Trabalho Prático I

Hariel D. Giacomuzzi[1], **Leonardo G. Carvalho**[1], **Matheus S. Redecker**[1]

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS),
Avenida Ipiranga, 6681. Prédio 32, CEP 90619-900. Porto Alegre, RS-Brasil

{hariel.dias, leonardo.gubert, matheus.redecker}@acad.pucrs.br

1. Algoritmos de Escalonamento

O objetivo deste trabalho é alterar a forma de escalonamento do Sistema Operacional HellfireOS[1]. O algoritmo sendo utilizado por padrão é o Rate Monotonic (RM), e o algoritmo de utilizado neste trabalho será o Earliest Deadline First (EDF). Após a implementação do algoritmo iremos comparar os resultados obtidos no sistema com os obtidos utilizando o simulador Cheddar[2]. Vale ressaltar que em ambos os tipos de escalonamento estamos assumindo que o *deadline* da tarefa é igual ao seu período.

1.1. Rate-Monotonic

O algoritmo de Rate-Monotonic (RM) é um algoritmo de escalonamento utilizado em sistemas de tempo-real que leva em consideração a duração das tarefas do sistema. Quanto **menor** a duração da tarefa, **maior** será sua prioridade para ser escalonada.

1.2. Earliest Deadline First

O algoritmo EDF tem como objetivo priorizar a execução de tarefas do sistema cujo *deadline* está mais perto de expirar. Este algoritmo tem a capacidade de utilizar até 100% da capacidade de processamento do sistema em que se encontra. A implementação dele no ambiente do HellfireOS pode ser observada abaixo.

```
/**
 *  Sort the queue of real time tasks by order of deadline,
 * in order to implement the EDF algorithm.
 **/
static void sort_queue_by_deadline(void)
{
    int32_t i, j, cnt;
    struct tcb_entry *e1, *e2;

    cnt = hf_queue_count(krnl_rt_queue);
    for (i = 0; i < cnt-1; i++){
        for (j = i + 1; j < cnt; j++){
            e1 = hf_queue_get(krnl_rt_queue, i);
            e2 = hf_queue_get(krnl_rt_queue, j);
            if (e1->deadline_rem > e2->deadline_rem)
                if (hf_queue_swap(krnl_rt_queue, i, j)) panic(PANIC_CANT_SWAP);
        }
    }
}
```

[1]https://github.com/sjohann81/hellfireos
[2]http://beru.univ-brest.fr/ singhoff/cheddar/

```
int32_t sched_edf(void)
{
    int32_t i, k;
    uint16_t id = 0;

    k = hf_queue_count(krnl_rt_queue);
    if (k == 0)
        return 0;

    sort_rt_queue_by_deadline();

    for (i = 0; i < k; i++){
        krnl_task = hf_queue_remhead(krnl_rt_queue);
        if (!krnl_task) panic(PANIC_NO_TASKS_RT);
        if (hf_queue_addtail(krnl_rt_queue, krnl_task))
            panic(PANIC_CANT_PLACE_RT);
        if (krnl_task->state != TASK_BLOCKED &&
            krnl_task->capacity_rem > 0 && !id){
            id = krnl_task->id;
            if (--krnl_task->capacity_rem == 0)
                krnl_task->rtjobs++;
        }
        if (--krnl_task->deadline_rem == 0){
            krnl_task->deadline_rem = krnl_task->period;
            if (krnl_task->capacity_rem > 0) krnl_task->deadline_misses++;
            krnl_task->capacity_rem = krnl_task->capacity;
        }
    }

    if (id){
        krnl_task = &krnl_tcb[id];
        return id;
    }else{
        krnl_task = &krnl_tcb[0];
        return 0;
    }
}
```

2. Aplicações sintéticas

Para realizarmos a análise dos algoritmos, criamos três aplicações sintéticas. As aplicações sintéticas são utilizadas apenas para simular o comportamento de tarefas reais em um Sistema Operacional, porém não realizam nenhuma tarefa significativa. As mesmas tarefas sintéticas foram modeladas na ferramenta de simulação Cheddar, com parâmetros apresentados na Tabela **??**. Para cada aplicação apresentaremos uma descrição do motivo da utilização desses parâmetros, o código usado para executar a aplicação no HellfireOS e uma amostra de escalonamento gerada no Cheddar e outra no Kprofiler. No Chedder o escalonamento para os dois algoritmos estão na mesma imagem e aparecem de forma intercalada, sendo primeiro o escalonamento da tarefa utilizando EDF e em seguida a mesma tarefa sendo escalonada com RM.

2.1. Aplicação 1

A aplicação 1 foi pensada para ter uma utilização que os dois algoritmos consigam executar normalmente, e é o que acontece. Com uma utilização de aproximadamente 65% os

Tabela 1. Especificação das Tarefas

Aplicação	Tarefa	Periodo	Capacidade	Deadline	Utilização
1					0,6483
	T1	15	2	15	0,1333
	T2	30	1	30	0,0333
	T3	20	2	20	0,1
	T4	24	4	24	0,1667
	T5	100	9	100	0,09
	T6	40	5	40	0,125
2					0,9011
	T1	12	1	12	0,0833
	T2	10	3	10	0,3
	T3	15	1	15	0,0667
	T4	17	2	17	0,1177
	T5	30	5	30	0,1667
	T6	24	4	24	0,1667
3					1
	T1	10	2	10	0,2
	T2	30	5	30	0,1667
	T3	50	10	50	0,2
	T4	30	6	30	0,2
	T5	30	1	30	0,0333
	T6	40	8	40	0,2

dois algoritmos escalonam fácil esse conjunto de tarefas.

2.1.1. Código

```
#include <hellfire.h>

void task(void){
        for(;;){
                printf("task %d \n", hf_selfid());
                delay_ms(200);
        }
}
void app_main(void){
        hf_spawn(task, 15, 2, 15, "task 1", 1024);
        hf_spawn(task, 30, 1, 30, "task 2", 1024);
        hf_spawn(task, 20, 2, 20, "task 3", 1024);
        hf_spawn(task, 24, 4, 24, "task 4", 1024);
        hf_spawn(task, 100, 9, 100, "task 5", 1024);
        hf_spawn(task, 40, 5, 40, "task 6", 1024);
}
```

2.1.2. Amostra do Escalonamento

A Figura **??** ilustra uma amostra do escalonamento gerado pelo HellfireOS, a Figura **??** mostra o escalonamento de dois processadores, um utilizando EDF e o outro RM.

Figura 1. Aplicação 1 HellfireOS

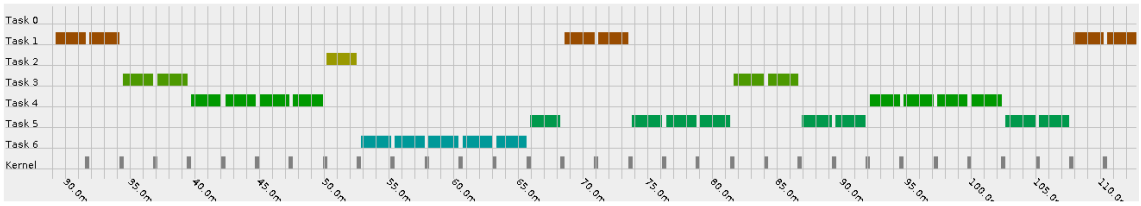
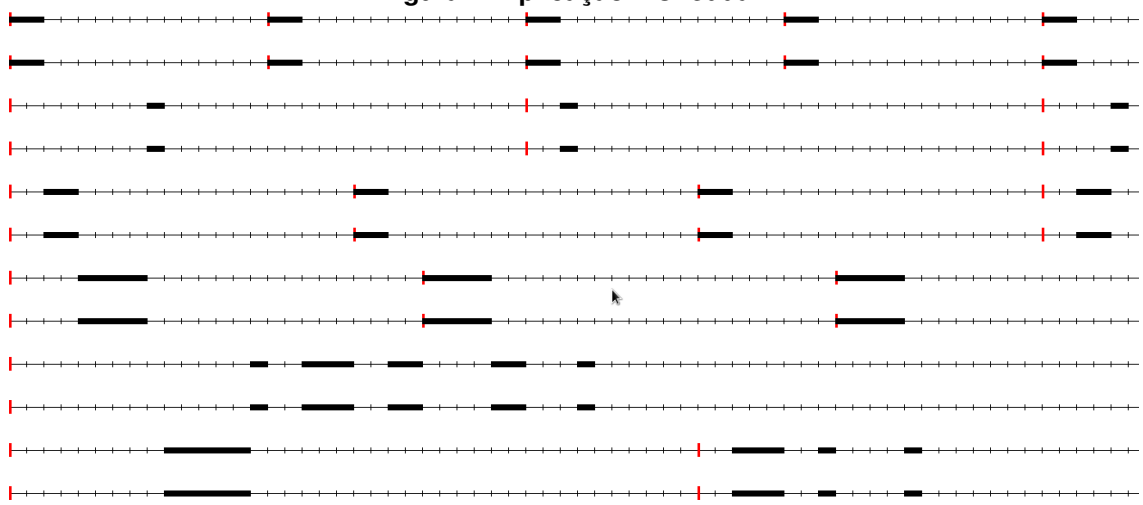


Figura 2. Aplicação 1 Cheddar



2.2. Aplicação 2

A aplicação 2 foi pensada para que a utilização ultrapasse 85%. Com o intuito de avaliar se os dois algoritmos continuam escalonando normalmente. A partir da analise dos escalonamentos, podemos notar que o algoritmo RM tem *deadlines* perdidos na tarefa 5, e por isso não consegue ser escalonável. Já o EDF escalona todas as tarefas sem perdas de *deadline*.

2.2.1. Código

```
#include <hellfire.h>

void task(void){
        for(;;){
                printf("task %d \n", hf_selfid());
                delay_ms(200);
        }
}
void app_main(void){
        hf_spawn(task, 12, 1, 12, "task 1", 1024);
```

```
        hf_spawn(task, 10, 3, 10, "task 2", 1024);
        hf_spawn(task, 15, 1, 15, "task 3", 1024);
        hf_spawn(task, 17, 2, 17, "task 4", 1024);
        hf_spawn(task, 30, 5, 30, "task 5", 1024);
        hf_spawn(task, 24, 4, 24, "task 6", 1024);
}
```

2.2.2. Amostra do Escalonamento

A Figura **??** ilustra uma amostra do escalonamento gerado pelo HellfireOS com o algoritmo de RM e a Figura **??** o escalonamento utilizando EDF, já a Figura **??** mostra o escalonamento de dois processadores, um utilizando EDF e o outro RM.

Figura 3. Aplicação 2 HellfireOS com algoritmo de RM

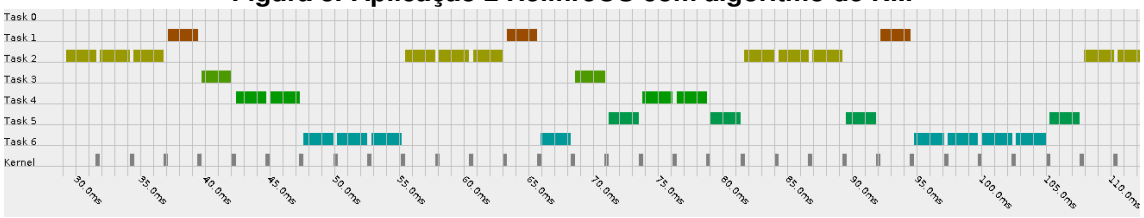


Figura 4. Aplicação 2 HellfireOS com algoritmo de EDF

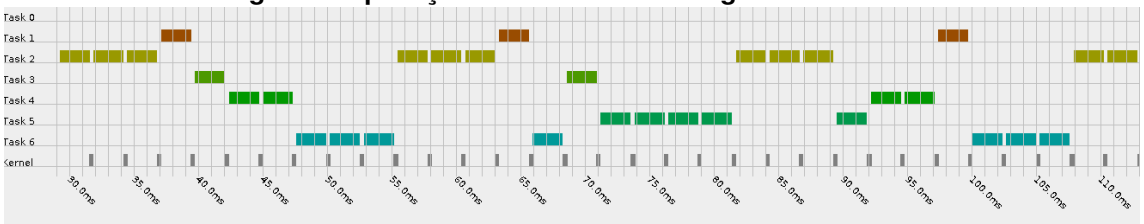
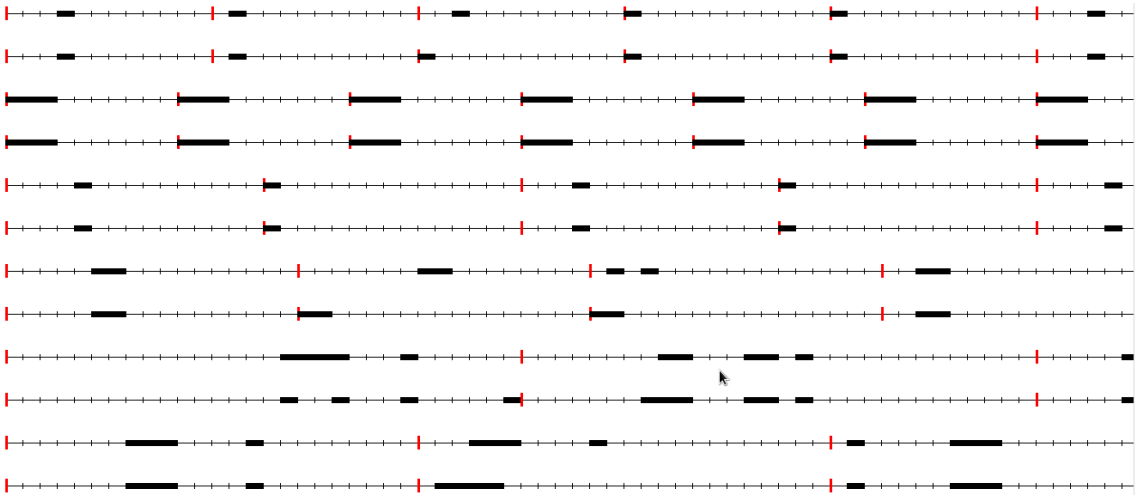


Figura 5. Aplicação 2 Cheddar



2.3. Aplicação 3

A aplicação 3 foi projetada para que a sua utilização seja 100%. O intuito de ter uma utilização igual a capacidade máxima é analisar se o algoritmo consegue utilizar todo o

poder de processamento disponível. Podemos notar que o algoritmo de EDF continua escalonando normalmente e não tem perdas de *deadlines*, já no RM há varias perdas de *deadline* na tarefa 3.

2.3.1. Código

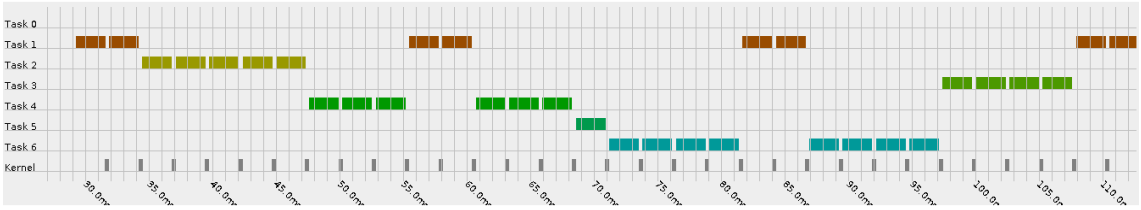
```
#include <hellfire.h>

void task(void){
        for(;;){
                printf("task %d \n", hf_selfid());
                delay_ms(500);
        }
}
void app_main(void){
        hf_spawn(task, 10, 2, 10, "task 1", 1024);
        hf_spawn(task, 30, 5, 30, "task 2", 1024);
        hf_spawn(task, 50, 10, 50, "task 3", 1024);
        hf_spawn(task, 30, 6, 30, "task 4", 1024);
        hf_spawn(task, 30, 1, 30, "task 5", 1024);
        hf_spawn(task, 40, 8, 40, "task 6", 1024);
}
```

2.3.2. Amostra do Escalonamento

A Figura **??** ilustra uma amostra do escalonamento gerado pelo HellfireOS, a Figura **??** mostra o escalonamento de dois processadores, um utilizando EDF e o outro RM.

Figura 6. Aplicação 3 HellfireOS



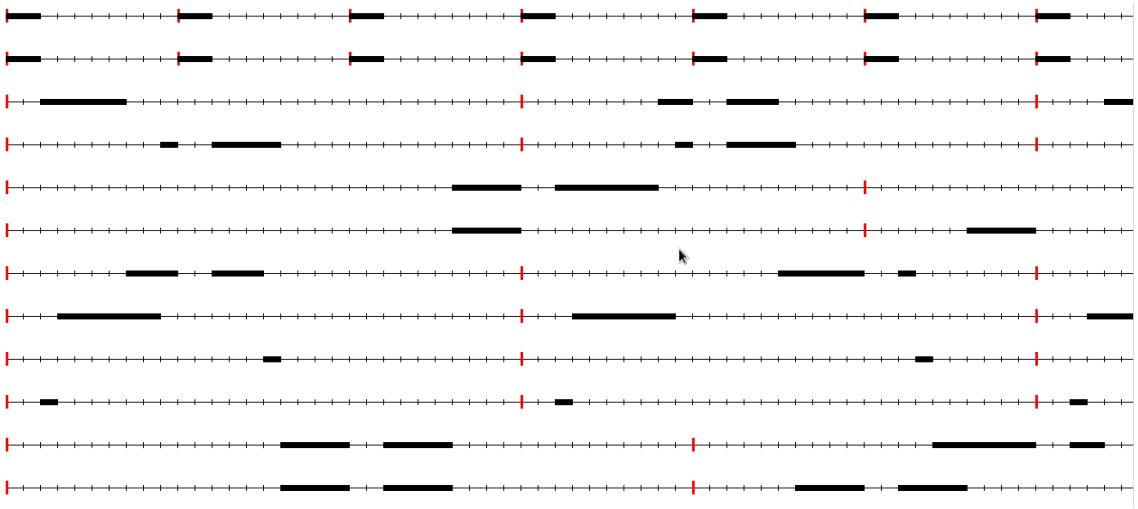
3. Dificuldades encontradas

Tivemos dificuldades no entendimento do funcionamento do HellfireOS na hora de analisar os resultados, pois como o simulador não mostra quando os *deadline* irão acabar fica um pouco mais dificil de identificar. Ao contrário do que acontece na ferramenta Cheddar, onde o *deadline* fica evidente e acusa quando acontecem perdas de *deadline*.

4. Comentários finais

Como podemos analisar através das aplicações, o algoritmo RM não conseguiu escalonar tarefas com uma utilização maior que 90%. Enquanto o EDF chegou a utilizar todo o processamento disponível, ou seja, 100% da CPU. O EDF consegue explorar melhor os recursos computacionais e consegue ser mais responsivo a tarefas aperiódicas.

Figura 7. Aplicação 3 Cheddar



Após pesquisar um pouco sobre os algoritmos, foi possível ver que sistemas embarcados que trabalham com recursos computacionais limitados e sistemas multimídias conseguem ser mais eficientes sobre um escalonamento EDF.