UFRGS – INF01142 – Sistemas Operacionais I N – 2010/2 – Prof. Johann

Tarefa 4 – Definição, material a ser entregue e formato

A tarefa 4 consiste em estender a implementação do simulador do *hardware* de um computador iniciada na tarefa3, de forma a fazer o processador executar programas na arquitetura **Neander** ou chamar o *kernel* para executar um escalonador simples.

A seguir são descritas as principais características ou passos a serem implementados no Processador, na Memória, no Controlador de Interrupções e finalmente no novo componente Kernel. Os componentes do Teclado, Monitor e Relógio permanecem praticamente inalterados nessa tarefa.

O Processador deve ser capaz de executar instruções da arquitetura Neander, descrita no livro de Arquitetura de Computadores da Série Didática da UFRGS, à qual deve ser acrescentada uma instrução **INT n**, interrupção de *software* **n**, usando um dos opcodes livres do Neander. Essa instrução serve para que um processo de usuário faça chamadas de sistema solicitado serviços ao *Kernel*. O processador deve ter os registradores da arquitetura e e registradores de serviço para que funcione corretamente, como RI, REM, etc., conforme for necessário. O processador busca cada instrução da memória, testa seu opcode e executa a operação segundo a semântica da arquitetura. Após cada instrução ser executada o processador deve testar se houve interrupção ou não. Caso tenha havido, ele deve chamar o *Kernel*, que será escrito em alto nível, em linguagem C, informando o número da interrupção retornado pelo Controlador de Interrupções. Desta forma, o processador ficará parte do tempo executando instruções de processos de usuário uma uma, em *assembler* Neander, e parte do tempo executando o código do *Kernel* escrito em C.

A Memória deve ser inicializada com programas fixos, através de uma rotina de inicialização. Esses programas foram escritos para rodar em endereços absolutos de memória, onde foram colocados. Sugere-se iniciar com apenas dois programas e um escalonador mais simples, e depois incluir mais, conforme descrito adiante.

O Controlador de Interrupções possui nessa versão apenas uma posição de memória para armazenar a última interrupção ocorrida, e oferece três formas (métodos de acesso), que você pode implemenar como funções. A primeira serve para armazenar uma interrupção, a segunda para testar se houve, retornado o número da interrupção que está armazenada, e a terceira para o processador apagar esse número após seu atendimento. A função de armazenar é chamada por dispositios de *hardware* (nessa versão apenas pelo Relógio), e as duas outras são chamadas pelo processador, a cada instrução e a cada atendimento.

O *Kernel*, finalmente, é um componente que também representa o processador, em termos de *hardware*, mas o representa quando executando código do sistema operacional. Ele será, portanto, idêntico a um código de *kernel* simples escrito em linguagem de alto nível. Ele deve fazer suas quatro tarefas fundamentais: salvar contexto; atender

interrupção, escalonar processo, restaurar contexto do próximo processo. Salvar e restaurar contexo será apenas copiar todos os registradores, incluindo PC, do processador para uma estrutura de dados de cada processo na memória do *Kernel*, e vice-versa.

Nessa versão do simulador, o único evento que é solicitado implementar e deverá chamar o *Kernel* é a interrupção de tempo gerada pelo Relógio (não esqueça de colocar a função no Relógio que seta essa interrupção no Controlador de Interrupções). Conforme sua preferência, outras interrupções já podem ser incluídas para pequenos testes, mas considerando apenas a de tempo, a única tarefa de atendimento é retirar o processo que estava rodando desse estado e colocá-lo no final da sua fila.

O escalonador que você deve implementar contém duas filas, 0 e 1, onde a fila 0 tem prioridade absoluta sobre a fila 1, e cada fila pode ter vários processos, encadeados em uma lista linear, os quais se alternam na CPU pelo algoritmo *round-robin*, ou fatia de tempo. O algoritmo do escalonador é, portanto, o seguinte: para fila de 0 a 1, se há algum processo nessa fila, seleciona esse primeiro processo da fila. A cada processo que sai da CPU (atendimento da interrupção de tempo), retira-o do início da fila e o coloca no final da mesma fila. Isso nos permite concluir que se você inicializar o simulador com algum processo na fila 0, jamais será executado qualquer processo da fila 1. Isso acontece no nosso trabalho porque não há ainda implementados meios para criar ou matar processos, ou eventos que os deixem bloquados, saindo dessas filas.

Apesar do escalonador solicitado ser de duas filas, com prioridade entre elas e fatias de tempo como algoritmo interno em cada fila, é importante que você implemente primeiro algoritmos e estruturas mais simples. A primeira aconselhada é apenas um vetor de duas posições, para dois processos fixos na memória, onde a cada interrupção de tempo troca-se um pelo outro na CPU. Basta fazer "`atual = (atual+1)%2);`" para selecionar o próximo. Depois, implemente apenas uma fila com *n* processos e só então após esses testes funcionarem implemente a versão final com duas filas.

Finalmente, não esqueça de que para os processos funcionarem, deve haver estruturas de dados, seus descritores, também criadas e inicializadas de forma fixa no *Kernel*, já que não há ainda como criar processos durante a simulação. Isso quer dizer que para cada processo que você quer incluir ou experimentar na simulação, você deve não só incluí-lo na memória, mas incluir código de inicialização para mais um descritor de processos no *Kernel*, e inicializa-lo corretamente, com o valor do PC nesse descritor apontando para a posição de início desse novo processo na memória. Você pode fazer versões diferentes de rotinas para inicializar o *Kernel* com 2, 3 ou mais processos, chamando uma delas no início da simulação conforme o teste que quer fazer a cada vez.

Para essa tarefa, você deve entregar o código fonte, incluindo um *Makefile* com os objeivos *clean* e *all* para limpar e recompilar tudo, e um arquivo de *log* com exemplo de uma execução do seu programa, respeitando as demais regras de formato e disponibilização das tarefas anteriores. Um programa automático irá recuperar o arquivo *tarefa4.tgz* do seu diretório no servidor *www*.