Implementação e Avaliação de Algoritmos de Escalonamento com Simulação de Timer em Hardware

Lorina Zondervan, Rafael Fernando Dos Reis Mecabô, Steff Kerry Toussaint

UNIVALI - Universidade do Vale de Itajaí – Ciência da Computação

zondervan@edu.univali.br, rafa_ferreis@edu.univali.br, steff@edu.univali.br

Abstract: This paper presents the implementation of different scheduling algorithms, including Round Robin, Round Robin with Priority, EDF, and Aging-based Priority, using a simulated timer through a thread. Features include time slice handling, ready queue management, and deadline or task aging control.

Resumo: Este trabalho apresenta a implementação de diferentes algoritmos de escalonamento, incluindo Round Robin, Round Robin com Prioridade, EDF e Prioridade com Aging, utilizando simulação de timer via thread. Foram considerados aspectos como gerenciamento de slice de tempo, manutenção da fila de aptos e manipulação de deadlines ou envelhecimento de tarefas.

1. Introdução

Este projeto visa a implementação de algoritmos de escalonamento de tarefas utilizando a linguagem C. Através da simulação de um timer com threads, foram implementados os algoritmos Round Robin (RR), Round Robin com Prioridade (RRP), Escalonamento por Menor Deadline (EDF) e Prioridade com Aging (PA).

2. Metodologia

Para implementar os algoritmos de escalonamento, foi utilizada uma simulação do funcionamento de um timer em hardware através de threads. A estrutura do código foi organizada de forma que a lista de tarefas fosse mantida durante toda a execução do sistema, com controle de prioridade e time slices. As funções para manipulação da fila de tarefas e o gerenciamento de tempo foram centralizadas em bibliotecas auxiliares.

3.1. Algoritmo Round Robin (RR)

O algoritmo round robin distribui o tempo de CPU entre as tarefas, alocando um quantum fixo de tempo para cada uma. Quando o tempo de execução de uma tarefa expira, ela é movida para o final da fila, permitindo que as demais tarefas sejam executadas.

```
void schedule_rr() {
    while (tasksRemaining) {
        tasksRemaining = 0;
        current = task_list_rr;
        while (current) {
            Task *t = current->task;
            if (t->burst > 0) {
                tasksRemaining = 1;
                int slice = (t->burst < QUANTUM) ? t->burst :
QUANTUM;
                run(t, slice);
                t->burst -= slice;
            }
            current = current->next;
        }
    }
}
```

O código percorre a lista de tarefas, executando cada uma delas por um tempo definido pelo quantum. após o tempo de execução, a tarefa é reavaliada, e se ainda precisar de mais tempo de cpu, ela volta para o final da fila.

3.2. Algoritmo Round Robin com Prioridade (RRP)

O round robin com prioridade mantém a ideia do Round Robin, mas com a adição de uma fila por prioridade. As tarefas com maior prioridade são executadas primeiro, embora ainda mantenham o conceito de quantum para evitar o monopolismo de CPU por tarefas de alta prioridade.

```
void schedule_rr_p() {
    while (tasksRemaining) {
        tasksRemaining = 0;
        for (int prio = MIN_PRIORITY; prio <= MAX_PRIORITY; prio++)
{
            struct node *current = priority_queues[prio];
            while (current) {
                Task *t = current->task;
                if (t->burst > 0) {
                    tasksRemaining = 1;
                    int slice = (t->burst < QUANTUM) ? t->burst :
QUANTUM;
                    run(t, slice);
                    t->burst -= slice;
                }
                current = current->next;
            }
        }
    }
}
```

O algoritmo percorre as filas de tarefas organizadas por prioridade. As tarefas de maior prioridade são executadas primeiro, respeitando o quantum de tempo. O algoritmo garante que as tarefas mais urgentes sejam processadas antes das menos prioritárias.

3.3. Algoritmo EDF (Earliest Deadline First)

O algoritmo Earliest Deadline First (EDF) escalona as tarefas com base no prazo de execução, ou deadline. A tarefa com o menor deadline é priorizada, o que é crucial para sistemas de tempo real.

```
void schedule_edf() {
    while (task_list_edf != NULL) {
        struct node *earliest =
find_earliest_deadline(task_list_edf);
        Task *t = earliest->task;

        while (t->burst > 0) {
            while (!time_expired);
            time_expired = 0;
            run(t, 1);
            t->burst -= 1;
```

```
            time_global += 1;
        }

        delete(&task_list_edf, t);
        free(t);
    }
}
```

A tarefa com o menor deadline é selecionada e executada. A execução é feita em unidades de tempo fixas (slices) e, ao final, a tarefa é removida da lista de tarefas. O algoritmo utiliza um timer simulado para controlar o avanço do tempo e garantir que as tarefas cumpram seus prazos

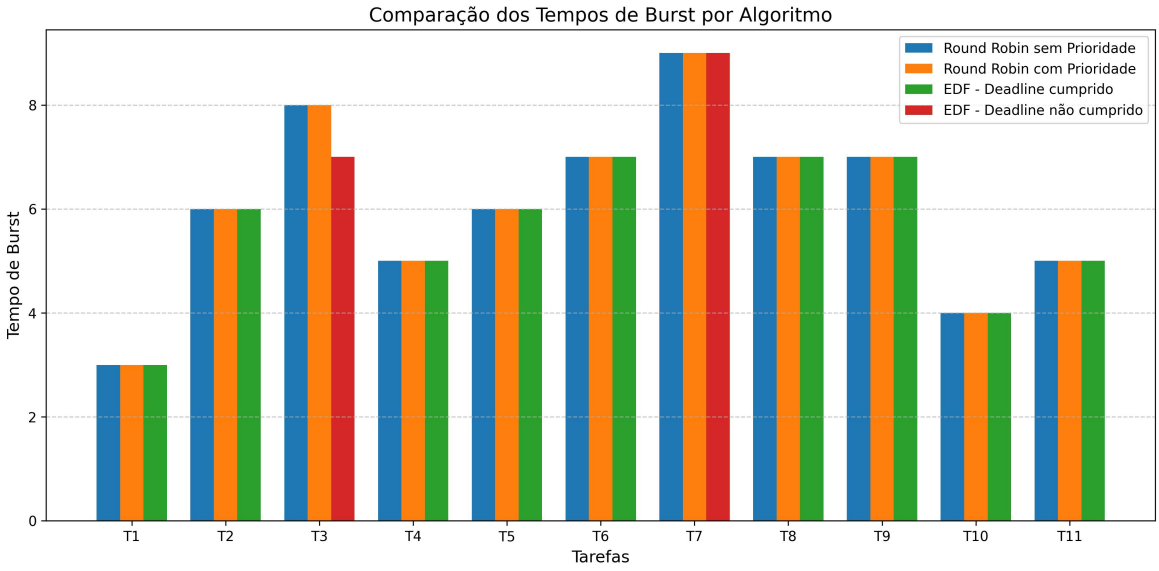
3.4. Algoritmo Prioridade com Aging (PA)

O algoritmo Prioridade com Aging aplica um mecanismo de envelhecimento para tarefas que ficam muito tempo na fila sem execução. O tempo de espera de uma tarefa aumenta sua prioridade, garantindo que ela seja eventualmente executada, mesmo que suas prioridades iniciais sejam baixas.

```
void apply_aging() {
    for (int prio = MAX_PRIORITY; prio >= MIN_PRIORITY + 1; prio--)
{
        struct node *current = priority_queues[prio];
        while (current) {
            Task *t = current->task;
            if (t->burst > 0 && t->waiting_time >= AGING_THRESHOLD)
{
                t->priority -= 1;
                t->waiting_time = 0;
                insert(&priority_queues[t->priority], t);
            }
            current = current->next;
        }
    }
}
```

Ele percorre as filas de tarefas de menor prioridade para maior, verificando se cada tarefa ainda possui tempo de execução e se já esperou tempo suficiente na fila. Quando essas condições são atendidas, a prioridade da tarefa é e seu tempo de espera é zerado. Em seguida, a tarefa é reinserida na fila correspondente à nova prioridade, promovendo sua execução futura e evitando que fique indefinidamente esperando na fila. Esse mecanismo garante justiça na execução, prevenindo a fome de tarefas com baixa prioridade.

4. Resultados



Para avaliar o desempenho dos algoritmos implementados round robin, round robin com prioridade e earliest deadline first, foram simuladas 11 tarefas com diferentes tempos de burst. O gráfico a seguir apresenta a comparação dos tempos de burst para cada algoritmo, destacando ainda os casos em que o algoritmo earliest deadline first cumpriu ou não o deadline das tarefas.

Observa-se que o e earliest deadline first, por priorizar as tarefas com menor deadline, tende a reduzir o tempo de burst para tarefas críticas, mas em alguns casos não consegue cumprir os deadlines, evidenciado pelas barras vermelhas. Já os algoritmos round robin e round robin com prioridade distribuem o tempo de CPU de maneira mais uniforme, porém sem garantia de cumprimento de prazos.

5. Conclusão

A implementação e avaliação dos algoritmos de escalonamento Round Robin, Round Robin com Prioridade e Earliest Deadline First demonstraram características distintas de cada abordagem. O algoritmo EDF destacou-se pela capacidade de priorizar tarefas com deadlines mais próximos, o que é essencial para sistemas de tempo real, embora tenha apresentado algumas falhas no cumprimento dos prazos sob condições de alta carga. Por sua vez, o Round Robin com Prioridade mostrou-se eficiente na alocação da CPU, favorecendo tarefas de maior prioridade e reduzindo o tempo de espera dessas em comparação ao Round Robin simples, que, apesar de garantir uma distribuição justa do

tempo de CPU entre todas as tarefas, não diferencia prioridades ou deadlines. Dessa forma, a escolha do algoritmo ideal deve levar em consideração os requisitos específicos do sistema, como a necessidade de cumprimento rigoroso de prazos ou a justiça na distribuição dos recursos computacionais, ressaltando a importância da análise contextual para a aplicação prática dos algoritmos de escalonamento.

6. Referências

ANENBAUM, Andrew S.; BOS, Herbert. *Modern Operating Systems*. 5. ed. Boston: Pearson, 2023.

SILBERSCHATZ, Abraham; GALVIN, Peter Baer; GAGNE, Greg. *Operating System Concepts*. 8. ed. Hoboken: Wiley, 2008.

TANENBAUM, Andrew S.; WOODHULL, Albert S. *Operating Systems: Design and Implementation*. 3. ed. Upper Saddle River: Prentice Hall, 2006.