Dicas para simulações nas atividades de projeto de Sistemas Operacionais

ADMINISTRAÇÃO DE MEMÓRIA

Metas da administração de memória

- Modelagem da memória e da sua administração
- Modelagem do uso da memória pelos programas
- Simulação da execução de programas (reprodução do comportamento individual dos programas)
- Simulação descritiva dos efeitos da execução dos programas (avaliação dos parâmetros descritivos do comportamento dos programas e do sistema)
- Ensaiar algoritmos e políticas de alocação, remoção, compactação e virtualização de memória
- Avaliar a influência das técnicas de administração de memória no desempenho dos programas.

Seleção do tipo de administração de memória

- Determinar o tipo de administração de memória que melhor se aplica aos objetivos do seu projeto:
  - Contígua simples
  - Particionada simples e relocável
  - Overlay
  - Paginada, simples e virtual
  - Segmentada, simples e virtual
  - Segmentada com paginação, simples e virtual

Baseada em partições

- Particionada, overlay e segmentada:
  - Modelar a memória como um vetor em que cada elemento corresponde a uma unidade de alocação adotada (por ex., alocar múltiplos de 1, 4, 8 ou 16 Kbytes)
  - Uma informação indicativa da ocupação dessa área de memória (por ex., um código para indicar vazio, ou a identificação do ocupante em caso contrário)
  - Uma informação sobre o tipo de elemento ocupante: código executável, área de leitura, área de leitura e gravação, etc.
  - Para cada programa, montar uma segment map table, e para cada segmento, uma page map table se for o caso.
  - Uma rotina de garbage collection (opcional)

Baseada em páginas

- Paginada:
  - Modelar a memória como um vetor, usando um elemento do vetor para simbolizar cada bloco de memória
  - Montar uma lista ligada de blocos vazios, e para cada programa, uma lista para os blocos alocados
  - Para cada programa, montar uma page map table

Memória Virtual

- Virtual
  - A simulação da execução de programas reais exige um processador cuja arquitetura suporte memória virtual
  - Montar um motor de eventos acionado pelas referências a páginas ou segmentos referenciados que não tenham sido alocados na memória.
  - O loader deve ser acionado para alocar áreas de memória adequadas para a transferência desejada.
  - Rotinas de leitura devem promover sua transferência do disco para a memória.
  - As tabelas de mapeamento de páginas ou de segmentos devem ser atualizadas
  - Rotinas de substituição de página ou de segmento devem ser acionadas sempre que for necessário.
  - Selecionar uma política simples, à sua escolha.

ADMINISTRAÇÃO DE PROCESSOS

Para simular o administrador de processos

- Estabelecer um diagrama de estados representativo dos diversos estados que os programas podem assumir durante sua permanência no sistema
- Associar a cada estado do diagrama: (i) uma fila de espera do recurso correspondente ao estado, e (ii) um evento cujo tratamento altera o estado do programa, removendo-o desta fila e inserindo-o em outra.
- Para cada programa, montar um process control block que descreva exatamente seus parâmetros e dados contabilizados durante sua execução.
- Esses process control blocks serão os elementos que irão ocupar as filas de espera do sistema, ou seja, dos estados do diagrama acima.

Prioridades

- O uso de critérios de desempate no caso de conflitos passa, geralmente, pelo estabelecimento de prioridades para os programas.
- A associação entre os programas e suas prioridades é inevitavelmente arbitrária, e tende a favorecer os programas mais prioritários.
- Algoritmos podem ser utilizados para, segundo seus critérios (também arbitrários), alterar prioridades já estabelecidas. Essa opção pode tornar menos tendenciosas essas escolhas de prioridades.

Multiprogramação

- É uma exigência em todos os sistemas operacionais em uso, e portanto deve ser incluída no projeto.
- Para torná-la flexível, deve ser parametrizada segundo o número máximo de programas simultâneos admissível no sistema.
- Usando grau 1 de multiprogramação, simula-se a uniprogramação.
- Uma fila de entrada deve ser criada para manter em espera os programas que já estejam no sistema, porém ainda não tenham sido aceitos na multiprogramação.
- Políticas podem ser escolhidas para alterar a ordem de ingresso dos programas à multiprogramação.
- O uso de fila circular, ordenada segundo a sequência de chegada (round-robin), é uma opção bastante aceita na prática.
- Tipicamente, estabelece-se um quantum fixo, intervalo máximo de tempo para o uso contínuo do processador. Após esse intervalo, o programa deve ceder o processador ao próximo da fila round-robin, e será reinserido ao final dessa fila, para aguardar sua vez de receber mais um quantum de tempo de processador.

- Construa vários conjuntos de programas, cada qual com um perfil diferente e significativo, cujos programas tenham percentuais de tempos de espera de entrada/saída parecidos ou bem diferentes, para testar a influência da homogeneidade dos programas no desempenho da multiprogramação (verifique o comportamento simulado com diversos graus máximos de multiprogramação):
  - Programas muito similares entre si, sem entrada/saída
  - Idem, com diversos percentuais de tempo de espera
  - Programas similares entre si, com muita entrada/saída
  - Programas bastante diversificados

ADMINISTRAÇÃO DE DISPOSITIVOS

- Para cada dispositivo a ser simulado no sistema, crie um device control block, que deverá conter todas as informações relevantes sobre o dispositivo: tempos de acesso e de resposta, capacidade máxima, eventos e interrupções associados, número de recursos que o dispositivo permite utilizar ao mesmo tempo, etc.
- Construa rotinas de tratamento para os eventos (interrupções) associados ao dispositivo, para simular o driver do dispositivo que se está simulando.
- Escolha o dispositivo físico do computador hospedeiro no qual as entradas ou saídas do dispositivo simulado deverão acontecer fisicamente.

- Virtualização
  - Caso o dispositivo deva ser virtualizado, é necessário eleger um dispositivo do computador hospedeiro no qual ele deve ser espelhado.
  - Rotinas especiais devem ser construídas para recolherem dados lidos no dispositivo hospedeiro, guardá-los até o momento oportuno, para então transferi-los para o dispositivo simulado, e vice-versa.
  - É importante não esquecer que a virtualização deve ser transparente ao programador, constituindo um procedimento interno ao sistema, que não precisa ser do conhecimento do programador nem do programa.

- Mecanismos de acionamento físico dos dispositivos
  - Disparo de um processo de entrada/saída: (i) no caso de simulação das instruções do programa, pela execução de uma chamada de supervisor, solicitando um serviço de entrada/saída ao sistema. (ii) no caso de simulação estatística, sorteando o instante da próxima solicitação de entrada/saída a ocorrer durante a utilização do processador.
  - Determinação do instante do término do atendimento à solicitação do serviço de entrada/saída: com base nos parâmetros disponíveis no device control block do dispositivo, programar o evento de fim de entrada/saída para o instante futuro correspondente ao tempo de resposta do dispositivo à operação realizada

- Simulação da entrada/saída em dispositivos dedicados
  - Dispositivos dedicados devem bloquear novos programas que solicitem sua utilização, impedindo-os de prosseguir enquanto o programa que os esteja utilizando estiver ativo.
  - No caso da simulação da execução de programas em máquina virtual, os dados manipulados pelo programa em execução devem ser devidamente movimentados de ou para os dispositivos da máquina hospedeira que os representam.
  - No caso de simulações estatísticas, as operações correspondentes a essas solicitações devem ser contabilizadas, e um log deve ser gerado para registro das atividades simuladas.

- Simulação de entrada/saída em dispositivo compartilhado (tais como discos e pen-drives)
  - Dispositivos compartilhados devem ser gerenciados em dois níveis: (i) nível do dispositivo físico, através de transferências de dados de/para posições físicas da mídia; (ii) nível do dispositivo lógico, mediante a administração individual dos arquivos como se cada um deles fosse um dispositivo separado.
  - Assim, quando se deseja acessar um arquivo, é necessário solicitar inicialmente o dispositivo lógico correspondente, e de posse desse arquivo, pode-se então solicitar acesso às posições físicas da mídia a ele correspondentes, e aí efetuar as devidas transferências.

- Simulação de entrada/saída em dispositivos virtuais
  - Essa situação nada mais é que uma simulação que ocorre dentro de outra simulação, pois um dispositivo virtual é um dispositivo simulado.
  - Para se efetivarem, as entradas e saídas em dispositivos virtuais precisam ser simuladas em algum dispositivo real (em geral, o disco) do computador hospedeiro da simulação
  - Para que os resultados de entradas e saídas virtuais, feitas em disco, alcancem o programa e o seu usuário, precisam ser convertidas em entradas e saídas reais em algum dispositivo dedicado da máquina hospedeira.
  - Mecanismos de spooling, que também são virtualizações de dispositivos, podem ser tratados e implementados exatamente da mesma maneira.

ADMINISTRAÇÃO DE INFORMAÇÃO

- O administrador de informação não precisa ser complexo, bastando aqui que o disco simulado seja implementado em uma pasta do disco hospedeiro.
- O sistema de arquivos a ser implementado é simplesmente uma lista dos arquivos presentes nessa pasta, que representa o sistema simulado.
- No caso mais simples, os arquivos contidos nessa pasta, que representa o disco do sistema simulado, não são organizados de forma complexa, bastando que tenham nomes diferentes uns dos outros.
- A alocação dos arquivos no disco simulado pode ser feita usando o mesmo tipo de administrador de espaço que tenha sido utilizado para administrar a memória. Uma boa opção é a alocação particionada.

Especificação da linguagem de controle

LINGUAGEM DE SCRIPT PARA CONTROLE DO SHELL DE UM SISTEMA OPERACIONAL

Interface entre o sistema e o usuário

- É usual chamar de *shell* (invólucro, casca) de um sistema o programa que implementa a interface entre esse sistema e os seus usuários.
- As formas mais comuns de *shells* para sistemas computacionais se materializam como interfaces textuais ou então como interfaces gráficas.
- Um mesmo sistema de programação pode apresentar compatibilidade com diversos *shells*, permitindo assim que seu usuário escolha o tipo de interface com a qual possa estar mais à vontade.
- Esta é a especificação de um *shell* muito simples, do tipo textual, cuja construção é enunciada no material apresentado a seguir, através da proposta de uma linguagem minimalista de script voltada ao controle da operação dos componentes de um sistema de programação.

Linguagem de controle do Sistema Operacional

- A linguagem de controle mínima, já implementada, da interface textual do Sistema Operacional, contempla os seguintes comandos:
  - **Iniciação da operação do sistema**
    - Login, preparação para início de funcionamento do sistema **(job)**
  - **Manipulação de arquivos de programas e dados, em disco**
    - Criar novo arquivo **(create)**, apagar arquivo **(delete),** mostrar conteúdo de um arquivo **(list),** apresentar os arquivos existentes **(directory).**
    - Designar arquivo como mídia de entrada **(infile)** ou saída **(outfile)** do programa a executar;
    - Designar pasta do hospedeiro p/simular disco **(disk)**
    - Designar arquivo como meio de armazenamento do programa a executar **(diskfile)**
  - **Execução de programas (run)**
    - Programas do sistema de programação (simulador MVN, loader absoluto, dumper absoluto, montadores absoluto e relocável, ligador e relocador)
    - Programas do usuário: programa desenvolvidos em código objeto absoluto
  - **Encerramento da operação do sistema**
    - Finalização de programas pendentes e do funcionamento do sistema **(endjob)**

Os comandos do *shell*

comando	formato	ação
Inicia operação	**$JOB nome**	Login - inicia um novo job, reinicia todo o sistema
Escolhe pasta	**$DISK pasta**	Simula disco do sistema nesta pasta do hospedeiro
Arquivos existentes	**$DIRECTORY**	Lista o conteúdo da pasta do hospedeiro
Cria arquivo	**$CREATE nome**	Cria no disco um novo arquivo, se ainda não existir
Apaga arquivo	**$DELETE nome**	Remove do disco o arquivo indicado, se existir
Mostra conteúdo	**$LIST nome**	Apresenta o conteúdo do arquivo indicado, se existir
Mídia de entrada	**$INFILE nome**	Adota o arquivo indicado como a fita de entrada
Mídia de saída	**$OUTFILE nome**	Adota o arquivo indicado como a fita de saída
Arquivo em disco	**$DISKFILE nome**	Adota o arquivo indicado como arquivo em disco
Executa programa	**$RUN nome**	Executa o programa indicado (de sistema ou ususário)
Encerra operação	**$ENDJOB nome**	Finaliza pendências e termina o job corrente

Para facilitar a implementação, o comprimento das palavras-chave e dos nomes e pastas utilizados nesses comandos pode ser limitado a um número máximo de caracteres.

Interpretação da linguagem de comandos

formato
$JOB nome
$DISK pasta
$DIRECTORY
$CREATE nome
$DELETE nome
$LIST nome
$INFILE nome
$OUTFILE nome
$DISKFILE nome
$RUN nome
$ENDJOB nome

- O extrator de sequências de caracteres deve separar e classificar as palavras-chave, precedidas de um sinal $ na tabela ao lado.
- Se for encontrada alguma palavra fora desse repertório, o suposto comando deve ser ignorado, devendo além disso ser emitida uma mensagem de erro.
- Caso contrário, deve ser enviado um evento, correspondente ao comando encontrado, para que seja devidamente tratado por sua rotina de interpretação.

A primeira ativação do sistema

- Para dar partida no sistema de programação deste projeto, aciona-se, no sistema hospedeiro (S.O. instalado no hardware em que você está desenvolvendo o projeto), o programa interpretador dos comandos de controle, que por ora faz o papel, ainda precário, de S.O. no nosso sistema de programação.
- Ao ser acionado, o interpretador deve forçar um diálogo de **login**, solicitando ao usuário que se identifique, e utilizando essa identificação para associar ao usuário uma pasta única onde estarão guardados os seus arquivos na máquina hospedeira.
- Tal interpretador, através dos seus comandos, disponibiliza ao usuário, na máquina hospedeira, os programas responsáveis pela efetiva execução de todas as funções do sistema de programação.
- Isto se dá executando-as diretamente no hardware, na forma de código da máquina hospedeira, ou então interpretando-as, na forma de código de máquina virtual, em um simulador desta (naturalmente, o último nível de simulação se executa sempre na máquina hospedeira).

Iniciação e finalização do sistema

- A função de iniciação é essencial pois representa o meio através do qual a máquina virtual e seu sistema de programação interagem com a máquina hospedeira.
- Iniciar o sistema corresponde a ativar um programa da máquina hospedeira que contém toda a lógica implementada para viabilizar a realização das funcionalidades de controle e executivas do sistema de programação da máquina virtual.
- Finalizar o sistema, por outro lado, encerra esse vínculo com o sistema hospedeiro, liberando-o para executar outras atividades.
- No nosso caso, a função de iniciação é executada diretamente na ocasião da primeira ativação do programa do sistema hospedeiro que implementa a simulação das funcionalidades do sistema de programação.
- Nessa oportunidade, o sistema exige que seja feito um **login** pelo usuário, que deve identificar-se para efeito de organização dos arquivos mantidos pelo sistema de programação na máquina hospedeira.
- Analogamente, a finalização se dá quando o sistema de programação recebe um comando de controle que solicita o encerramento das atividades.

Linguagem de comandos

- O sistema hospedeiro deverá, naturalmente, abrigar um interpretador de linguagem de comandos, responsável por disponibilizar ao usuário um meio para comunicar-se com o sistema de programação.
- A linguagem de comandos proposta consta do seguinte conjunto mínimo de comandos acionáveis através de uma janela de diálogo:
  - **$DIR** – informa os programas do usuário disponíveis no sistema
  - **$DEL** – remove do sistema um programa do usuário
  - **$RUN** – inicia a execução de um programa do usuário
  - **$END** – encerra operação do sistema e reporta informações coletadas
- Através do interpretador da linguagem de comandos, o sistema hospedeiro simulará a máquina virtual, que "executará" o código de máquina dos programas do usuário.
- Apresenta-se a seguir a especificação funcional dos comandos desta pequena linguagem.

Comando $DIR

- Formato: **$DIR**
- Exibe como saída a lista de nomes dos programas do usuário correntemente disponíveis no sistema.
- Para isto, é necessário que o interpretador de comandos do sistema de programação mantenha atualizada uma tabela com todos os nomes dos programas de usuário prontos para serem executados no sistema.
- Embora os sistemas reais mantenham, em formato binário, os programas executáveis em arquivos do sistema operacional hospedeiro, sugere-se, por simplicidade, que neste projeto o código objeto desses programas seja mantido em codificação ASCII hexadecimal, no formato **.txt**, para que seu conteúdo possa ser facilmente visualizado quando necessário.

Comando $DEL

- Formato: **$DEL <nome>**
- Remove do sistema de programação o acesso ao programa cujo nome é fornecido como parâmetro do comando.
- Para isso, pesquisa-se o nome fornecido na tabela de nomes de programas do usuário, e caso o nome não seja encontrado, acusa-se a detecção de um erro, caso contrário, remove-se o elemento correspondente da tabela de nomes.
- Sugere-se que o arquivo **.txt** correspondente ao programa removido seja mantido no sistema de arquivos da máquina hospedeira, com o nome estrategicamente modificado (por exemplo, adicionando-se ao nome original um dígito e/ou um caractere especial) para facilitar a identificação e a remoção definitiva posterior dos arquivos desnecessários.

Comando $RUN

- Formato: **$RUN <nome>**
- Dispara a execução do simulador da máquina hospedeira, carrega o loader e promove sua execução para que seja feita a carga do código do programa cujo nome é fornecido como parâmetro. O loader deve obter no código do programa a informação do seu endereço de execução. O interpretador de comandos deve forçar o simulador a desviar para esse endereço antes de prosseguir executando o programa.
- Programas bem construídos devem executar, como última instrução, uma chamada de sistema operacional dando-lhe conhecimento do término de sua execução, e promovendo a devolução do controle ao sistema, no caso representado pelo interpretador de comandos do sistema de programação.

Comando $END

- Formato: **$END**
- Através deste comando o interpretador de comandos é informado de que o usuário não deseja mais continuar utilizando o sistema de programação. Assim, o sistema deve apagar definitivamente do sistema de arquivos da máquina hospedeira todos os arquivos **.txt** referentes a programas do usuário marcados como removidos, e, antes de encerrar suas atividades, listar para o usuário todas as demais informações consideradas úteis.
- Programas de usuário não removidos podem ser mantidos no sistema de arquivos hospedeiro em uma pasta associada ao usuário seu proprietário, para que possam ser utilizados em uma sessão futura de execução do sistema de programação. Como não existe ainda um sistema operacional na máquina virtual, a manutenção desses arquivos pode ser feita pelo próprio usuário externamente, por meio de comandos do sistema operacional hospedeiro.

Comandos a serem adicionados

- Complete a linguagem de comandos com os seguintes comandos adicionais:
  - **$LOGIN/$LOGOUT** – entrar/sair legalmente do sistema
  - **$PROG/$EPROG** – delimita as ações de um programa
  - **$PROC** – especifica um tempo de processamento
  - **$IN/$OUT** – entrada/saída sequencial em dispositivo
  - **$OPEN/$CLOSE** – abrir/fechar arquivo sequencial
  - **$READ/$WRITE** – entrada/saída sequencial em arquivo
- Especifique cada um desses comandos, e incorpore-os aos demais apresentados anteriormente. Implemente o interpretador dessa linguagem e incorpore-o ao seu sistema de programação.