0 | Tempo: 1782970 ms
[Sensor] Prioridade: 0 | Tempo: 1783480 ms
[Teclado] Prioridade: 1 | Tempo: 1783670 ms
[Sensor] Prioridade: 0 | Tempo: 1783990 ms
[Sensor] Prioridade: 0 | Tempo: 1784500 ms
[Teclado] Prioridade: 1 | Tempo: 1784680 ms
[Display] Prioridade: 2 | Tempo: 1784890 ms
[Sensor] Prioridade: 0 | Tempo: 1785010 ms
[Sensor] Prioridade: 0 | Tempo: 1785520 ms
[Teclado] Prioridade: 1 | Tempo: 1785690 ms
[Sensor] Prioridade: 0 | Tempo: 1786030 ms
[Sensor] Prioridade: 0 | Tempo: 1786540 ms
[Teclado] Prioridade: 1 | Tempo: 1786700 ms
```

6.Análise e Discussão

- **Preemptivo com prioridade fixa:** A tarefa Sensor (prioridade 0) sempre preempteia as outras. Isso garante baixa latência e alta responsividade para tarefas críticas.

- **Round Robin (com mesma prioridade):** Ao atribuir mesma prioridade para Sensor, Teclado e Display, observou-se um revezamento uniforme, devido ao time slicing habilitado.

- **Modo cooperativo (opcional):** Ao utilizar K_PRIO_COOP(n) e k_yield(), foi possível simular comportamento onde as tarefas só liberam o processador manualmente.