UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO

ALGORITMOS E ESTRUTURA DE DADOS 1

Relatório – Simulador de Escalonamento de Processos de um Sistema Operacional

Amanda Aparecida Gonçalves Chaves

Anderson Carlos da Silva Morais

Marília Fonseca Andrade

Pau dos Ferros - RN

28/07/2025

Arquitetura e Organização

- **Queue.h**: Arquivo que descreve as estruturas de dados utilizados em todo o sistema para representar o processo e o fluxo de execução. Ele define:
    - **Processo:** encapsula todos os dados de uma tarefa (ID, tempo de chegada, tempo de execução e tempo de descanso)
    - **Queue:** gerencia uma fila de processos.
- **main.c**: possui a função do algoritmo round_robin, além da função main onde se inicia um teste com um array de processos.
- **Queue.c**: Este módulo realiza todas as operações de manipulação da fila de processos. Ele contém a lógica para as principais funções de uma estrutura do tipo fila, como adicionar um processo ao final do arquivo (queue_enqueue), remover o próximo processo do início do arquivo para execução (queue_dequeue) e exibir o estado atual da fila.

Lógica de Simulação

O algoritmo Round-Robin, se trata de um algoritmo que executa processos em frações de tempo (*quantum*). O algoritmo permite escalonar processos em um sistema operacional (SO), em que os processos que se encontram no estado de pronto passem a ser executados pelo processador por uma quantidade de tempo (*quantum)*, garantindo que todos os processos tenham acesso ao processador por um tempo fixo.

Os processos podem ser organizados em quatro principais categorias: em execução, prontos, concluídos, bloqueado ou esperando. Os processos que estão prontos são adicionados a uma fila, onde cada processo será executado de acordo com a ordem de chegada pela mesma quantidade de tempo, e, caso o processo necessite de mais tempo para ser executado, ele deve ser adicionado novamente a fila para outra rodada de execução. Além disso, outros processos podem ser adicionados a fila em qualquer momento de execução, necessitando que o algoritmo identifique quando um novo processo chegue e adicione à fila.

Para entendimento do projeto, é necessário entender a estrutura do processo, que possui quatro variáveis do tipo inteiro:

1. **Id:** simula o id único do processo;

2. **Tempo_execucao_total**: o tempo total que o processo requisita de execução;
3. **Tempo_restante:** o tempo que o processo tem para ser executado;
4. **Tempo_chegada:** o identificador de quando o processo chegou no algoritmo de escalonamento, sendo usado para ser adicionado o processo na fila.

```c
typedef struct Processo
{
    int id;                 // ID do processo
    int tempo_execucao_total; // Tempo original
    int tempo_restante;       // Tempo que falta
    int tempo_chegada;        // Momento que entra no sistema
} Processo;
```

Funções

Round_robin

Função do algoritmo de escalonamento de round-Robin.

- retorno: void;
- parâmetros:
    - Ponteiro de processos para percorre uma quantidade determinada de processos;
    - Total de processos utilizado para controle de loop;
    - Timer: representa o quantum, ou seja, valor padronizado que cada processo será executado.

1. A função cria e inicializa uma nova fila para armazenar os processos prontos, além de criar variáveis de escopo para controle interno como:
    a. Tempo_atual: se trata do tempo de execução do algoritmo usado para descobri quando um novo processo deve ser adicionado;
    b. Processos_concluidos: quantidade de processos que já foram concluídos;
    c. Processo_next_indice: variável de controle para gerenciar qual o próximo processo.

```c
void test(Processo *processos, int tot_processos, int timer)
{
    Queue fila_prontos;
    queue_init(&fila_prontos);

    int tempo_atual = 0;
    int processos_concluidos = 0;
    int processo_next_indice = 0;

    printf("Iniciando Simulacao (Quantum: %ds)\n", timer);
```

2. Início do loop principal, para continuar em execução enquanto os processos não forem concluídos
    a. Loop interno para adiciona a fila de prontos todos os processos que obedecem a duas condições:
        i. É um processo dentro da lista de processos repassada, ou seja, seu índice está dentro da quantidade de processo;
        ii. O tempo_chegada do processo é menor ou igual ao atual, logo, todo processo que o tempo de chegada estiver dentro do tempo atual do algoritmo será adicionado.
    b. O Loop interno irá exibir na tela qual o processo e em sequência adicionar ele a fila, também atualizando o índice para que seja verificado o próximo processo.

```c
// O loop principal continua enquanto houver processos a chegar ou na fila
while (processos_concluidos < tot_processos)
{
    // VERIFICAR CHEGADAS
    // Adiciona à fila de prontos todos os processos que chegaram neste instante de tempo
    while (processo_next_indice < tot_processos &&
           processos[processo_next_indice].tempo_chegada <= tempo_atual)
    {
        printf("[Tempo %d] Chegada: Processo %d (Execucao: %ds)\n",
               tempo_atual,
               processos[processo_next_indice].id,
               processos[processo_next_indice].tempo_execucao_total);

        queue_enqueue(&fila_prontos, processos[processo_next_indice]);
        processo_next_indice++;
    }
```

3. Execução dos processos, esse passo só ocorre se existir processos prontos na fila de pronto.
    a. Nesse cenário é removido o processo mais antigo da fila, e calculado o tempo de execução do processo por meio de uma if-ternário;
        i. O tempo máximo é definido por timer, logo, se o tempo restante do processo for menor que o tempo máximo, para evitar desperdícios, o processo é executado pela quantidade restante;
        ii. Se o tempo_restante for maior ou igual ao tempo máximo (timer), então ele será executado pela quantidade de tempo máximo.
    b. O tempo_atual do algoritmo é incrementado pelo tempo que a execução levou, seja o valor máximo ou o tempo_restante do processo;
    c. O tempo restante do processo é decrementado pelo tempo de execução em que o processo foi executado.

```
// EXECUTAR PROCESSO
if (!queue_is_empty(&fila_prontos))
{
    Processo processo_atual = queue_dequeue(&fila_prontos);

    // Determina quanto tempo este processo vai executar nesta rodada
    int tempo_de_execucao = (processo_atual.tempo_restante < timer)
                            ? processo_atual.tempo_restante
                            : timer;
    printf("[Tempo %d] Executando Processo %d por %ds...\n", tempo_atual, processo_atual.id, tempo_de_execucao);

    // Avança o relógio da simulação
    tempo_atual += tempo_de_execucao;
    processo_atual.tempo_restante -= tempo_de_execucao;
```

4. Outra etapa que é verificada dentro do bloco de execução de processos é verificar se um já está concluído e não precisa de mais tempo de execução.
    a. Essa etapa consiste em verificar se o processo que estava em execução ainda possui tempo_restante para ser executado;
    b. Antes de adicioná-lo novamente a fila, é necessário verificar se novos processos chegaram a fila com o mesmo bloco de 2. Em sequência, o processo é adicionando novamente a fila;
    c. Caso o processo tenha concluído e não possua tempo restante, o algoritmo irá informar na tela e incrementar a variável de processo concluídos;
    d. Finalizando o bloco de execução é exibido na tela todos os processos na fila de processos prontos.

```
    // Verifica se o processo terminou
    if (processo_atual.tempo_restante > 0)
    {
        // Se não terminou volta para a fila
        // verifica se novos processos chegaram enquanto este executava
        while (processo_next_indice < tot_processos &&
               processos[processo_next_indice].tempo_chegada <= tempo_atual)
        {
            printf("[Tempo %d] Chegada: Processo %d (Execucao: %ds)\n",
                   processos[processo_next_indice].tempo_chegada,
                   processos[processo_next_indice].id,
                   processos[processo_next_indice].tempo_execucao_total);
            queue_enqueue(&fila_prontos, processos[processo_next_indice]);
            processo_next_indice++;
        }
        //  o processo atual volta para o fim da fila
        queue_enqueue(&fila_prontos, processo_atual);
    }
    else
    {
        printf("[Tempo %d] Finalizado: Processo %d\n", tempo_atual, processo_atual.id);
        processos_concluidos++;
    }
    queue_display(&fila_prontos);
    printf("-----------------------------------------\n");
}
```

5. Além disso, caso nenhum processo esteja pronto e na fila de processos prontos, o processador permanecerá ocioso por determinado tempo até que um novo processo seja adicionado.
    a. Nesse bloco, apenas é exibido que a CPU está ociosa e o tempo_atual é incrementado sem execução de processos;
    b. Por fim, a o fechado do bloco condicional de execução, do loop principal, quando todos os processos são concluídos é exibido mensagem de finalização da simulação e o fechamento da função.

```
        else
        {
            // Se a fila está vazia, mas ainda há processos para chegar, avança o tempo.
            printf("[Tempo %d] CPU Ociosa...\n", tempo_atual);
            tempo_atual++;
        }
    }
    printf("\nSimulacao concluida em %d segundos.\n", tempo_atual);
}
```

Main

Função principal, é definido um array de Processos de maneira aleatória para serem adicionados ao algoritmo, também são definidos a quantidade de processos e o *quantum*, representado pela variável timer. Finalizando com a chamada da função round_robin.

```
int main()
{
    Processo processos[] = {
        {1, 8, 8, 0}, // id, tempo_exec, tempo_restante, tempo_chegada
        {2, 4, 4, 2},
        {3, 6, 6, 4},
        {4, 2, 2, 7}};

    int tot_processos = sizeof(processos) / sizeof(Processo);
    int timer = 3;

    round_robin(processos, tot_processos, timer);

    return 0;
}
```

Queue

Estrutura de dados para o funcionamento de filas que armazenado valores do tipo inteiro, sua estrutura e construção sendo abordadas na disciplina e não sendo aprofundados no projeto.