SOPA **Versão 1.1 – 18/04/2012**

Sistema Operacional Para Avaliação - Definição

O trabalho proposto é a implementação de um simulador de Sistema Operacional com programação concorrente para uma **MÁ-QUINA**. O processador deve executar programas de usuário em código objeto próprio, e executar o sistema operacional **SOPA** em código de alto nível, o mesmo usado no simulador da Má-Quina. Isto significa que o sistema operacional será uma rotina chamada pelo processador da Má-Quina, representando o que a CPU faz quando executa o S.O. Mesmo assim o *kernel* de SOPA deve ser programado como um objeto distinto, encapsulando as suas estruturas próprias, não acessíveis à Má-Quina. Os demais componentes de *hardware* também serão modelados como objetos distintos, sendo alguns deles, especiais, de funcionamento independente da CPU, modelados com *threads*. Na versão deste semestre o simulador deve suportar dois discos, duas CPUs, e ter uma interface gráfica que desenha um diagrama de tempo semelhante àqueles encontrados em livros didáticos de sistemas operacionais, onde se observa os processos que tomam a CPU ou usam o disco ao longo do tempo.

Os componentes de cada máquina são os seguintes:

Controlador de Interrupções: Objeto escravo, sinalizado pelos demais componentes, sincronizado, com fila para enfileirar as requisições de interrupção de *hardware*;
Memória: Objeto escravo sincronizado, acessado pelo processador e pelo disco, para DMA;
Discos: *Threads* que modelam dois discos com suas controladoras. Na versão atual, elas recebem comandos para ler ou gravar em uma posição determinada, endereço físico de disco. O kernel é responsável por controlar os acessos e implementar a idéia de arquivos, que podem ter operações de *open* e *close*.
Console: *Thread* que simula um terminal de mais alto nível, que somente gera interrupção de *hardware* após o usuário entrar com uma linha inteira de comando, que será de carga e execução de programas objeto, indicando antes de qual disco origem;
Timer: *Thread* representando relógio fixo de tempo que gera interrupções de *hardware*;
Processadores: *Threads* CPU, que executam programas objetos e o sistema operacional. Após cada instrução, uma CPU verifica o Controlador de Interrupções para saber se deve chamar o *kernel*. As interrupções de *software* são testadas antes, no ciclo normal, e não usam o mesmo objeto Controlador de Interrupções para se sincronizarem;
Kernel: Objeto escravo do Processador que implementa o núcleo do SOPA, com suas rotinas e estruturas de dados (possivelmente outros objetos). O código deve ser reentrante para rodar simultaneamente nos dois processadores, quando for necessário.

Sistema Operacional Para Avaliação – Gerência de Memória

A gerência de memória no SOPA usará a técnica de partições fixas. A memória da Má-Quina tem 1024 palavras nesta versão, e será dividida em 8 partições de 128 palavras. A MMU deve ser implementada no objeto escravo memória, e usará um registrador base e um

registrador limite para efetuar proteção. Acessos ilegais devem gerar interrupções de *hardware* de tipo específico. A memória deve ter também alguns métodos privilegiados de acesso sem proteção, que serão usados pelo *kernel*, que supostamente roda em modo supervisor, e também pelo disco, que pode fazer DMA.

Sistema Operacional Para Avaliação – Gerência de Entrada e Saída

A gerência de Entrada e Saída envolve console e discos, mas é implementada no *kerne*l. O console deve ler linhas de comando que indicam a execução de um programa a partir de um dos dois discos da Má-Quina. O formato de uma linha deve ser o seguinte:

```
<número do disco> <posição inicial do arquivo programa>
```

Os arquivos são designados, tanto no console como nas operações de disco, por posições do disco, ou seja, endereços físicos de disco. Nenhum sistema de arquivos sofisticado precisa ser implementado. O simulador interpretará como arquivo a seqüência de palavras desde uma posição inicial indicada até uma marca de fim de arquivo que será a palavra 255 255 255 255. O formato do arquivo de programa é definido na Má-Quina. O formato de arquivo de dados segue o mesmo padrão. O arquivo é um conjunto de palavras, separadas por marcas de fim de linha, e cada palavra é um conjunto de 4 bytes, separados por espaços. Cada byte desses pode estar especificado por um número decimal sem sinal entre 0 e 255, ou então por um caractere ASCII, onde seu valor numérico será usado.

No início da simulação, o simulador deve ler um arquivo texto da máquina hospedeira com 1024 palavras, que representarão a imagem de todo o disco da máquina simulada. A critério do aluno, uma nova imagem, atualizada, pode ser gravada quando o simulador é encerrado. Essa imagem deve conter todos os programas e arquivos de dados que serão utilizados. Uma vez iniciado o simulador, o operador tem a sua disposição o console e poderá comandar a execução de um ou mais programas.

Cada comando emitido pelo console deve ser verificado pelo *kernel*. As duas primeiras partições de memória são reservadas para o sistema operacional. Assim, restam 6 (seis) partições para rodar programas de usuário. Se o console solicitar a execução de um programa quando não há mais partições livres, o sistema operacional pode optar por ignorar a requisição, informar o erro, ou deixá-la em uma fila esperando por espaço para rodar. As operações de disco podem ler palavras isoladas ou blocos, sendo estes serviços definidos pelo sistema operacional.

Um aspecto importante, e que será avaliado, é que a carga do próprio programa a ser executado deve ser feita através do disco, competindo com as outras operações de entrada e saída. Ou seja, a seqüência para iniciar um processo é a seguinte:

1- o operador digita 2 números e ENTER
2- o console emite uma interrupção de *hardware* para o disco correto
3- o *kernel* atende, encontra partição, cria o descritor do processo e tabela de arquivos
4- o *kernel* verifica se o disco está livre, e enfileira ou emite requisição especial para leitura do programa inteiro (essa é uma operação obrigatória de leitura de várias palavras)

5- quando o disco terminar a leitura, emite interrupção de *hardware*
6- finalmente o *kerne*l coloca o novo processo na fila de preparados

Sistema Operacional Para Avaliação – Gerência do Processador

A gerência do processador define basicamente o algoritmo de escalonamento. Nesta versão, será implementado um algoritmo simples de fatias de tempo (ou *round-robin*), onde cada processo que sai do processador por término da fatia de tempo entra no final da fila de aptos a rodar. Outros processos que precisam entrar nesta fila, recém iniciados, ou desbloqueados de operações de entrada e saída, também devem entrar no final da fila. O tamanho da fatia de tempo fica a critério de cada estudante. Apenas recomenda-se o cuidado de implementar o sistema com uma boa coerência de tempo. A fatia de tempo deve permitir que os programas executem um número pequeno de instruções, como 8 ou 10. As operações de disco devem demorar bem mais, por exemplo, como o tempo de execução de 20 ou 50 instruções, mas devem ser chamadas com menor freqüência, ao menos em alguns programas de teste. Podem ser especificados programas padrão para teste dos trabalhos.

Sistema Operacional Para Avaliação – Interrupções de *Hardware*

A seguir são listadas as interrupções de *hardware* necessárias e padronizados seus números. O uso desta padronização visa apenas auxiliar os alunos, e não será exigido. Apenas a existência destas interrupções é que é necessária. São as interrupções:

0	Não houve interrupção. Em um sistema real, a existência da interrupção e seu número são obtidos de forma distinta. No nosso sistema, é apenas um valor testado.
2	Interrupção de relógio. Gerada por *Timer*.
3	Acesso ilegal à memória. Gerada pela *MMU*.
5 e 6	Interrupções de disco. Gerada por *Disk*.
15	Interrupção de console. Gerada por *Console*.

Sistema Operacional Para Avaliação – Interrupções do Processador

A seguir são listadas as interrupções de *hardware* geradas pelo processador. São condições de erro na execução de um programa, e por esta razão o *kernel* deve ser chamado imediatamente, assim como na interrupção de *software*. Não é que as interrupções de *software* tenham prioridade, mas como a execução da instrução que as gera já começou, então deve terminar antes de verificar as interrupções (de *hardware*) que tenham ocorrido no período, e seu término implica em atendê-la chamando o *kernel*. O mesmo vale para estas interrupções de processador. A execução da instrução com erro já começou, e, portanto deve terminar, com o tratamento do erro para o processo que a executou. A MMU, que comumente é implementada junto ao processador, também deve ter esse comportamento quando gerar as interrupções de acesso ilegal à memória.

1	Instrução ilegal. Condição testada pelo *Processor*.

Sistema Operacional Para Avaliação – Chamadas de Sistema

As chamadas de sistema são feitas com interrupções de *software*. Neste caso, para que os programas possam rodar em qualquer implementação da Má-Quina, deve haver consenso sobre os serviços, número das interrupções, e passagem de parâmetros. Assim, é obrigatório atender à interface definida abaixo. Os números das interrupções de *software* não conflitam com os das interrupções de *hardware*. Isto não é uma necessidade real, mas simplifica o ponto de entrada do *kernel*, que só precisa receber um número para saber que evento ocorreu. Os serviços hachurados não precisam ser implementados nesta versão.

32	EXIT	Indicação de encerramento do processo.
33	KILL	Não implementada.
34	OPEN	Abre um arquivo e prepara para utilização. **Entrada:** R0 deve conter 0 para leitura ou 1 para escrita; R1 deve conter o número do disco (também 0 ou 1 nessa versão); R2 deve conter o endereço inicial do arquivo no disco; **Saída:** 0 em R1 indicando operação com sucesso, e o número descritor do arquivo em R0, para o programa utilizar nas chamadas GET ou PUT. Esse número pode ser seqüencial e particular de cada processo ou pode ser um índice global do *kernel*.
35	CLOSE	Fecha um arquivo aberto. **Entrada:** R0 deve conter o número do arquivo;
36	GET	Leitura de uma palavra de um arquivo aberto para leitura, atualizando a posição corrente. Não há *buffer*ização (ou *cache*). Cada leitura ou escrita solicitada exigirá um acesso ao disco. **Entrada:** R0 deve conter o número do arquivo: 0 reservado para console; 1 e 2, arquivos de entrada; **Saída:** a palavra lida do arquivo em R0 e 0 em R1, ou 1 em R1 e o código de erro em R0: 0-fim de arquivo, 1-arquivo errado;
37	PUT	Escrita de uma palavra em um arquivo aberto para escrita, atualizando o tamanho do arquivo. **Entrada:** R0 deve conter o número do arquivo: 0 reservado para console; 3, arquivo de saída; R1 deve conter a palavra a ser escrita; **Saída:** 0 em R1 para operação correta, ou 1 em R1 e o código de erro em R0: 1-arquivo errado;
38	READ	Leitura de um conjunto de palavras de um arquivo aberto. A operação de leitura de várias palavras usaria acesso direto à memória, mas não será implementada nesta versão.
39	WRITE	Escrita de um conjunto de palavras em um arquivo aberto. Idem ao anterior.
42	SEMINIT	Operação de inicialização de um semáforo s. O sistema possui 8 semáforos no kernel, todos inicializados com o valor 0. Ao chamar SEMINIT um processo define o novo valor inicial, dado por R1, para um semáforo s, dado em R0, e isto é usado porque o número de semáforos é fixo e eles precisam ser reusados por outros programas.

		Mas se, ao chamar SEMINIT, já houver processos bloqueados nesse semáforo, todos eles são liberados e voltam para a fila de Aptos antes do valor inicial ser atribuído. Não há semáforos em 2006-1.
43	P	Operação P(s) em um semáforo s, informado em R0. Idem.
44	V	Operação V(s) em um semáforo s, informado em R0. Idem.
46	PRINT	Imprime uma palavra no console. A palavra é impressa com 4 caracteres ASCII usando o valor de seus *bytes*, seguida de uma marca de fim de linha. Todos os processos usam o mesmo console. Esta é uma forma primitiva do "operador" verificar a execução dos programas. **Entrada:** R0 deve conter a palavra a ser impressa;

Observações sobre o tratamento de interrupções:

(i) As interrupções de *software* não são sinalizadas através do Controlador de Interrupções;

(ii) As interrupções geradas pelo processador também não usam o Controlador de Interrupções, embora conceitualmente sejam geradas pelo processador que é *hardware*. Não faz sentido ele *set*ar o Controlador para em seguida testá-lo. Além disto, ele não pode concluir a operação corrente se houve erro, e por isto tem que chamar o *kernel* imediatamente. A interrupção de acesso ilegal à memória também se enquadra nessa situação.

(iii) O nosso *kernel* tem apenas um ponto de entrada, e recebe o número da interrupção que ocorreu. Em sistemas reais haveria uma tabela de ponteiros para vários tratadores de interrupção, e o próprio processador encontraria o ponto certo a chamar;

(iv) O endereço contido no PC no momento da interrupção, em máquinas reais, é salvo em uma pilha (do processo ou de sistema). Na nossa simulação isto não é necessário, pois o desvio para o *kernel* é feito na linguagem de alto-nível da nossa simulação, sem alterar o valor do PC. Assim, no ponto de entrada do *kernel*, o valor do PC (acessado através do objeto Processador) é justamente o último valor antes do desvio para o *kernel*;

(v) A *time-slice* deve ser programável. Para garantir que cada processo receba o tempo justo, um contador no *kernel* deve ser reinicializado com o valor de uma fatia de tempo imediatamente antes do *kernel* ceder a CPU ao próximo processo; O *timer* apenas gera ticks com a sua interrupção de *hardware*, e o próprio *kernel* atualiza (decrementando) esse contador, testando então se a fatia de tempo terminou.

Implementação de referência

A definição deste trabalho é acompanhada de uma implementação de referência que é dada em várias versões de código funcional em Java, desde a inicial apenas com *threads* vazias rodando em paralelo até uma implementação que modela as principais operações de intercomunicação do *hardware*. Esta implementação serve a dois propósitos: 1) mostrar a parte que é obrigatória da estrutura do trabalho, isto é, a correta modelagem do que ocorre

em paralelo e de quais componentes têm acesso a quais informações; 2) fornecer exemplos e código utilizável para alguns trechos que demandariam maior tempo de implementação devido a peculiaridades de linguagem (neste caso, Java), como leitura de arquivos e interface. A última versão da implementação de referência inclui a execução de todas as instruções da Má-Quina no processador, e a leitura do arquivo texto que guarda o conteúdo do disco para o objeto disco. A implementação de referência utiliza semáforos para sincronização, e implementa toda a modelagem de recursos de *hardware* necessários a uma Má-Quina. Os alunos devem se concentrar em desenvolver o código do *kernel* para essa Má-Quina, instanciar e controlar a presença de dois discos, dois processadores, e atualizar o *timer*. Finalmente, deve ser implementada a interface gráfica que gera os diagramas de tempo, mostrando os processos que estão nas CPUs e nos discos, bem como os eventos que ocorreram.

O trabalho pode ser implementado em qualquer linguagem de programação (incluindo pacotes adicionais), mas é aconselhado que seja feito em Java pela existência da implementação de referência, por ser multi-plataforma, e por ser mais fácil o esclarecimento de dúvidas. Os alunos que optarem por implementar em outra linguagem devem garantir que: a) a estrutura do trabalho atende as exigências, e é semelhante à apresentada na implementação de referência; b) o ambiente necessário para demonstrar o trabalho esteja disponível no horário agendado para avaliação. É de inteira responsabilidade dos alunos garantir essas duas condições.

Avaliação 2012 – Pontuação

Multiprocessamento:	
Inicialização do processo dummy	1.0 pontos
Com processos fixos (>3, e parte da gerência de memória)	1.0 pontos
Carga de Processo:	
Através da INT 15 (toda a gerência de memória)	1.0 pontos
Fila do disco ok	1.0 pontos
Acesso a Disco (INT 36 e 37) e File System:	
Acesso com GET/PUT/LOAD	1.0 pontos
Operações OPEN/CLOSE	1.0 pontos
Timer e Memória:	
Fazer o kernel controlar tempo dos processos	1.0 pontos
Matar processo por acesso ilegal	1.0 pontos
Recursos de 2012:	
Instanciando 2 CPUs, kernel re-entrante	1.0 pontos
Geração de gráficos com processos nas CPUs e discos	1.0 pontos
Geração de gráficos com identificação dos eventos	1.0 pontos
Uso de monitores no lugar de semáforos	1.0 pontos

Observações sobre a avaliação:
Certifique-se de que você consegue observar e demonstrar o funcionamento das partes implementadas.
Trabalhos que não compilam não serão avaliados: nota final 0 (zero);
Trabalhos que causam erro em tempo de execução terão sua nota multiplicada por 0.8;
A nota máxima possível nesse trabalho será de 12 sobre 10. Isso permite recuperar deficiência em outras partes da avaliação, ou a escolha de características a serem implementadas no trabalho;

Observações Adicionais

Quando não há nenhum processo apto a usar a CPU (que é o caso no início da simulação), o *kernel* deve usar um processo *dummy* para entregar a CPU a ele. Este processo deve, portanto, existir sempre, usando um descritor e uma partição de memória. O código presente em SOPA820061INT.java também não inicializa corretamente a MMU (isso é tarefa de inicialização do kernel) e por isso a interrupção 3 é chamada insistentemente.