Sistemas Embarcados - Trabalho Prático I

O RTOS HellfireOS possui escalonamento em dois níveis, conforme apresentado em sala de aula. O primeiro nível (escalonador de tempo real) tem como objetivo escalonar as tarefas de tempo real de acordo com seus parâmetros (período, capacidade e deadline) e a política de escalonamento *Rate Monotonic* (RM). O segundo nível de escalonamento é responsável pelo escalonamento das tarefas de melhor esforço, ou seja, aquelas que não possuem parâmetros de tempo real (*period == 0*, *capacity == 0* e *deadline == 0*). No segundo nível as tarefas compartilham o tempo de CPU utilizando o algoritmo *Round-Robin* com um sistema simples de prioridades relativas.

Parte 1: Sua tarefa consiste em modificar o sistema operacional para que este implemente o suporte ao escalonamento de tarefas aperiódicas. Para isso, você terá que modificar o *kernel* do HellfireOS (contido no diretório */sys*). O mecanismo a ser implementado consiste em um servidor de tarefas aperiódicas (*Polling Server*). O servidor aperiódico deve trabalhar juntamente com o *kernel*, com o objetivo de retirar tarefas aperiódicas de uma fila ou lista e executar *jobs* dentro do tempo da tarefa de tempo real do servidor *Polling Server*. Os escalonadores do *kernel* (tempo real e melhor esforço) não devem escalonar tarefas aperiódicas. Assim, o *kernel* terá três níveis de escalonamento: tempo real, aperiódico e melhor esforço.

Parte 2: Para demonstrar o funcionamento do seu servidor, descreva uma aplicação que consiste em:

- Tarefa *Polling Server*;
- Tarefas de tempo real periódicas (inicialmente pode ser usada a aplicação *sched_ test2*);
- Uma tarefa que realiza o disparo (criação) de tarefas aperiódicas entre 50 ms e 500 ms aproximadamente (use as funções *delay_ms()* e *random()*). Esta pode ser uma tarefa de melhor esforço;

Parte 3: Escrever um relatório, apresentando as modificações que foram realizadas no *kernel*, algoritmo implementado, organização para demonstração do servidor aperiódico e comentários gerais. É importante que seja feita uma análise do desempenho do escalonador, incluindo parâmetros como *jitter* e *delay* para diferentes cenários (conjuntos de tarefas periódicas e aperiódicas) que demonstrem seu comportamento. A leitura de tempo pode ser feita a partir de um temporizador do hardware, com o uso da função *_read_us()*. O trabalho deve ser realizado em duplas ou trios e entregue pelo Moodle por um dos integrantes.

Anexo - detalhes da implementação

As principais modificações a serem implementadas no kernel são as seguintes:

1. *kernel.h* - adicionar um ponteiro para a fila ou lista de tarefas aperiódicas;
2. *main.c* - inicializar a fila / lista junto com as outras já existentes (tempo real e melhor esforço);
3. *task.c* - modificar *hf_spawn()* e *hf_kill()* pra funcionar com tarefas aperiódicas. Dessa forma, tarefas são enquadradas em uma categoria (tempo real, melhor esforço ou aperiódica) de acordo com seus parâmetros;
   - uma tarefa que possui período > 0, capacidade > 0 e deadline > 0 é definida como tempo real;
   - uma tarefa que possui período, capacidade e deadline $== 0$ é definida como melhor esforço;
   - uma tarefa que possui período e deadline $== 0$, mas possui capacidade > 0 é definida como aperiódica.
4. *scheduler.c* - modificar para incluir o servidor aperiódico. O servidor implementa:
   (a) um *dispatcher* para tarefas aperiódicas (verifique o dispatcher implementado em *scheduler.c*);
   (b) um escalonador (que gerencia uma fila circular de tarefas aperiódicas).

O comportamento do servidor aperiódico pode ser descrito resumidamente a partir das seguintes ações:

1. Verificar se existem tarefas aperiódicas na fila (com *hf_queue_count()*); Se não existirem, retornar (continuar o escalonamento da próxima classe, melhor esforço usando a chamada *hf_yield()*);
2. Existem tarefas aperiódicas, então deve-se pegar a primeira da fila (com *hf_queue_get()*);
3. Se ainda existirem *jobs* a serem executados,
   (a) Decrementa a contagem de *jobs*;
   (b) Escalona essa task aperiódica. Para isso, salvar o contexto de execução da tarefa atual (*Polling Server*) e restaurar o contexto da tarefa aperiódica;
4. Senão
   (a) Remove da fila e da um *hf_kill()* na task aperiódica;
   (b) Volta pro item 1 (verificar se existem tasks aperiódicas);