Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências da Computação
SCC0201 — Introdução a Ciências da Computação II

Trabalho 03: Simulador de Escalonador de Processos

Professores: Fernando Pereira dos Santos e Moacir Antonelli Ponti

PAEs: Edresson Casanova (C) e Leo Sampaio Ferraz Ribeiro (D)

Monitores: Gabriel Alves Kuabara (B) e Guilherme Amaral Hiromoto (A)

Desenvolva o trabalho sem olhar o de colegas.
Se precisar de ajuda pergunte, a equipe de apoio está aqui por você.

1 Objetivo do Trabalho

Você deverá utilizar os conhecimentos adquiridos até o momento na disciplina para implementar um simulador de escalonador de processos, na linguagem C.

2 Escalonador de Processos

Um processo é um software em execução. Para o Sistema Operacional (SO), é importante a distinção entre estes dois conceitos: um processo possui mais informações que o descrevem, já que se trata de um software em execução, e, portanto, possui informações como o ponto atual de execução e o valor de cada variável declarada, por exemplo. No entanto, um SO pode permitir que diversos processos sejam executados ao mesmo tempo. No entanto, sabe-se que a quantidade de núcleos de um processador é limitada: dado um processador com X núcleos, o SO pode executar, no máximo, X processos ao mesmo tempo, mesmo que o SO possua 2X processos "em execução". Esse comportamento só é possível porque o SO permite que cada processo execute por um breve momento, chamado quantum. A rotina do SO que cuida do "revezamento" entre os processos, decidindo quais podem ser executados em um dado núcleo do processador, é chamada de escalonador de processos.

Na Figura 1, podemos ver a execução do Word, Excel e Power Point, perceba que os processos não executam ao mesmo tempo, e que a combinação da execução de todos os programas resulta na intercalação desses processos no processador. O núcleo do processador não é capaz de executar dois processos ao mesmo tempo, e por isso eles devem "revezar" para serem executados.

No entanto, nós, usuários, enxergamos os processos como executados ao mesmo tempo. Isso acontece porque o *quantum* de um SO geralmente é pequeno (alguns milissegundos), então os processos são alternados frequentemente.

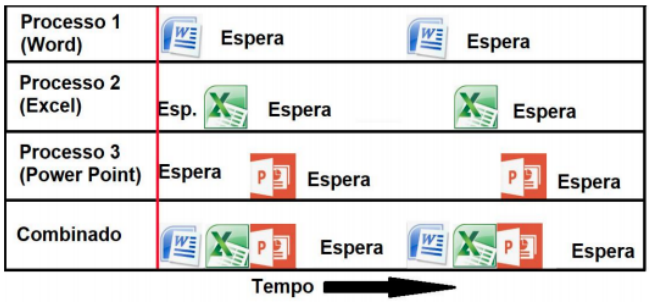

Figure 1: Exemplo de funcionamento de um escalonador para um processador de 1 núcleo. Os processos são intercalados durante a execução conforme o *quantum* de execução. Após a execução rápida, que equivale ao *quantum*, os processos ficam em espera enquanto outros são executados.

No entanto, para garantir que os processos tenham porções justas de tempo em execução, o escalonador deve utilizar algoritmos para decidir quais processos devem ser executados a cada instante. Alguns dos algoritmos mais conhecidos para realizar esta decisão são:

- **First In, First Out (FIFO)**: os processos que são reconhecidos primeiro pelo escalonador são executados até o final. Em geral, este algoritmo utiliza uma fila para organizar os processos. Também é conhecido por *First Come, First Served*;
- **Shortest Remaining Time First**: neste algoritmo os primeiros processos executados são os com menor tempo de execução restante;
- **Round-Robin**: este algoritmo apenas arranja todos os processos em uma lista, e trata todos os processos como tendo a mesma prioridade. Assim, um processo não é executando uma segunda vez antes que todos os outros processos executem ao menos uma vez. Pode-se pensar na execução do *Round Robin* como a utilização de uma lista circular em que todos os elementos recebem a mesma quantidade de recursos. Pode-se pensar na execução do *Round Robin* como uma lista circular, como pode ser observado na Figura 2.
- **Prioridade Fixa Preemptiva**: este algoritmo divide os processos em diferentes níveis de prioridade, e um processo com nível mais alto de prioridade sempre é executado antes de qualquer processo de prioridade mais baixa.

3 Proposta

O trabalho consiste na implementação de um simulador de escalonador de serviços, na linguagem C. O simulador deve receber uma lista de processos, contendo o código de identificação de um processo p_i, o tempo $t0_i$ em que o processo é reconhecido pelo escalonador

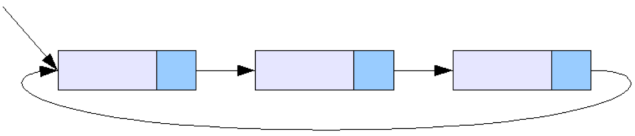

Figure 2: Exemplo de funcionamento do algoritmo *Round Robin*. O primeiro processo executado é o que está na cabeça (head) da lista, seguindo a sequência da lista a partir daí. Após executar o último processo da lista, o algoritmo volta para a cabeça da lista, e estes passos são repetidos até a lista estar vazia.

e o tempo tf_i que o processo leva para ser executado. O simulador deve exibir, como saída, o tempo tf_i em que o processo p_i tem sua execução finalizada. O simulador deve utilizar o tempo tf_i apenas para controlar a execução da simulação, portanto o **escalonador implementado não deve levar o tempo tf_i em consideração**.

O código de identificação p_i de um processo deve ser único. Caso o simulador reconheça um novo processo com o mesmo código, este novo processo deve ter seu código alterado para o próximo código maior disponível. O código p_i de um processo está no intervalo [1, MAXINT]. Os tempos $t0_i$ e tf_i do processo p_i são do tipo **int**.

Cada processo também possui um nível r_i de prioridade. Dados dois processos p_i e p_j, o algoritmo usado é *Round Robin* caso $r_i = r_j$. No entanto, caso $r_i > r_j$ então o *Round Robin* ainda será usado, no entanto iniciando o ciclo de iterações por p_i. O simulador deve possuir apenas 4 níveis de prioridade distintos, e a ordem de execução entre processos de mesmo nível de prioridade é dada por ordem ascendente de código. O simulador não interrompe a execução de processos num dado nível de prioridade, no entanto retorna para um nível de prioridade superior caso haja um novo processo naquele nível. O novo processo sempre é reconhecido pelo escalonador antes de decidir o novo processo a ser executado. Portanto, no momento da "chegada" de um novo processo, o escalonador ainda não deve ter escolhido um novo processo, apontando para o processo executado no ciclo anterior.

Para cada um dos níveis de prioridade, o simulador deve realizar a ordenação dos processos em ordem ascendente de p_i. Para padronizar a contagem do tempo, o simulador deve operar em frequência de 1 quantum: cada ciclo de atividade do simulador corresponde a 1 *quantum* de execução. Portanto, cada iteração do simulador altera o estado dos processos em execução correspondente à passagem de 1 *quantum* de tempo.

3.1 Entradas e Saídas

Os dados do arquivo de entrada definem a lista de processos a ser executada pelo simulador. Dada uma lista $\mathcal{L}$ de processos, cada linha de $\mathcal{L}$ deve conter o código p_i de um processo, seu tempo inicial de entrada no simulador $t0_i$, o volume de *quantum* necessário para ser completamente executado tf_i, e o nível de prioridade do processo r_i. O arquivo de entrada terá o seguinte formato:

- (int)p_0 (int)$t0_0$ (int)tf_0 (int)r_0

- (int)p_1 (int)$t0_1$ (int)tf_1 (int)r_1
- $\vdots$ $\vdots$ $\vdots$ $\vdots$
- (int)p_n (int)$t0_n$ (int)tf_n (int)r_n

Após terminar a execução de todos os processos definidos no arquivo de entrada, o simulador deve criar uma saída, com os dados da simulação. A saída deve possuir, em cada linha, os dados da execução de um único processo p_i, contendo:

1. o código p_i do processo, atualizado pelo simulador;
2. o ciclo de execução do simulador em que a execução de p_i foi finalizada;
3. uma quebra de linha.

Os processos na saída devem ser apresentados em ordem cronológica ascendente (processos que terminam primeiro são exibidos antes).

Por exemplo, dada a seguinte lista $\mathcal{L}$ de processos:

```
333 1 5 4
1571 1 3 3
1571 2 1 2
227 5 2 1
```

A saída produzida deve ser a seguinte:

```
1572 3
1571 8
227 9
333 11
```

4 Submissão

Envie seu código fonte para o run.codes (apenas o arquivo `.c`).

1. **Crie um header com identificação**. Use um header com o nome, número USP, código do curso e o título do trabalho. Uma penalidade na nota será aplicada se seu código estiver faltando o header.
2. **Comente seu código**. O objetivo é que tenhamos um código claro e que facilite a correção. Exemplos: Se uma variável deixa sua função clara em nome, um comentário sobre ela não é necessário; Um loop triplo que acessa um vetor de matrizes e altera os valores a depender da posição provavelmente pede por um comentário explicando a ideia por trás.

3. **Organize seu código em funções**. Use funções para deixar cada passo da execução mais clara, mesmo que a função seja chamada apenas uma vez. Comente sua função, descrevendo suas entradas e saídas, sempre que o nome da função e entradas não deixar isso óbvio.

4. **Utilize Alocação Dinâmica** para armazenar os elementos da matriz e não se esqueça de liberar a memória alocada ao fim da execução.

5. **Busque uma maior eficiência do seu algoritmo**. Busque evitar loops desnecessários para aumentar a velocidade de execução de seu simulador de escalonador de processos, portanto, após resolver todo os demais pontos do trabalho verifique se é possível melhorar o tempo de execução de seu algoritmo, caso seja possível melhore.

6. **Tire Dúvidas com a Equipe de Apoio**. Se não conseguiu chegar em uma solução, dê um tempo para descansar a cabeça e converse com a equipe de apoio sobre a dificuldade encontrada se precisar.