Atividades Práticas no Ensino Introdutório de Sistemas Operacionais

Cassio P. de Campos[1], **Nicolas Kassalias**[1]

[1] Faculdade de Computação e Informática – Universidade Mackenzie

{cassiopc,nicolask}@mackenzie.br

Resumo. *Este artigo apresenta nossa experiência na utilização de aulas práticas em laboratórios para o ensino de disciplinas introdutórias de Sistemas Operacionais. Descrevemos o ambiente utilizado, as atividades desenvolvidas com os alunos, e como a inclusão destas atividades tem sido importante no aprendizado e aumento do interesse dos alunos pelo assunto.*

Abstract. *This article presents our experience on using practical lab classes for teaching Operating Systems introductory disciplines. We describe the working environment, the activities that are proposed to students, and why the introduction of such activities has been important for learning and increasing student interest in the area.*

1. Introdução

Este trabalho apresenta nossa experiência na utilização de aulas práticas em laboratórios para o ensino de disciplinas introdutórias de Sistemas Operacionais. Diversos livros-texto sobre o assunto levantam a necessidade de trazer a prática de implementação de Sistemas Operacionais para as disciplinas teóricas [Deitel et al. 2004, Tanenbaum 1987, Silberschatz et al. 2004]. [Silberschatz et al. 2004, Deitel et al. 2004] utilizam a linguagem de programação de alto nível Java para elaborar os exemplos e exercícios. O sistema operacional Minix [Tanenbaum 1987] foi desenvolvido com foco no ensino de Sistemas Operacionais. O Minix foi essencialmente feito na linguagem de programação C.

Adotamos para as atividades práticas o Minix em sua segunda versão [Tanenbaum e Woodhull 1997]. Ele é um sistema de código aberto, baseado no Unix, escrito principalmente para propósitos educacionais. Além disso, o Minix também inspirou a criação do Linux, utilizado em larga escala atualmente. Com cerca de vinte mil linhas de código, o sistema é simples e bem documentado. Ressaltamos que, em 2005, a versão 3 do Minix foi anunciada pelo professor Tanenbaum, e ela acompanha a nova edição de seu livro. Embora ainda sirva como exemplo didático, a nova versão foi reescrita para ser mais difundida no mundo, principalmente aplicada a sistemas embutidos e/ou computadores com capacidade extremamente limitada. Ela promete ser extremamente confiável e robusta. Com as modificações, seu núcleo tem menos de quatro mil linhas de código.

A utilização de outros sistemas para o ensino, como o NachOS [Christopher et al. 1993], foram avaliadas. Todavia a história do Minix e sua relação com o Linux, atualmente muito utilizado pelos alunos, e a falta de portabilidade e facilidade de instalação do NachOS em ambientes largamente difundidos (como o

Windows) tornaram-no menos atrativo. Além disso, o Minix é um sistema operacional mais realista (no sentido de não ser apenas um simulador), e alunos com maior interesse eventualmente fazem a instalação do Minix em seus computadores pessoais. Uma vantagem da utilização do NachOS está na elaboração de atividades relacionadas ao gerenciamento de memória. Como o Minix é simplista neste sentido (não implementa, por exemplo, memória virtual), as atividades neste assunto ficam prejudicadas.

Descrevemos neste texto o ambiente computacional utilizado para aulas práticas de laboratório na disciplina Sistemas Operacionais da Faculdade de Computação e Informática da Universidade Mackenzie. Ela é uma disciplina introdutória e visa apresentar os conceitos básicos sobre processos (escalonamento, comunicação e concorrência), sistemas de arquivos, gerenciamento de memória e entrada e saída. Também fazem parte do currículo as disciplinas Sistemas Concorrentes e Distribuídos, Sistemas Embarcados e Sistemas Tolerantes a Falhas. Estas disciplinas complementam e aprofundam alguns assuntos inicialmente abordados em Sistemas Operacionais.

Apresentamos um conjunto de pequenas atividades trabalhadas durante os últimos quatro anos, em sete oferecimentos da disciplina. Argumentamos como a inclusão destas atividades tem sido importante no aprendizado e no aumento do interesse dos alunos pelo assunto.

Este trabalho está dividido como segue. A seção 2 descreve o ambiente de trabalho utilizado para facilitar as atividades propostas no laboratório. A seção 3 apresenta algumas das atividades trabalhadas com os alunos durante os últimos anos, e a seção 4 resume os resultados obtidos. Finalmente a seção 5 conclui o texto e aponta caminhos futuros.

2. Ambiente emulado para testes

Instalar o sistema operacional Minix de forma nativa em um computador requer uma partição previamente reservada para tal fim. Apesar de ser possível fazer tal configuração em um laboratório especializado da faculdade, em geral os alunos não tem essa facilidade em seus computadores pessoais em casa ou no emprego. Além disso, as atividades propostas aos alunos podem tornar o sistema instável e as alterações no núcleo do sistema operacional necessitam da reinicialização do sistema para terem efeito. Um ambiente emulado é certamente uma alternativa interessante, pois resolve todos esses contra-tempos.

Nos últimos anos utilizamos os pacotes QEMU [Bellard 2005] e Bochs [Lawton et al. 2001] para fazer a simulação de um computador executando o sistema operacional Minix. Eles têm se mostrado úteis para a realização das atividades, e certamente diminuíram o trabalho dos funcionários responsáveis pelos laboratórios na instalação dos sistemas operacionais e do ambiente educacional. Devemos lembrar que os laboratórios são compartilhados entre dezenas de disciplinas com propósitos diferentes, e a facilidade de instalação (e reinstalação) desses pacotes é essencial neste sentido.

Bochs é um emulador de PC de código aberto altamente portável. Ele inclui a emulação da CPU, dispositivos e BIOS de um PC. É capaz de executar diversos sistemas operacionais, como Linux, DOS, Windows e Minix. Tem maior suporte aos dispositivos de um PC que o QEMU, porém sua desvantagem está na eficiência: executa os códigos de forma semelhante a um processador de *scripts*, ou seja, faz a tradução em tempo de

execução. O pacote Bochs está disponível para os sistemas Linux e Windows, podendo ser rapidamente instalado pelos alunos em qualquer computador pessoal.

QEMU é um emulador genérico e de código aberto que tem excelente desempenho através de tradução dinâmica de código. Ele emula um sistema computacional completo, incluindo o processador e periféricos. QEMU é mais eficiente que o Bochs pois usa compilação dinâmica. Quando ele encontra um pedaço de código, ele o converte para instruções da máquina hospedeira. A idéia básica é quebrar cada instrução em algumas novas instruções mais simples. Cada instrução simples está implementada por um pequeno trecho de código. Assim uma ferramenta de compilação recebe um arquivo objeto e gera um código dinâmico pela concatenação das instruções simples da máquina hospedeira.

Uma das dificuldades encontradas pelo alunos no uso dos emuladores é o transporte de arquivos internos ao emulador e externos do sistema operacional hospedeiro. Transportar arquivos de dentro para fora do emulador e vice-versa é essencial, pois:

- As atividades propostas não são suficientemente pequenas para que os alunos comecem e terminem em um mesmo momento.
- Os alunos têm a obrigação de escrever relatórios, e por isso precisam dos trechos de código alterados para elaborá-lo.
- Algumas atividades podem ser facilitadas com os arquivos do trabalho realizado em atividades anteriores.
- A edição dos arquivos dentro do simulador é restrita, pois o editor disponível é antigo e com poucos recursos.
- A alteração do núcleo do ambiente pode tornar o sistema instável, e o aluno pode eventualmente perder os arquivos modificados.

Essa dificuldade de transporte de arquivos deve ser enfrentada logo no início do curso, para que todas as atividades subseqüentes sejam facilitadas e que a motivação venha dos desafios propostos relacionados à implementação de sistemas operacionais, sem desgastes ou tempo desperdiçado com problemas não centrais.

3. Atividades de Laboratório

Descrevemos nesta seção as atividades práticas que tem obtido maior sucesso em relação ao aprendizado e interesse dos alunos. As atividades estão descritas em ordem cronológica de utilização na disciplina introdutória de Sistemas Operacionais. Vale ressaltar que é essencial que as aulas teóricas abordem os assuntos de forma acoplada às atividades práticas, criando no aluno a necessidade de estar atento a todas as explicações.

Usualmente exigimos relatórios detalhados sobre as atividades, incluindo explicações sobre os testes realizados, detalhes sobre o que foi alterado (ou criado) no código e como foram analisadas e testadas as alterações. O principal foco das atividades é trazer ao aluno a experiência prática de manipular os conceitos abordados durante a exposição da parte teórica da disciplina, sem a pretensão da criação de novas idéias ou mesmo melhorias na qualidade da implementação do sistema operacional. Ainda que possam acontecer, tais idéias e melhorias não são exigidas.

3.1. Trocas de contexto

A primeira atividade proposta aos alunos em geral é muito simples e tem o objetivo de ambientá-los com as ferramentas e problemas que serão enfrentados durante o curso.

Solicitamos que os alunos alterem o núcleo do sistema para que toda vez que a tecla F4 for pressionada seja impresso na tela uma mensagem simples, como `Kernel hacked!`. Indicamos detalhadamente os passos a serem feitos para que a atividade seja cumprida, pois o objetivo aqui é exatamente conhecer o ambiente.

Em um segundo passo, propomos uma pequena alteração adicional, onde toda vez que a tecla F4 é pressionada imprimimos o número de trocas de contexto. A princípio não indicamos claramente o significado do número que está sendo calculado, e o aluno apenas percebe que estamos contando o número de vezes que uma dada função é executada. Pedimos que eles realizem testes gerais (executem programas, digitem informações quaisquer) e analisem o comportamento daquele número. No devido momento eles recebem a explicação precisa sobre as trocas de contexto.

3.2. Tabela de Processos e Chamadas de Sistema

Nesta atividade propomos que o aluno escreva um programa que pergunte ao usuário por um número N, e então se utilize da chamada de sistema *fork* para criar N processos filhos. Cada processo filho deve imprimir uma mensagem que diz `Filho com PID=X criado`, onde `X` é o número do processo filho (*Process ID*). Após imprimir esta mensagem, cada filho deve esperar dois segundos (através do uso da chamada de sistema *sleep*), e então imprimir a mensagem `Filho com PID=X terminando`. O processo pai deve esperar até que todos os filhos tenham terminado (através do uso da chamada de sistema *wait*), e então deve imprimir na tela a mensagem `Pai terminando`.

Além de trabalhar com diversos processos e chamadas de sistema, essa atividade é utilizada para testar o número de processos que o sistema pode carregar para execução. Para isso, pedimos que os alunos utilizem seu próprio código, aumentando o valor de N até que a chamada *fork* não consiga criar novos filhos. Usando comandos de visualização de processos, como o `ps`, os alunos conseguem perceber o limite da tabela de processos.

Em uma nova parte da atividade, pedimos que o núcleo do sistema seja alterado, aumentando o tamanho da tabela de processos. Isso leva os alunos a investigarem o código-fonte do núcleo, a tabela de processos e seus atributos.

Introduzimos também aos alunos o ataque conhecido por *fork bomb*, e fazemos com que eles sobrecarreguem o sistema e analisem as conseqüências (incluindo seu travamento). A experiência mostra que existe uma satisfação geral dos alunos ao conseguirem criar um código que faça o sistema travar. Então solicitamos que o núcleo seja alterado, incluindo um verificador na criação de processos que limite a quantidade que cada usuário pode criar. Isso mostra aos alunos uma possível solução para o problema em questão.

3.3. Alterações no Escalonador

Propomos aqui uma mudança de parâmetro no algoritmo de escalonamento utilizado e então uma mudança de algoritmo. O núcleo do Minix normalmente utiliza 100 milissegundos como *quantum* para o algoritmo Round-Robin. Nesta atividade é proposta uma alteração do núcleo para que, toda vez que o usuário pressionar a tecla F6, o *quantum* seja acrescido em 30 milissegundos, e toda vez que for pressionada a tecla F7, o *quantum* seja decrescido em 30 milissegundos. Além disso, o aluno deve criar cenários para testar as mudanças, verificando os tipos de programas que se beneficiam dos diversos valores para o *quantum*.

A segunda parte é a mudança do algoritmo de escalonamento propriamente dito. Em um primeiro passo, sugerimos ao aluno simplesmente remover do código as instruções que retiram um processo do começo da fila e colocam-o no final da mesma. Essa alteração leva a efeitos interessantes (e as vezes desastrosos) para o sistema. Uma outra alteração de algoritmo proposta é a troca do Round-Robin pelo algoritmo garantido: será escolhido para executar o processo que recebeu a menor fatia da CPU até o momento. Note que usualmente não exigimos dos alunos soluções com estruturas de dados eficientes para agilizar a troca de contexto, dado que essa exigência implicaria em aumento significativo da dificuldade de completar a atividade.

Após as alterações, pedimos que os alunos tentem medir (informalmente) a eficiência do sistema, verificando se a alteração produziu um ambiente melhor ou pior. Utilizamos também o resultado da atividade da seção 3.1 para analisar as quantidades de trocas de contexto de cada algoritmo. Como o código do escalonador é parte crítica do sistema, o aluno pode perceber que pequenas alterações e maneiras diferentes de tratar o problema tendem a mudar significativamente seu desempenho.

3.4. Problemas de Concorrência

Nesta atividade propomos aos alunos a implementação e teste dos códigos para diversas idéias que resolvem (ou tentam resolver) o problema da exclusão mútua de uma região crítica. Temos trabalhado com as seguintes idéias:

- **Variáveis de bloqueio (*Lock*)**. Espera-se que o aluno perceba que a introdução destas variáveis não resolve o problema de exclusão mútua. Para tal disponibilizamos código-fonte que se utiliza de *threads* e imprime a situação de cada uma e um alerta quando duas *threads* entram "ao mesmo tempo na região" crítica. Ainda assim é usualmente difícil de verificar na prática a ocorrência deste problema, pois é necessária a combinação exata de tempo entre as *threads* no teste da variável de bloqueio. Resolvemos esse problema incluindo algum tipo de "gasto de tempo inútil" entre o teste da variável de travamento e sua atualização (instruções que normalmente ficam muito próximas no código, dificultando a ocorrência de problema). Isso possibilita ao aluno verificar que o problema da exclusão mútua não foi resolvido.
- **Alternação estrita**. Espera-se que o aluno note, nesta solução para o problema de exclusão mútua, que uma *thread* fora de sua região crítica pode impedir a entrada de outra, o que é extremamente indesejado. Fazemos isso solicitando aos alunos que implementem códigos que fiquem alternando entre região crítica e região não-crítica, e que eles incluam algum tipo de "gasto de tempo inútil" na região não-crítica de uma das *threads*. Com isso e algumas linhas de depuração, é possível visualizar que a *thread* com região não-crítica mais demorada irá impedir a outra de entrar na região crítica, devido à idéia da alternação estrita.
- **Solução de Peterson.** O atributo importante desta solução é sua independência total do *hardware*. No laboratório é possível mostrar ao aluno que esta solução sofre com o problema da **espera ocupada** (ou *busy waiting*), assim como as duas idéias anteriores. Podemos visualizar tal problema analisando a quantidade extremamente grande de trocas de contexto que ocorre enquanto executamos o experimento. A atividade da seção 3.1 ajuda nesta constatação. Vale ainda comparar esta solução com a alternação estrita, mostrando que o problema enfrentado anteriormente foi resolvido.

- **Semáforos.** Para trabalhar com semáforos, usualmente utilizamos os problemas clássicos produtor-consumidor e filósofos. O essencial neste ponto é conseguir mostrar no laboratório que as dificuldades das idéias anteriores são devidamente atacadas e resolvidas com o uso de semáforos. Em geral sugerimos que os alunos verifiquem o comportamento dos códigos para produtor-consumidor e filósofos utilizando quantidades superiores a duas *threads* em execução, apresentando assim mais uma qualidade da solução com semáforos.

Notamos nas aulas de laboratório com alunos que nunca trabalharam com concorrência e implementações usando *threads* que é necessário dar uma ajuda especial para que esta atividade seja executada com sucesso. Sugerimos, nos casos onde o tempo é limitado, disponibilizar aos alunos (de forma total ou parcial) os códigos-fonte, e deixá-los responsáveis apenas pela condução de testes e pequenas alterações nos códigos.

3.5. Depurador de Alocações de Memória

Nesta atividade propomos que sejam impressas linhas (contendo informações sobre o processo) sempre que houver uma alocação ou liberação de memória. Para testar as alterações, pedimos que seja criado um *memory bomb*. Além dele, solicitamos que os alunos criem um programa que faça alocações e liberações de memória aleatoriamente, mas de forma esperta para que seja possível chegarmos em uma situação de falta de memória (ou pelo menos a falta de um espaço contínuo de tamanho suficiente para a requisição, já que o Minix não possui esquema de memória virtual ou *swapping*). Exigir que se altere o núcleo do Minix, incluindo por exemplo um esquema de memória virtual ou *swapping* são atividades interessantes, porém usualmente maiores do que o escopo e tempo disponível em uma disciplina introdutória.

3.6. Inclusão de Lixeira no Sistema de Arquivos

O sistema de arquivos do Minix (assim como os sistemas de arquivos do Linux) não possui uma forma de "desistir" de uma remoção já executada. O objetivo desta tarefa é alterar o gerenciador de arquivos para que, toda vez que um arquivo é removido, ele seja de fato transportado para o diretório /undelete localizado na raiz da árvore de diretórios. Arquivos removidos do diretório /undelete podem ser permanentemente apagados.

Quando diretórios são removidos ou arquivos com nomes iguais, temos algumas dificuldades. Uma forma de facilitar o serviço e tornar a atividade mais direta é assumir que alguns casos mais difíceis simplesmente não acontecem. Outro detalhe importante é que a cópia do arquivo para o diretório /undelete pode ser lenta, pois o arquivo original pode estar em uma partição diferente da partição daquele diretório. Uma solução elegante é exigir que, no lugar de sempre mover os arquivos para o diretório /undelete, movê-los para o diretório undelete existente na raiz da própria partição onde o arquivo estava localizado. Caso o diretório undelete não exista no diretório raiz da partição onde o arquivo foi apagado, então esse arquivo deve ser apagado diretamente (essa é uma forma de controlar se desejamos ter ou não a operação de *undelete* em funcionamento).

Além de fazer o aluno trabalhar com o sistema de arquivos, esta atividade tem como finalidade mostrar uma alteração prática para o dia-a-dia, pois certamente os alunos já passaram pela experiência de remover um arquivo e desejar tê-lo de volta. Podemos ainda sugerir e conduzir experimentos para analisar se a alteração diminui significativamente o desempenho do sistema de arquivos.

4. Resultados da Experiência

As atividades citadas foram conduzidas em sete oferecimentos da disciplina Sistemas Operacionais para o bacharelado em Ciência da Computação da Faculdade de Computação e Informática da Universidade Mackenzie, oferecidas nos últimos quatro anos. A disciplina é semestral e tem carga horária de quatro horas semanais, sendo duas horas teóricas em sala e duas horas em laboratórios de informática. Nos laboratórios, as turmas tem tamanho reduzido e os alunos podem ser acompanhados de perto pelos professores.

Com a inclusão de atividades diretamente relacionadas com as aulas teóricas, notamos aumento do interesse dos alunos e conseqüente diminuição no número de reprovações (ainda que esse último efeito possa ter outros fatores relacionados). Constatamos também que a Faculdade de Computação e Informática teve seu primeiro projeto de conclusão de curso de graduação puramente na área de Sistemas Operacionais em 2005. Através de algumas entrevistas, podemos afirmar que este é um efeito das aulas de laboratório da disciplina de Sistemas Operacionais, oferecida aos alunos do quinto semestre do curso.

Argumentamos que é mais produtivo, para uma disciplina introdutória, uma seqüência de atividades curtas do que um projeto mais elaborado e extenso. Acreditamos que um projeto extenso tem melhor aplicação quando os alunos já têm algum conhecimento prévio sobre o assunto. Propomos uma nova atividade a cada duas semanas, e conduzimos o tempo disponível no laboratório para que grande parte das atividades sejam executadas com o acompanhamento do professor.

Realizamos experimentos com atividades mais extensas, reduzindo a quantidade para duas e com duração prevista para 6 semanas. Dentre elas, podemos citar o desenvolvimento de um novo sistema de arquivos, a criação do gerenciamento de memória baseado em *swapping*, e a modificação completa do esquema de escalonamento, utilizando prioridades. O resultado final foi menos satisfatório. Os alunos tiveram grande dificuldade para completar as atividades, e poucos atingiram o objetivo.

Indicamos ainda que não existe necessidade do acompanhamento presencial do professor. O fato importante é criar um cronograma dos assuntos teóricos que seja fortemente relacionado com as atividades práticas propostas. Introduzimos, no último oferecimento da disciplina, o uso do ambiente de educação não presencial Moodle [Cole 2005]. Através de alguns experimentos com *chats* e fóruns, o acompanhamento não presencial obteve excelente efeito. Todavia não pode ser comparado com as aulas em laboratório, onde a proximidade do professor tem obtido maiores benefícios.

5. Conclusão

Este texto descreve a experiência no ensino de disciplinas introdutórias em Sistemas Operacionais através do uso de aulas práticas em laboratório. Evidenciou-se neste estudo que as atividades práticas de alteração do núcleo do sistema operacional Minix trouxeram ao aluno um aumento significativo no seu interesse pelo assunto e conseqüente melhora em seu desempenho acadêmico. Argumentamos que as aulas práticas presenciais podem ser substituídas pela utilização de algum sistema de ensino não-presencial, e indicamos nossa experiência com a utilização do ambiente Moodle.

Sugerimos que este texto pode ser utilizado como um ponto de partida para au-

las práticas de disciplinas de introdução a Sistemas Operacionais. Indicamos que, para uma disciplina introdutória, atividades curtas em maior quantidade mostraram melhores resultados em motivação e aprendizado que atividades longas em menor número. Grande número de cursos de graduação no país tem disciplinas de Sistemas Operacionais em nível introdutório, e muitas disciplinas não utilizam atividades práticas. As versões detalhadas das atividades aqui propostas e suas soluções comentadas podem ser solicitadas aos autores.

Um caminho futuro é introduzir nos cursos novas atividades que busquem trabalhar com o núcleo do sistema operacional Linux, pois a manipulação de um sistema largamente utilizado em escala prática certamente aumentará o interesse dos alunos. O Minix tem demonstrado excelentes qualidades para o ensino, e a clareza de seu código ajuda o desenvolvimento das atividades. Ainda assim acreditamos (e estamos iniciando) propostas de pequenas alterações no núcleo do Linux, para que no mínimo seja criado no aluno o sentimento de alteração de um sistema que está em produção, muitas vezes na própria casa, estágio ou emprego.

Finalmente vale ressaltar que a constante utilização de ambientes derivados do Unix (ou semelhantes) não deve retirar a generalidade nas explicações teóricas sobre sistemas operacionais, e deve ficar claro que muitas das técnicas estudadas também são empregadas em sistemas comerciais/proprietários.

Referências

Bellard, F. (2005). QEMU, A fast and portable dynamic translator. Em *USENIX Annual Technical Conference, FREENIX Track*, páginas 41–46. USENIX Associaton.

Christopher, W. A., Procter, S. J., e Anderson, T. E. (1993). The Nachos Instructional Operating System. Em *USENIX Winter*, páginas 481–488.

Cole, J. (2005). *Using Moodle*. O'Reilly, first edition.

Deitel, H. M., Deitel, P. J., e Choffnes, D. R. (2004). *Operating Systems*. Prentice Hall, third edition.

Lawton, K., Denney, B., e Bothamy, C. (2001). The Bochs IA-32 emulator project. http://bochs.sourceforge.net/. Acesso em abril/2006.

Silberschatz, A., Galvin, P., e Gagne, G. (2004). *Operating System Concepts with Java*. Wiley.

Tanenbaum, A. S. (1987). *Minix Binaries and Sources*. Prentice Hall.

Tanenbaum, A. S. e Woodhull, A. S. (1997). *Operating Systems: Design and Implementation*. Prentice Hall, second edition.