TP1 - Projeto de Execução Dinâmica de Processos

Sistemas Operacionais

O presente trabalho tem por objetivo explorar temas referentes ao escalonamento e troca entre processos que utilizam um dado processador. É previsto o desenvolvimento de um ambiente que empregue uma política de escalonamento específica, bem como gerencie a inclusão e remoção de processos que ocupam o processador. A carga de processos deverá ser realizada a partir de programas que utilizarão uma linguagem assembly hipotética.

Descrição de programas e características de execução e ocupação

O usuário deverá ser capaz de descrever pequenos programas a serem executados pelo ambiente. O ambiente de execução é baseado em acumulador. Assim, para a execução de um programa, dois registradores estão presentes: ***(i)*** o acumulador (acc) onde as operações são realizadas e ***(ii)*** o ponteiro da instrução em execução (pc). A linguagem de programação hipotética a ser empregada pelo usuário para a programação e que manipula os dois registradores descritos é apresentada na Tabela 1. Ali são definidos em quatro categorias, conforme listado na primeira coluna.

Tabela 1 - Mnemônicos e funções associadas

Categoria	Mnemônico	Função
Aritmético	ADD op1	acc=acc+(op1)
	SUB op1	acc=acc–(op1)
	MULT op1	acc=acc*(op1)
	DIV op1	acc=acc/(op1)
Memória	LOAD op1	acc=(op1)
	STORE op1	(op1)=acc
Salto	BRANY label	pc ← label
	BRPOS label	Se acc > 0 então pc <- op1
	BRZERO label	Se acc = 0 então pc <- op1
	BRNEG label	Se acc < 0 então pc <- op1
Sistema	SYSCALL index	Chamada de sistema

As instruções da categoria aritmético podem operar em modo de endereçamento direto ou imediato com a memória. No modo de endereçamento imediato, o valor de *op1* pode ser o valor real a ser considerado na operação. No Modo de endereçamento diretor, *op1* representa a variável na área de dados cujo valor deve ser capturado. A diferenciação entre os dois modos no código assembly a ser gerado deve ser dado pela presença do caractere sustenido (#) em frente ao operador *op1*, a fim de representar o modo imediato, ou a ausência deste caractere, a fim de representar o modo direto. Um exemplo desta situação é ilustrado na linha 3 no código de exemplo da Figura 1.

Duas instruções são utilizadas na categoria memória, e representam a leitura ou a escrita de dados em memória de dados. Assim como assumido para as instruções aritméticas, a instrução de leitura da memória de dados pode ser realizada tanto em modo imediato quanto direto, e a interpretação segue tal como descrito anteriormente. Um exemplo pode ser visto na Figura 1, na linha 2. Já o comando de escrita (i.e. STORE) dá suporte apenas ao modo direto de operação, ou seja, um dado somente pode ser escrito em uma variável declarada na área de dados.

Para os mnemônicos referentes a saltos condicionais, label representa a posição da área de código que deve ser assumida por PC. Há quatro tipos de saltos, sendo um incondicional e três condicionais, assumindo-se então as condições destacadas na Tabela 1. Para a criação de *labels* utilize a representação de um "nome" alfanumérico, seguido de dois pontos (:), conforme ilustrado na Figura 1, na linha 3. Um exemplo de uso de uma instrução de salto condicional é apresentado na mesma figura, na linha 5.

A última categoria da tabela 1 representa a instrução de operação com o sistema. Deverá ser possível a associação de 3 tipos de pedidos de operação com o sistema, representados pelos valores numéricos '0', '1' e '2'. Uma chamada de sistema referenciando o valor '0' (zero) caracteriza um pedido de finalização/encerramento do programa, tal como um *halt*. Uma chamada de sistema referenciando o valor '1' (um) caracteriza um pedido de impressão de um valor inteiro na tela. Uma chama de sistema referenciando o valor '2' (dois) caracteriza um pedido de leitura de um valor inteiro, que deverá ser feito via teclado. Às chamadas de sistema cujos valores '1' e '2' forem

atribuídos, deverá ser associada uma situação de bloqueio deste processo, com um valor aleatório entre 8 e 10 unidades de tempo.

```
1  .code              #Define o início da área de código
2    LOAD variable    # Carrega em acc o conteúdo de variable
3    ponto1: SUB #1   # Subtrai do acc um valor constante (i.e. 1)
4    SYSCALL 1        # Imprime na tela o conteúdo do acumulador
5    BRPOS ponto1     # Caso acc>0 deve voltar à linha marcada por "ponto1"
6    SYSCALL 0        # Sinaliza o fim do programa
7  .endcode           #Define o final da área de código

8  .data              #Define o início da área de dados
9    Variable 10      # Conteúdo da posição 1 da área de dados é 10
10 .enddata           #Define o final da área de dados
```

Figura 1 - Código exemplo.

Como finalização da descrição dos processos, devem ser assumidas as seguintes características.

- Ocupação de memória
  - Cada instrução ocupa uma posição da memória, independente de sua categoria;
  - Cada variável ocupa uma posição da memória;
  - O número de posições de um programa é então definido pelo número de instruções somado ao número de variáveis;
  - A organização de ocupação da memória por parte dos processos está fora do escopo deste trabalho;
- Os valores atribuídos às variáveis ou às instruções que operam de modo imediato podem assumir valores tanto positivos quanto negativos

Políticas de escalonamento e modelo de estados

Como política de escalonamento entre processos, deve ser possível escolher entre duas políticas, antes do início da operação: **Round Robin (RR) com *quantum* definível e com prioridade**, e **Shortest-Job-First (SJF) com preempção.** Informações adicionais sobre políticas de escalonamento estão disponíveis no moodle da disciplina e no capítulo 6 do livro texto da disciplina, caso necessário.

- RR

Deve-se assumir 3 níveis de prioridade: baixa prioridade (2), prioridade média (1) e alta prioridade (0). No momento da carga do processo, deve-se poder alterar a prioridade, que por padrão deverá ser baixa. Uma vez carregado o processo, este não pode mais sofrer alteração de prioridade (prioridade estática).

O tempo de ocupação de cada processo no processador é um parâmetro que poderá ser definido quando do início da operação (*quantum)*. Uma vez carregado o processo na lista de pronto, o *quantum* do processo não poderá mais ser alterado. Os processos podem ter diferentes valores de *quantum*.

Os processos são armazenados na fila de prontos, ordenados de acordo com a sua prioridade. Caso existam outros processos com a mesma prioridade na fila de prontos, o processo deve ser inserido ao final da sequência de processos com a mesma prioridade. A ordem do próximo processo a assumir o processador é dada pela prioridade, seguindo o ordenamento da fila de processos prontos. O algoritmo, a cada intervalo de tempo, interrompe o processador, reavalia as prioridades, e decide qual processo deverá ocupar o processador no próximo instante de tempo.

- SJF

Este algoritmo de escalonamento associa cada processo ao seu tempo de execução. Quando o processador está livre, aquele processo que ocupar o menor tempo de processamento para terminar a sua execução será selecionado para ocupar o processador. Caso exista um processo com tempo de execução menor que o tempo de execução do processo que está ocupando o processador, o escalonador interrompe o processo que está em execução e escalona o processo com o tempo de execução menor.

Um processo somente deixa o estado de ***running*** quando: ***(i)*** o processo tem seu ***timeout*** alcançado pelo quantum, para o caso do RR; ***(ii)*** existe um processo de mais alta prioridade pronto para ser executado, para o caso do RR; ***(iii)*** na fila de processo prontos para execução tenha um processo com tempo de execução menor; para o caso do SJF, ou ***(iv)*** realiza uma chamada de sistema, para ambos algoritmos. Nos três primeiros casos, o processo volta para o final da lista de pronto. No último caso, diferentes estados podem ser alcançados. No caso de um pedido de encerramento do processo, este deve avançar para o estado de ***exit*** e o espaço que este processo ocupava na memória deve ser liberado. Caso o pedido seja de uma impressão ou leitura a partir da chamada do sistema, este processo deverá assumir o estado de bloqueado e um intervalo de tempo de permanência neste estado deverá ser assumido (aleatório entre 8 e 10 unidades de tempo). Passado este tempo, o processo pode avançar para o estado de ***pronto***.

Para os algoritmos de escalonamento implementados, deve-se assumir que, supondo que um processo A venha a ser removido do processador para que um processo B o ocupe, quando da retomada do processo A, este deve seguir sua execução do último ponto de parada. O tempo necessário para troca de contexto deve ser desconsiderado para o presente trabalho.

Interface da aplicação

O ambiente a ser desenvolvido deve permitir definir qual(is) programa(s) será (serão) carregado(s) e o(s) instante(s) de carga (*arrival time*). Para o algoritmo RR, além do instante de carga, a interface deve permitir definir a prioridade e o *quantum* para cada processo. Já para o algoritmo SJF, além do instante de carga, a interface deve permitir definir o tempo de execução. Considere que os espaços necessários para o controle dos processos, tal como os valores de PC, ACC e o estado em que se encontra, fazem parte de um espaço reservado para o sistema operacional. Como resultado da operação com os programas, deve-se permitir avaliar o *waiting time*, o *processing time* e o *turnaround time* de cada um dos processos finalizados, bem como o tempo de permanência em cada um dos estados durante sua existência para o sistema. Considere como *waiting time*, todos os instantes em que o processo esteve pronto para ser executado mas que não conseguiu assumir o processador. Processing time como o tempo em que ficou em estado de *running* e o turnarround time como o tempo desde sua criação até sua finalização. Adicionalmente, deve-se poder observar os processos que estão nos diferentes estados (pronto, executando, bloqueado, finalizado).

Informações adicionais

O trabalho deverá ser realizado em grupos de 4 alunos (obrigatoriamente). Deverá ser entregue o código fonte do programa desenvolvido, bem como um manual do usuário em formato PDF contendo as explicações de como compilar e executar o programa. A linguagem de programação utilizada para desenvolver o trabalho é de escolha do grupo, desde que seja possível compilar e executar o programa no ambiente computacional disponível nas salas de aula laboratório do prédio 32.

A data de entrega está prevista para o dia 20/04/2023, até às 17hs15min. As apresentações serão realizadas pelos alunos a partir do material postado no Moodle.

Cada aluno deverá assinalar no Moodle o grupo ao qual pertence e **um integrante** do grupo deverá escolher **um horário** para o grupo apresentar o trabalho no dia 20/04/2023 ou no dia 25/04/2023, dentre os horários disponíveis no Moodle.

O trabalho deverá ser entregue no moodle a partir de um arquivo compactado (.zip ou .rar). O nome do arquivo deverá ser tal que contenha o nome e sobrenome de todos integrantes do grupo. O material postado no moodle é de inteira responsabilidade do aluno. A presença de arquivos corrompidos, que impeçam a avaliação do trabalho pelo professor será considerada como a não entrega do trabalho. Também não serão considerados trabalhos com erro de compilação. Casos onde sejam identificados plágio/cópia, receberão nota zero.