PUCRS/FACIN – Sistemas Operacionais (SI) – 2014/01 – Profa. Alexandra Aguiar

Projeto de execução dinâmica de processos - Etapa 1 - Trabalho Prático

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 políticas de escalonamento específicas, bem como gerencie a inclusão e remoção de processos que ocupam o processador. A carga de processos deverá ser feita 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 = label
	BRZERO label	Se acc == 0 então pc = label
	BRNEG label	Se acc < 0 então pc = label
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 direto, *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 dessa 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 apresentada na mesma figura, na linha 5.

A última categoria daquela tabela 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', '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 chamada 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 10 e 40 unidades de tempo de duração.

```
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. A memória precisa ser endereçável de maneira precisa através de índices;
  - 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 desta etapa do trabalho e deve ser feita de forma sequencial (o programa tem que caber inteiro em uma dada parte da memória);
- Os valores atribuídos às variáveis ou às instruções que operam de modo imediato podem assumir valores tanto positivos quanto negativos

Política de escalonamento e modelo de estados

Como política de escalonamento entre processos, deve ser possível a utilização de Round Robin com *quantum* definido no início da simulação OU Shortest Job First preemptivo. No caso de se optar por SJF, o processing time (PT) vem da pré-análise do arquivo assembly, indicando quantas instruções são necessárias. Observe que, PT é uma estimativa (baseada na contagem de instruções) e não necessariamente vai representar o tempo total que um processo irá executar (uma vez que podem haver laços). No entanto, para a métrica final de TT (turnaround time) deve-se levar em consideração o tempo de processamento real utilizado. As políticas não são cumulativas, ou seja, deve-se escolher uma delas no início da simulação. A mudança de estados de cada processo deverá seguir o modelo de 7 estados similar àquele visto em sala de aula e ilustrado na Figura 2.

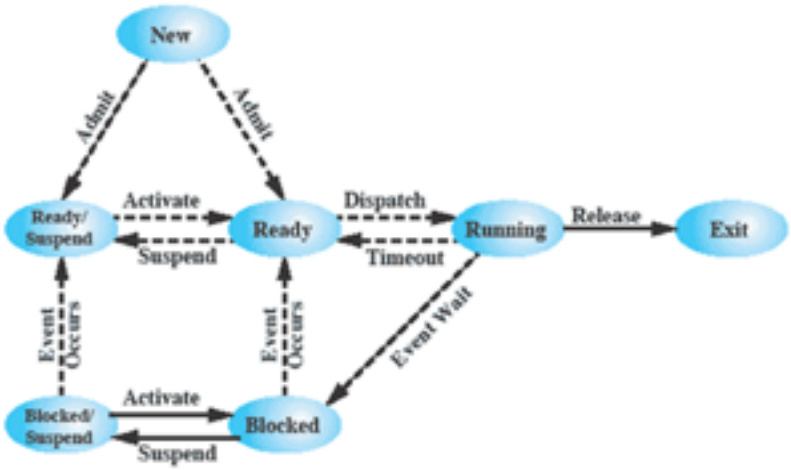

Figura 2 - Estados de um processo

Quando da criação de um processo, este pode ocupar a fila de ***pronto*** sempre que houver espaço na memória. O estado de ***suspenso mas pronto*** deverá ser assumido por um processo em três casos: ***(i)*** quando de sua criação não há qual(is)quer processo(s) que possa(m) ser suspenso(s), a partir dos estados possíveis conforme ilustrado na Figura 2 ou ***(ii)*** estando na fila de ***pronto***, a suspensão do(s) processo(s) de menor prioridade garante espaço para o processo de maior prioridade que está sendo criado ou ***(iii)*** estando na fila de ***bloqueado e suspenso*** o evento que aguardava é alcançado.

O algoritmo *Round Robin* terá o comportamento conforme apresentado em sala de aula. Este algoritmo levará em consideração tão somente os processos na lista de pronto para execução. O tempo máximo de ocupação de cada processo no processador é um parâmetro que deverá ser definido quando do início da operação. Uma vez carregado o primeiro programa na lista de pronto, o *quantum* não poderá mais ser alterado. 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.

Um processo somente deixa seu estado de ***running*** sob duas condições, quais sejam: ***(i)*** o processo tem seu ***timeout*** alcançado pelo quantum ou ***(ii)*** realiza uma chamada de sistema. No primeiro caso, o processo volta para o final de sua lista de pronto, conforme mencionado anteriormente. No segundo 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 então ser liberado.Caso o pedido seja de uma impressão ou leitura a partir da chamada de sistema, este processo deverá assumir o estado de bloqueado e um intervalo de tempo de permanência neste estado deverá ser assumido (aleatório entre 10 e 40 unidades de tempo). Passado este tempo, o processo pode avançar para o estado de ***pronto***.

Interface da aplicação

O ambiente a ser desenvolvido deve permitir definir qual(is) programa(s) será (serão) carregado(s), o(s) instante(s) de carga (*arrival time*), a prioridade como alta média ou baixa (para auxiliar no caso de suspensão). Outro parâmetro que deverá ser informado é a quantidade de posições de memória que estarão disponíveis para os processos. 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 e que não é contabilizado. 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 finalizado, bem como o tempo de permanência em cada um dos estados durante sua existência para o sistema. Deve-se observar os registradores e o comportamento geral dos programas. Considere como *waiting time*, todos os instantes em que o processo esteve pronto para ser executado (ou *suspenso mas pronto*) mas que não conseguiu assumir o processador; *processing time* como o tempo em que ficou em estado de *running,* e; *turnarround time* como o tempo desde sua criação até sua finalização.

Informações adicionais

O trabalho deverá ser realizado em grupos de até quatro componentes e será avaliado na aula combinada, sendo entregue via Moodle até o início da aula de apresentações por um dos componentes do grupo. O trabalho deverá ser submetido como um arquivo compactado, juntamente a um relatório de perguntas guia e de auto-avaliação do grupo, tendo como nome, a junção do nome dos alunos segundo o seguinte formato: "Nome1Sobrenome1_Nome2Sobrenome2.zip". Não serão aceitos trabalhos entregues fora do prazo. Trabalhos que não compilam ou que não executam não serão avaliados. Todos os trabalhos serão analisados e comparados. A nota do trabalho depende da apresentação no laboratório. Trabalhos entregues mas não apresentados terão sua nota anulada automaticamente. Caso seja identificada cópia de trabalhos, TODOS os trabalhos envolvidos receberão nota ZERO. Deve ser adicionado no zip um arquivo README para a execução do trabalho. Os itens para avaliação são: (i) funcionamento do simulador, (ii) execução do processo de escalonamento, (iii) saída do simulador (métricas dos processos executados); (iv) clareza do código (utilização de comentários e nomes de variáveis adequadas), (v) interface e usabilidade, (vi) relatórios.