

Tarefa 5 – Definição, material a ser entregue e formato

A tarefa 5 consiste em implementar no seu simulador um conjunto de recursos que permita o operador do sistema comandar a execução de processos via um terminal (teclado). Para isso, você deve implementar o disco e uma operação de carga, o sistema de gerenciamento de memória, o teclado do terminal, e um controlador de interrupções capaz de evitar os conflitos gerados pelos múltiplos dispositivos gerando interrupções, potencialmente ao mesmo tempo. Esses recursos são brevemente descritos a seguir.

O Disco deve armazenar um conjunto de arquivos que representam programas executáveis e identificá-los por um número, simplificando a interface que o operador e o *kernel* utilizarão. O disco pode ser pré-inicializado com dados globais (estruturas e vetores de C), ou opcionalmente você pode implementar uma função no simulador que lê o conteúdo do disco (simulado) à partir de um arquivo extra. De qualquer forma, ao rodar o simulador ele deve ter seu disco representado com o conteúdo dos diversos programas possíveis já gravados nele. Cada arquivo pode ter um tamanho diferenciado, mas aconselha-se que utilizem uma definição de tamanho máximo para facilitar a implementação. Assim, o disco pode ser uma matriz bidimensional onde as linhas representam os diferentes arquivos. O componente disco do seu simulador precisa permitir ao menos uma operação, a ser requisitada pelo kernel (ou seja, pelo processador), que é a operação de leitura de um programa a ser executado em um novo processo. A requisição é feita através de uma função (ou método) que libera um semáforo onde a *thread* do disco estará esperando, o que representa uma operação de I/O de um sistema real. Após um tempo trabalhando em paralelo, o disco responde a leitura positivamente enviando uma interrupção de disco ao controlador de interrupções. Os dados lidos do disco devem ter sido transferidos da matriz que o representa para um *buffer* interno dele, do qual o *kernel* pode copiar para a memória, ou pode ser transferida pelo disco diretamente para a memória (DMA), sendo que nesse caso, por ocasião do pedido de leitura do *kernel*, este precisa informar para qual endereço inicial de memória deve ir o programa.

O Gerenciamento de Memória consiste em implementar a MMU e as tarefas de gerência de memória do kernel. MMU deve ter os recursos de registrador base e registrador de limite e precisa ter as funções de acesso à memória para leitura e escrita que testam o limite e realizam a soma. Na verdade a MMU é essas duas funções. O kernel e/ou o disco provavelmente necessitam também acessar a memória em endereços absolutos e o kernel precisa inicializar esses registradores com o valor correto para cada processo. A gerência de memória, por sua vez, é feita por um conjunto de estruturas e funções do kernel, e usará particionamento com partições fixas. Deve haver um vetor indicando quais partições estão livres ou ocupadas, e nos descritores de processo uma indicação de qual partição cada processo recebeu. O kernel deve buscar uma nova partição para cada processo que precisa criar, antes de solicitar ao disco que leia o processo, principalmente no caso de DMA. Quando um processo termina, o kernel deve liberar a partição por ele usada. O controle de ocupação das partições pode ser feito por

um simples vetor de bytes, e com partições de mesmo tamanho é simples converter o número da partição para obter o endereço inicial do processo, a ser colocado no registrador de base da MMU a cada vez que esse processo recebe a CPU.

O Teclado do Terminal tem uma função bastante simples, que é apenas ler um número digitado, armazená-lo, gerar uma interrupção e permitir que o kernel consulte esse número. O número digitado corresponderá à identificação de qual arquivo executável o operador deseja executar em um novo processo. Aconselha-se utilizar um processo separado que se comunica com o simulador via memória compartilhada e semáforo nomeado, conforme demonstrado em aula, para implementar o teclado em uma nova janela.

O Controlador de Interrupções nesta versão precisa ser seguro para que uma nova interrupção não seja escrita sobre a anterior antes que aquela tenha sido atendida. Há diversas formas de implementar isso, mas a mais simples é usando um semáforo. Cada semáforo tem implicitamente uma fila no sistema operacional, e você estará usando essa fila em vez de implementar uma fila em seu código. Isso significa que a função de gerar a interrupção deve executar P() antes de armazenar o número, e a operação de resetar a interrupção, chamada pelo *kernel* após cada atendimento, deve executar V(). Você deve ter o cuidado de não gerar as interrupções de acesso ilegal à memória através do mesmo mecanismo no controlador de interrupções, pois o processador poderá travar nesse caso.

Finalmente, há um último recurso que precisa ser implementado, que é o serviço de término de um processo. Um processo em execução pedirá seu encerramento executando uma instrução de interrupção INT, acrescentada no Neander, passando um número que indica esse serviço. É nesse momento que o kernel deve retirar da lista de prontos e eliminar o descritor deste processo, liberando a partição de memória por ele utilizada.

Para essa tarefa, você deve entregar o código fonte, incluindo um *Makefile* com os objetivos *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. Você deve incluir um arquivo de documentação de uma ou duas páginas informando as opções principais que fez, como usar ou não DMA, os tamanhos definidos, como número de partições e seu tamanho, e qualquer outra informação de estrutura que julgar importante. Um programa automático irá recuperar o arquivo *tarefa5.tgz* do seu diretório no servidor *www*.