Reflexões sobre o ensino prático de Sistemas Operacionais

Carlos Maziero

Programa de Pós-Graduação em Informática Aplicada
Pontifícia Universidade Católica do Paraná
E-mail: maziero@ppgia.pucpr.br

Abstract. *This work presents an analysis of laboratory practices on operating systems teaching, under a constructivist viewpoint. The most common approaches in practical operating systems teaching are enumerated and compared, in order to reveal their qualities and problems, and to suggest appropriate contexts for their use.*

Resumo. *Este trabalho apresenta uma análise crítica das práticas de laboratório na disciplina de Sistemas Operacionais sob uma perspectiva construtivista. As principais abordagens empregadas no ensino prático de sistemas operacionais são enumeradas e comparadas, buscando deixar claras suas virtudes, deficiências e seus contextos de uso mais adequados.*

1. Introdução

A disciplina de *Sistemas Operacionais* é uma disciplina fundamental nos currículos de graduação em Ciência da Computação, ao lado de outras disciplinas ditas "de sistema", como *Arquitetura de Computadores*, *Redes* e outras. Nesse contexto, ela busca apresentar aos alunos os conceitos e mecanismos fundamentais usados na construção de sistemas operacionais e como esses conceitos podem ser usados – e influenciar – na construção de aplicações.

Além dos tópicos clássicos da área, como gerência de processos, memória e arquivos, a ementa da disciplina pode ainda cobrir assuntos específicos como escalonamento de tempo real, coordenação distribuída, sistemas embarcados, máquinas virtuais, etc. Uma boa discussão sobre o conteúdo programático desta disciplina pode ser encontrada em [Anido, 2000].

Uma das principais características da disciplina de Sistemas Operacionais é a relativa dificuldade em definir um seqüenciamento didático claro entre seus diferentes tópicos. Existem diversas inter-dependências, sobretudo no que se considera o "núcleo" da disciplina. Por exemplo, a noção de contexto de processo depende de conceitos associados à gestão de memória, como o espaço de endereçamento, enquanto esta depende de conceitos oriundos da gestão de processos, como o estado dos processos.

O sistema operacional constitui um conjunto de software extenso, complexo e de difícil compreensão global, por executar várias tarefas bastante diversas, mas complementares e inter-dependentes. De um lado, ele manipula detalhes do hardware, como registradores e interrupções. No outro extremo, ele deve definir políticas de alto nível para o uso dos recursos do sistema, e construir abstrações desses recursos para simplificar

a construção e gerência das aplicações. Como exposto em [Perez-Davilla, 1995], o sistema operacional constitui uma "ponte" entre o mundo abstrato das teorias e algoritmos e o mundo prático e concreto do hardware. Essa característica de ponte entre dois mundos tão distintos pode tornar o ensino de sistemas operacionais uma tarefa trabalhosa e pouco efetiva. Notadamente, exige-se do alunos uma grande capacidade de construir abstrações mentais dos mecanismos envolvidos para poder compreender os aspectos mais densos da disciplina. Com isso, observa-se que muitos alunos concluem a disciplina conhecendo os principais conceitos e algoritmos, mas têm muita dificuldade em integrar todos aqueles conceitos de forma coerente.

Por isso, o ensino de sistemas operacionais deve ter uma componente prática significativa e muito bem elaborada, que complemente as aulas teóricas e, sobretudo, permita ao aluno ter uma compreensão integrada das diversas partes de um sistema operacional. O objetivos deste trabalho é analisar as abordagens em uso e discutir suas virtudes e deficiências. Este documento está estruturado como segue: esta seção introduz o contexto para o desenvolvimento do trabalho; a seção 2. faz uma rápida revisão dos principais conceitos da pedagogia construtivista; a seção 3. analisa a aplicação do construtivismo no ensino da informática e a seção 4. faz uma análise crítica das principais abordagens para experimentação no ensino de sistemas operacionais, levantando suas possibilidades e deficiências.

2. A pedagogia construtivista

O psicólogo suíço Jean Piaget lançou as bases do construtivismo, formulando a *Epistemologia Genética* [Piaget, 1932, Bybee and Sund, 1982], que analisa a evolução do raciocínio humano, do nascimento à maturidade do indivíduo. Piaget formulou um modelo com base na interação *sujeito – objeto*, segundo o qual o conhecimento não está no sujeito nem no objeto, mas na interação entre ambos. A formação do conhecimento depende da ação simultânea do sujeito e do objeto um sobre o outro.

Em seus estudos, Piaget considera duas formas de conhecimento: o *conhecimento físico*, que consiste no sujeito explorando os objetos, e o *conhecimento lógico-matemático*, que consiste no sujeito estabelecendo novas relações com os objetos. Piaget aborda a inteligência como algo dinâmico, decorrente da construção de estruturas cognitivas que, à medida que vão sendo construídas, vão se alojando no cérebro. A inteligência, portanto, não aumenta por acréscimo e sim por *reorganização*. Sob esta ótica, pode-se afirmar que o indivíduo aprende por si, construindo e reconstruindo suas próprias hipóteses sobre a realidade que o cerca.

Sob a perspectiva construtivista, os fatores mais influentes no desenvolvimento do conhecimento são a maturação biológica, a interação com os objetos, a transmissão social (informações transmitidas no seio do grupo social) e a *equilibração*. Este último ponto é o que contrabalança os outros três, ou seja, equilibra uma nova descoberta com todo o conhecimento até então construído pelo sujeito. Os mecanismos de equilíbrio são a *assimilação* e a *acomodação*. Todas as idéias tendem a ser *assimiladas* às possibilidades de entendimento até então construídas pelo sujeito. Se ele já construiu as estruturas necessárias, a aprendizagem tem o significado real a que se propôs. Se, ao contrário, ele não possui essas estruturas construídas, a assimilação é deformante, resultando em um *erro construtivo*. Diante disso, havendo o desafio, o sujeito faz um esforço contrário ao

da assimilação. Ele modifica suas hipóteses e concepções anteriores ajustando-as às experiências impostas pela novidade que não foi passível de assimilação. É o que Piaget chama de *acomodação*, onde o sujeito age no sentido de transformar-se, em função das resistências impostas pelo objeto.

O desequilíbrio, portanto, é fundamental para que haja o erro construtivo, a fim de que o sujeito sinta a necessidade de buscar o reequilíbrio. Portanto, o desenvolvimento da inteligência se processa para que o sujeito consiga manter o equilíbrio com o meio ambiente, redefinindo e recombinando suas estruturas de conhecimento de forma interativa e contínua. Nesse contexto, o erro, em vez de denunciar uma não-aptidão, torna-se uma etapa necessária do processo de construção do conhecimento [Bybee and Sund, 1982].

A partir da teoria de Piaget, J. Bruner e outros pesquisadores em educação elaboraram a *teoria construtivista* [Bruner, 1966], na qual o aprendizado é considerado um processo ativo: o aprendiz seleciona e transforma a informação recebida, constrói hipóteses e toma decisões, contando, para isto, com uma estrutura cognitiva. A estrutura cognitiva (esquemas, modelos mentais) fornece significado e organização para as experiências e permite ao indivíduo "ir além da informação dada" [Bybee and Sund, 1982]. Bruner enuncia em [Bruner, 1966] alguns princípios para a organização do processo de aprendizagem:

- A instrução precisa estar preocupada com as experiências e os contextos que levam o aluno a estar pronto e apto para aprender.
- A instrução precisa ser estruturada para que possa ser facilmente compreendida pelo aluno, de forma espiral, para que o aluno construa continuamente sobre o que já aprendeu.
- A instrução precisa ser criada para facilitar a extrapolação ou preencher as brechas no conhecimento (ir além da informação dada).

Tendo por base os princípios construtivistas, o professor passa a ser o mediador da relação entre o sujeito que aprende e o objeto do conhecimento. Essa mediação nada mais é que a intervenção planejada para favorecer a ação do aprendiz sobre o objeto. Para cumprir seu papel com êxito, ele deve seguir alguns princípios em seu relacionamento com os alunos:

- Respeitar a produção individual do aluno, valorizando suas conquistas e estimulando-o na direção adequada;
- Prover um espaço adequado para o aluno interagir com o meio e testar suas hipóteses;
- Traduzir a informação a ser aprendida em um formato apropriado ao estado verdadeiro de compreensão do aluno.
- Estimular o trabalho em grupo, favorecendo assim as interações sociais que facilitam o aprendizado.

Para o exercício da mediação, o professor precisa ter instrumentos para detectar com clareza o que seus alunos já sabem e o que eles não sabem. Para isso necessita de um conhecimento consistente do conteúdo, do objeto do conhecimento e de informações sobre o processo de construção, que lhe permita antecipar o caminho através do qual o aluno vai se apropriar desse conhecimento. Conforme [Weisz, 1999], um mediador é alguém que, em cada momento, em cada circunstância, toma decisões pedagógicas

conscientes: nunca está limitado a corrigir ou deixar errado, pois além de informar e respeitar o erro quando construtivo, ele pode problematizar, questionar e ajudar a pensar.

3. Construtivismo e informática

O ensino de informática envolve ao menos dois aspectos que devem ser abordados de forma distinta, no que diz respeito à sua abordagem pedagógica. Por um lado, *informática é matemática*, o que direciona o aprendizado no sentido de favorecer a assimilação de conceitos lógico-formais, ou seja, a construção de abstrações mentais que permitam ao aluno compreender as bases matemáticas e algorítmicas subjacentes ao conteúdo apresentado. Essa é a perspectiva das disciplinas ditas "fundamentais' como lógica de programação, algoritmos e estruturas de dados, ou mesmo de disciplinas mais avançadas como inteligência artificial e compiladores.

Por outro lado, *informática é engenharia*. Esse enfoque prioriza os aspectos arquiteturais do conteúdo apresentado, como as relações existentes entre os diversos elementos apresentados e as estruturas utilizadas para sua interação. Nesse contexto, os aspectos algorítmicos ou matemáticos do conteúdo apresentado são colocados em segundo plano. Essa abordagem é certamente a mais adequada para as disciplinas ditas "de sistema", como é o caso de arquiteturas de computadores, sistemas operacionais e redes. Apesar disso, muitos professores dão uma clara ênfase aos aspectos algorítmicos no ensino de disciplinas de sistema, em detrimento dos seus aspectos arquiteturais. Citando Tanenbaum [Tanenbaum and Woodhull, 1999], "muitos cursos não dão a ênfase adequada aos assuntos realmente importantes, como entrada/saída ou gerência de arquivos, e concentram-se demasiadamente em assuntos de importância secundária, como escalonamento de processos ou tratamento de *deadlocks*".

O ensino de ciência da computação tem historicamente empregado uma abordagem *instrutivista*, que se baseia no desenvolvimento de um conjunto de seqüências instrucionais com objetivos pré-definidos. O objetivo do ensino instrutivista é facilitar a transferência de conhecimento entre um mestre ativo e um aprendiz passivo. A abordagem clássica das aulas expositivas traz embutido o conceito do modelo instrutivista, buscando a exposição da maior quantidade possível de informação [Kanuka and Anderson, 1999].

Essa abordagem tem evidentemente suas falhas. Confrey [Confrey, 1990] sugere que o modelo instrutivista imputa ao professor, ao invés do aluno, a responsabilidade pela simplificação do conteúdo didático. Assim, os alunos recebem um conhecimento simplificado de uma realidade complexa e são privados do conhecimento que o processo de simplificação em si pode trazer. A teoria construtivista coloca menos ênfase nas seqüências de aprendizagem e mais ênfase no projeto de um *ambiente de aprendizagem*, no qual o aprendiz desempenha um papel ativo e essencial. Sob essa ótica, aprender é um processo ativo de construção do conhecimento e instruir é sobretudo dar suporte ao processo de aprendizado, mais que simplesmente comunicar conhecimento [Duffy and Savery, 1995].

Em [Ben-Ari, 1998], Ben-Ari expõe diversas possibilidades de uso de técnicas construtivistas no ensino de ciência da computação. Na prática do ensino de sistemas operacionais, a aplicação de métodos construtivistas pode trazer diversos benefícios aos alunos. Sem dúvida, um de seus maiores benefícios é conduzir o aluno no processo de

construir um modelo mental do computador e seu sistema operacional, que o permita conjeturar na direção correta e assim "ir além do conhecimento transmitido".

O emprego de técnicas construtivistas no ensino de ciência da computação já foi investigado por diversos autores [Denning, 1989, Ben-Ari, 1998]. No caso específico de sistemas operacionais, poucos estudos foram efetuados nesse sentido [Perez-Davilla, 1995]. Todavia, através do exposto é possível estabelecer algumas premissas básicas para as atividades de laboratório em sistemas operacionais sob uma ótica construtivista:

- O planejamento das *situações de aprendizagem* deve oportunizar a vivência de um contexto didático rico em interações com o objeto de estudo.
- O *erro construtivo* é resultado de uma hipótese lógica do aluno, necessário no seu processo de apropriação desse conhecimento. Para que o erro construtivo ocorra, é necessária a *interação* entre o aluno e o sistema operacional. Todavia, é importante que o aluno possa descobrir e compreender as causas do erro, com o auxílio de ferramentas adequadas.
- A *troca entre iguais*, isto é, a interpretação entre os alunos, é fundamental para que eles avancem em suas concepções sobre o objeto de estudo. Como os alunos pensam de forma distinta, a heterogeneidade de uma turma enriquece e apóia o trabalho do professor.

4. Laboratório de Sistemas Operacionais

Como exposto na seção 1., a componente prática da disciplina de sistemas operacionais tem grande importância pedagógica, como complemento dos conceitos abordados nas aulas teóricas, mas sobretudo como ferramenta para que o aluno possa extrapolar seu conhecimento. Sob este aspecto, as aulas práticas de sistemas operacionais devem buscar os seguintes objetivos:

- Consolidar a compreensão dos mecanismos e estruturas básicas do funcionamento de um núcleo de sistema operacional típico.
- Compreender claramente como as diversas partes constituintes de um sistema operacional interagem e se integram.
- Observar o impacto das políticas internas do núcleo no funcionamento das aplicações.
- Compreender os compromissos envolvidos nas escolhas de implementação.
- Desenvolver a capacidade de propor e testar suas próprias soluções.

Para atingir esses objetivos, as aulas de laboratório devem ser planejadas de forma a incentivar o aluno a interagir com o sistema operacional e explorar seus diversos aspectos, tanto externos quanto internos. Essa certamente não é uma tarefa simples, haja visto a complexidade do tema e a ampla variedade de tópicos a considerar. Outro fator dificultante é a dificuldade de observação da maior parte dos mecanismos envolvidos. Muitos dos mecanismos envolvem operações de baixo nível sobre os registradores da CPU (como as trocas de contexto), sobre os endereços de memória (como a memória virtual) ou em dispositivos de entrada/saída (como os *drivers* de disco).

As atividades de laboratório devem portanto explorar o uso dos serviços do sistema operacional para a construção de aplicações (aspectos externos) e a forma como

esses serviços são construídos (aspectos internos). Na seqüência serão apresentadas e analisadas algumas das principais técnicas normalmente empregadas nas aulas práticas de sistemas operacionais.

4.1. Uso de abstrações do núcleo

Esta técnica consiste em propor a construção de programas aplicativos que façam uso dos mecanismos oferecidos pelo núcleo, como comunicação entre processos, controle de concorrência, escalonamento de tempo-real, etc. Seu objetivo básico é estimular a compreensão das abstrações oferecidas pelo núcleo do sistema operacional. Um exemplo típico de aplicação desta técnica é a construção de programas clientes e servidores para protocolos usuais de rede usando *sockets*, ou programas simulando problemas de sincronização usando semáforos, como o conhecido *Jantar dos Filósofos* e tantos outros.

Sob uma perspectiva construtivista, essa técnica deve ser abordada sob a forma de mini-projetos, nos quais o desafio é exposto aos alunos e o professor passa a ser uma referência de informação em caso de dúvidas (que não deve ser no entanto uma referência passiva, mas instigante e desafiadora). De preferência, deve-se envolver o uso de programas ou ferramentas familiares ao aluno (como clientes de e-mail ou web comerciais, no caso do dos servidores de rede). Esse fator traz diversos benefícios:

- contribui muito para contextualizar o conhecimento adquirido pelo aluno, deixando claras suas possibilidades e limitações;
- faz com que o aluno valorize seu trabalho e sua capacidade, na medida em que "o produto de suas próprias mãos" consegue interagir com softwares complexos do mundo real;
- desmistifica os produtos comerciais, demonstrando que, apesar de sua complexidade e sofisticação, estes seguem conceitos básicos e devem obedecer padrões para funcionar corretamente (freqüentemente o aluno "descobre" que consegue compreender o princípio de funcionamento do software comercial usado, e até arrisca desenvolver sua própria versão simplificada do mesmo).

Esta técnica permite ao aluno alcançar uma boa compreensão das abstrações oferecidas pelo núcleo do sistema operacional, tornando-o capaz de usá-las para resolver seus problemas de programação. Todavia, ela não contribui para compreender como essas abstrações são construídas e gerenciadas pelo núcleo do sistema. Portanto, essa técnica é mais adequada para cursos onde o enfoque principal é o uso do sistema operacional, e não a construção e gerência de suas estruturas internas. Ela tem seu lugar na prática de laboratório de sistemas operacionais em ciência da computação, mas não pode ser a única abordagem utilizada.

4.2. Análise por inferência

Esta técnica tem como idéia motriz a utilização do sistema operacional de forma a suscitar conjeturas sobre o comportamento dos mecanismos do núcleo subjacente. Para tal, normalmente são desenvolvidos programas para testar a reação do núcleo a cargas computacionais diversas, como por exemplo demandas excessivas de uso de memória, de entrada/saída ou de criação de novos processos.

A atividade de laboratório a seguir representa, de forma bastante simplificada, um exemplo típico dessa abordagem:

Atividade: comparação de políticas de escalonamento

1. Escreva e compile o seguinte programa C:

```
void main () {
   while (1) ;
}
```

2. No LINUX, lance 5 processos a partir do executável gerado no passo anterior, cada um em uma janela de terminal distinta.
3. Analise a distribuição da CPU entre os processos lançados.
4. Repita os passos anteriores no ambiente WINDOWS.
5. Teça uma análise comparativa sobre os resultados obtidos.

Esta técnica busca estimular a capacidade do aluno em relacionar os conceitos abordados nas aulas teóricas ao comportamento de sistemas operacionais reais. Embora os mecanismos internos não sejam explicitamente "manipulados", seus efeitos e conseqüências diretas são analisados em situações concretas.

Qual o melhor contexto para a aplicação desta técnica ? Novamente, por explorar mais a influência das decisões do sistema operacional sobre as aplicações que seus aspectos internos, seu uso parece ser mais indicado a cursos cuja ênfase seja o uso do sistema operacional. Todavia, essa técnica também pode ser usada, de forma complementar, para auxiliar na compreensão das estruturas internas do núcleo, sobretudo no que diz respeito às estratégias de gestão dos recursos compartilhados (CPU, memória e dispositivos de E/S).

4.3. Simulação de algoritmos

O objetivo desta técnica é facilitar a compreensão dos algoritmos comumente encontrados na literatura para a definição das políticas internas do núcleo do sistema operacional. Para isso lança-se mão de programas que simulem algoritmos de escalonamento de CPU, substituição de páginas de memória ou escalonamento de acesso a disco, gerando resultados estatísticos ou animações gráficas para facilitar a compreensão dos mesmos. Um exemplo típico desta técnica é a implementação de um simulador gráfico do escalonamento de processos.

Esta técnica, também conhecida como *visualização de algoritmos*, proporciona oportunidades interessantes de aprendizado na observação e interação com a simulação. Este aspecto é bastante privilegiado pelo uso de tecnologias recentes, como a Internet. Podem ser encontradas muitas simulações de algoritmos típicos de sistemas operacionais implementados como *applets* Java e portanto disponíveis para interação via Internet [Thomson Learning, Inc, 2001].

Alguns autores [Pane et al., 1996, Stasko, 1997] sugerem que os alunos devem atuar não somente no uso das simulações, mas na própria construção das ferramentas de simulação, para que estes adquiram a plena compreensão dos algoritmos simulados e suas implicações. A experiência pessoal deste autor confirma essa hipótese, indicando que a simples observação e interação com as simulações não é suficiente, pois muitos detalhes passam despercebidos. Somente através da implementação dos simuladores os alunos capturam as sutilezas e deficiências dos algoritmos simulados.

Devido às possibilidades gráficas e interativas, esta técnica é bastante atrativa para os alunos e portanto mostra-se bastante efetiva sob o aspecto pedagógico. No entanto, geralmente as simulações tendem a ser excessivamente simplistas e concentrar-se em aspectos isolados do sistema, não contribuindo para a formação de uma visão global e integrada do mesmo. Além disso, a implementação dos simuladores pode se mostrar excessivamente trabalhosa, dependendo da sofisticação gráfica exigida.

4.4. Exploração de código em sistemas reais

Muitos professores consideram que a melhor forma de transmitir conhecimento aos alunos é fazendo-os interagir com o mundo real, através de problemas concretos. No caso de sistemas operacionais, isto poderia ser traduzido através da exploração de núcleos de sistemas operacionais reais e plenamente funcionais. Para tal, podem ser usados sistemas operacionais de produção com código-fonte disponível, como o LINUX ou o FREEBSD. Também podem ser utilizados sistemas operacionais simplificados, mas plenamente funcionais, desenvolvidos especificamente para esse fim, como o MINIX [Tanenbaum and Woodhull, 1999] e o XINU [Comer, 1984].

A análise e modificação do código-fonte de um núcleo estimula o aluno a "mergulhar" no sistema operacional, buscando com isso compreender simultaneamente estruturas, mecanismos, políticas e integração entre as partes. Entretanto, algumas barreiras podem tornar essa abordagem inviável no ensino de graduação:

- Essa abordagem exige uma carga horária de estudo significativa, tanto em classe quanto extra-classe, que pode não estar disponível aos alunos.
- O tamanho do código pode ser um fator complicador, sobretudo em núcleos reais. Por exemplo, o núcleo do LINUX na versão 2.4.10 conta com cerca de 3.3 milhões de linhas de código. Mesmo núcleos simplificados podem ser relativamente volumosos (o MINIX 2.0 tem cerca de 35000 linhas de código).
- Em núcleos reais, a profusão de detalhes de implementação pode tornar a compreensão do código uma tarefa árdua e pouco produtiva em termos didáticos, sobretudo se a documentação disponível for escassa, como é o caso do LINUX. Além disso, os núcleos reais são projetados buscando otimizar seu desempenho, o que pode tornar sua estrutura obscura e complexa. Por exemplo, o código do escalonador de processos do kernel LINUX é bastante difícil de ser interpretado, devido aos aspectos de escalonamento SMP (*Symmetric Multi-Processing*) que nele estão embutidos.
- A observação da dinâmica interna em núcleos reais pode ser uma tarefa complexa. Como o núcleo real executa diretamente sobre o hardware, normalmente não é possível depurá-lo ou executá-lo passo-a-passo, como o permitiria uma aplicação convencional.
- O desenvolvimento em "modo núcleo" impõe dificuldades técnicas aos alunos. O ciclo de desenvolvimento intra-kernel consiste em editar, compilar (geralmente em uma outra máquina), instalar o núcleo, reinicializar o sistema e torcer para que funcione sem falhas. Esse é um processo longo e trabalhoso, que acaba prejudicando os objetivos efetivos da disciplina. Existe a possibilidade de executar o sistema operacional MINIX sobre um software emulador de hardware, como sugerido em [Tanenbaum and Woodhull, 1999], mas essa técnica resolve apenas parcialmente o problema. As etapas de instalação do kernel, reinicialização do

computador e observação externa do funcionamento são bastante simplificadas, mas a depuração do núcleo através de ferramentas permanece um problema.

- Finalmente, o uso de núcleos reais impõe a necessidade de computadores dedicados a esse fim. Entretanto, a realidade habitual em cursos de graduação é o uso de laboratórios compartilhados entre várias disciplinas, ou mesmo entre vários cursos. A instalação de núcleos reais nas máquinas, para experimentação pelos alunos, equivale a atribuir a eles privilégios de administração desses equipamentos. Isso é certamente indesejável do ponto de vista da segurança das instalações, pois os alunos poderiam (inadvertidamente ou intencionalmente) destruir ou corromper os softwares já instalados nas máquinas, prejudicando o andamento das demais disciplinas e gerando um custo administrativo significativo.

Do exposto, conclui-se que esta abordagem é adequada se os alunos tiverem um excelente nível técnico, que permita dominar rapidamente os problemas operacionais e a complexidade interna do núcleo, e se houver a disponibilidade de laboratórios específicos para esse fim. Finalmente, caso a capacidade técnica da turma seja muito heterogênea, esta abordagem pode provocar desistências por parte dos alunos mais fracos, ao invés de estimulá-los a sanar suas deficiências.

4.5. Uso de núcleos simulados

Este método consiste em utilizar um sistema operacional simulado, total ou parcialmente, para demonstrar os conceitos teóricos e explorar sua implementação. Uma grande vantagem do sistema operacional simulado é o completo controle sobre sua execução por parte do aluno, que pode inclusive depurá-lo passo-a-passo em um ambiente controlado. Além disso, por se tratar de um software "convencional" sob o ponto de vista do sistema operacional das máquinas de laboratório, seu uso não impõe as restrições de exclusividade discutidas na seção anterior para o caso dos sistemas operacionais reais. Os principais sistemas operacionais simulados de uso didático encontrados na literatura são os seguintes:

OSP : é um simulador de sistema operacional apresentado e usado em [Kifer and Smolka, 1992]. Não simula um sistema inteiro de forma integrada, mas possui diversos componentes isolados que modelam partes do sistema operacional.

PRMS (*Process and Resource Management Simulator*): um sistema operacional simulado com animação gráfica de algumas de suas estruturas e algoritmos internos, desenvolvido para o ambiente MS-DOS [Hayes et al., 1990].

NACHOS (*Not Another Completely Heuristic Operating System*): desenvolvido na *Berkeley University* por Tom Anderson e outros [Anderson et al., 1993], é um simulador de sistema operacional bastante completo, escrito em C++ para diversos ambientes Unix.

RCOS (*Ron Chernich's Operating System*): um simulador de sistema operacional destinado a demonstrar graficamente os principais conceitos e algoritmos internos do sistema. Ele também permite ao aluno modificar os algoritmos internos e acrescentar novas funcionalidades. Foi escrito em C++ sobre a plataforma MS-DOS [Chernich, 1994, Chernich et al., 1996].

RCOS.JAVA : não se trata de uma simples re-implementação em Java do sistema RCOS, mas de um novo ambiente, reformulado a partir da experiência do autor sobre o sistema anterior [Jones and Newman, 2001].

YALNIX : não é um sistema pronto, mas um projeto de graduação da *Carnegie Mellon University* para construir um sistema operacional completo *from scratch*, a partir de um hardware virtual simplificado [Kesden, 2000].

Um dos problemas enfrentados por esta abordagem é o custo de aprendizado (*learning curve*) associado à mesma, que pode ser significativo [Chernich, 1994]. Além disso, o comportamento do sistema operacional simulado pode não representar exatamente o comportamento de um sistema real, ou pode ser excessivamente simplificado [Whithers and Bilodeau, 1992]. Devido a isso o aluno pode se ver confrontado a um "gap" significativo entre o universo simulado e um sistema operacional real.

5. Conclusão

Este trabalho teve como objetivo lançar um olhar crítico sobre a prática de laboratório no ensino de sistemas operacionais e as ferramentas ali empregadas. Muitas das reflexões aqui apresentadas são fruto da experiência do autor na área, e algumas outras provém de trabalhos correlatos [Whithers and Bilodeau, 1992, Chernich, 1994, Anido, 2000]. Esta discussão resulta do fato da área de sistemas operacionais normalmente representar um desafio ao aprendizado, devido a suas exigências tanto em programação quanto em compreensão dos mecanismos de baixo nível e das abstrações envolvidas [Perez-Davilla, 1995].

Ao longo do texto, houve a preocupação de se definir um contexto pedagógico claro para a análise das práticas de laboratório e para a proposta de ambiente computacional que decorre dessa análise. A opção do autor pela abordagem construtivista justifica-se por ser esta uma abordagem consagrada no ensino das ciências, notadamente no campo da engenharia [Denning, 1989, Duffy and Savery, 1995, Ben-Ari, 1998].

Foram discutidas várias abordagens para as atividades práticas no ensino de sistemas operacionais para a graduação. No entanto, a perspectiva construtivista não considera uma separação clara entre teoria e prática, pois ambas são facetas do mesmo processo de construção do conhecimento. Portanto, para ser completa, esta discussão deveria envolver também as aulas teóricas, numa tentativa de reformular completamente os procedimentos de ensino dessa disciplina. No entanto, esse objetivo exigiria um trabalho bem mais extenso e profundo, que o autor considera fora do escopo do presente trabalho.

As técnicas apresentadas têm características distintas em termos de necessidades (tempo disponível, conhecimento prévio do aluno, disponibilidade de equipamentos) e resultados pedagógicos, além de enfatizar certos aspectos dos sistemas operacionais em detrimento de outros. A tabela 1 apresenta um quadro comparativo que tenta sumarizar os principais benefícios e deficiências de cada uma das abordagens apresentadas.

Observa-se que nenhuma dessas abordagens é auto-suficiente para o ensino efetivo de sistemas operacionais. Segundo a análise do autor, a abordagem que se mostra mais promissora é a de uso de sistemas operacionais simulados. O uso de um sistema operacional simulado que forneça um ambiente rico em interações, com amplas possibilidades de modificação e suficientemente realista constitui uma abordagem privilegiada para o ensino dessa disciplina. Essa abordagem deve ser no entanto complementada em segundo plano pelas demais abordagens.

abordagem	benefícios	deficiências
Uso de abstrações do núcleo	Compreensão do uso da API do núcleo para problemas de programação reais.	Os mecanismos internos do núcleo permanecem obscuros.
Análise por inferência	Compreensão do impacto das políticas do núcleo sobre as aplicações.	A forma como essas políticas são implementadas permanece obscura.
Simulação de algoritmos	Permite compreender os algoritmos internos do núcleo e como eles são implementados.	A observação pode ser simplista e omitir detalhes importantes no mundo real; geralmente a simulação se concentra sobre aspectos isolados.
Exploração de código em sistemas reais	Tem contato com o sistema real e completamente funcional.	Complexidade excessiva, dificuldades de utilização e desenvolvimento, dificuldades operacionais em laboratórios compartilhados.
Uso de núcleos simulados	Simulação da maioria das funcionalidades do núcleo em um ambiente controlado.	Risco de "gap" entre o ambiente e a realidade; curva de aprendizado muito longa.

Tabela 1: Quadro comparativo entre as abordagens apresentadas

A análise aqui apresentada é parte de um projeto maior, que visa construir um novo sistema operacional didático, com base na análise dos sistemas anteriores, buscando aproveitar suas qualidades e sanar algumas de suas deficiências. A definição e construção desse novo sistema operacional, denominado SODA (*Sistema Operacional Didático Aberto*) ainda está em sua fase preliminar e será objeto de publicações futuras. Além do sistema propriamente dito, também é necessário definir claramente a abordagem construtivista empregada na construção e uso do mesmo. Para tanto, deverão ser definidos claramente os projetos a serem desenvolvidos pelos alunos, a exemplo do que foi proposto para o sistema NACHOS [Anderson et al., 1993].

Referências

Anderson, T., Christopher, W., and Procter, S. (1993). The Nachos instructional operating system. In *Proceedings of the Winter Usenix Technical Conference*, pages 481–489.

Anido, R. (2000). Uma proposta de plano pedagógico para a matéria Sistemas Operacionais. In *Anais do II Curso de Qualidade, Workshop sobre Educação em Computação, XX Congresso da Sociedade Brasileira de Computação*, pages 125–148, Curitiba PR.

Ben-Ari, M. (1998). Constructivism in computer science education. *ACM SIGCSE Bulletin*, 30(2):257–261.

Bruner, J. (1966). *Toward a Theory of Instruction*. Harvard University Press.

Bybee, R. and Sund, R. (1982). *Piaget for Educators, 2nd edition*. Columbus OH.

Chernich, R. (1994). The design and construction of a simulated operating system. In *Proceedings of the Asia-Pacific Information Technology in Teaching and Education Conference*, pages 1033–1038, Brisbane, Australia.

Chernich, R., Jamieson, B., and Jones, D. (1996). RCOS: Yet another teaching operating system. In *Proceedings of the 1st Australian Conference on Computer Science Education*, Sidney – Australia.

Comer, D. (1984). *Operating System Design - The XINU Approach*. Prentice-Hall.

Confrey, J. (1990). What constructivism implies for teaching. In *Constructivist Views on the Teaching and Learning of Mathematics – National Council of Teachers of Mathematics*, pages 107–122, Reston Virginia.

Denning, P. (1989). Computing as a discipline. *Communications of the ACM*, 32(1):9–23.

Duffy, T. M. and Savery, J. R. (1995). Problem based learning: An instructional model and its constructivist framework. *Educational Technology*, 35(5):31–38.

Hayes, J., Miller, L., Othmer, B., and Saeed, M. (1990). Simulation of process and resource management in a multiprogramming operating system. In *Proceedings of the 21st ACM Technical Symposium on Computer Science Education*, page 125.

Jones, D. and Newman, A. (2001). RCOS.Java: a simulated operating system with animations. In *Proceedings of the Computer-Based Learning in Science Conference*, Brno – Rep. Tcheca.

Kanuka, H. and Anderson, T. (1999). Using constructivism in technology-mediated learning: Constructing order out of the chaos in the literature. *Radical Pedagogy*, 1(2).

Kesden, G. (2000). The Yalnix kernel project. `http://www-2.cs.cmu.edu/~412/projects/proj3/proj3.pdf`.

Kifer, M. and Smolka, S. (1992). OSP – an environment for operating systems projects. *Operating Systems Review*, 26(4):98–99.

Pane, J., Corbet, A., and John, B. (1996). Assessing dynamics in computer-based instruction. In *CHI'96 Conference Proceedings, ACM Press*, New York.

Perez-Davilla, A. (1995). OS – bridge between academia and reality. *ACM SIGCSE Bulletin*, 27(1):146–148.

Piaget, J. (1932). *The Moral Judgement of the Child*. NY Harcourt.

Stasko, J. (1997). Using student built algorithm animations as learning aids. In *Proceedings of the 28th Technical Symposium on Computer Science Education*, pages 25–29.

Tanenbaum, A. and Woodhull, A. (1999). *Sistemas Operacionais: Projeto e Implementação, 2ª edição*. Editora Bookman.

Thomson Learning, Inc (2001). A simple CPU simulator applet. `http://www.brookscole.com/compsci/aeonline/course/7/5/index.html`.

Weisz, T. (1999). *O Diálogo entre o Ensino e a Aprendizagem*. Editora Ática.

Whithers, J. and Bilodeau, M. (1992). An examination of operating systems laboratory techniques. *ACM SIGCSE Bulletin*, 24(3):60–64.