Uma Plataforma de Aprendizado Baseado em Projeto para Ensino e Treinamento em Sistemas Operacionais

Renê S. Pinto[1], **Francisco J. Monaco**[1]

[1]ICMC – Universidade de São Paulo – Campus de São Carlos,
Caixa Posta 668, 13560-970 São Carlos, SP, Brazil

rene@icmc.usp.br, monaco@icmc.usp.br

Resumo. *No atual cenário de inovações em sistemas embarcados, além da tradicional função de formação teórica, a disciplina de Sistemas Operacionais refocaliza o conhecimento técnico em projeto e desenvolvimento, e reemerge como uma capacitação crescentemente relevante em uma indústria em expansão. O ensino em SOs sob essa nova perspectiva constitui um desafio frente sua inerente complexidade e multidisciplinaridade. Este artigo apresenta uma nova plataforma para ensino e treinamento no desenvolvimento de SOs fundada em uma metodologia de aprendizagem baseada em projeto, a qual guia o estudante através do processo de projetar e programar um completo sistema operacional, simples, porém funcional e com os recursos essenciais de um SO convencional. Uma comparação com alternativas de finalidade educacional revela uma estrutura de implementação substancialmente menos complexa e que mapeia módulos conceituais de aprendizagem aos blocos de implementação através de uma correspondência direta e intuitiva, e com menor acoplamento de código e funções, o que propicia um programa passo-a-passo sem percursos de retroação e saltos entre os tópicos de instrução.*

1. Introdução

Depois da emergência dos SOs nos anos 60 e 70, a hegemonia de algumas poucas plataformas de hardware e seus sistemas operacionais proprietários, por mais de uma década, relegou o escopo da disciplina a um significado mais ligado à formação teórica básica, e menos à capacitação para atuação profissional no campo específico. Tal realidade, todavia, está rapidamente se alterando. A disseminação do modelo *open source* e a evolução de arquiteturas embarcadas capazes de executar SOs sofisticados recriam o cenário de crescente demandas de especialistas para atuarem diretamente no desenvolvimento de Sistemas Operacionais. A introdução dessas aplicações em sistemas críticos contribui ainda mais para essa perspectiva, tendo em vista o campo que se abre para pesquisa e desenvolvimento em Sistemas Operacionais, para novas arquiteturas e aplicações específicas, com restritos requisitos de desempenho, confiabilidade e eficiência.

Ao lado da abordagem teórica sobre os paradigmas e problemas relevantes para a compreensão do funcionamento de um Sistema Operacional, um recurso que vem sendo utilizado em cursos mais avançados é o de oferecer ao aluno a oportunidade de atingir esse objetivo através do desafio concreto de intervir diretamente na implementação de um SO, seja exercitando algoritmos em simuladores,

seja efetuando alterações exploratórias em código-fonte exemplo. Existem alguns casos reais de implementações dessas práticas de laboratório que exploram exercícios onde o aprendiz é convidado a estudar, manipular, alterar e ampliar diretamente o código-fonte de um sistemas operacional, tendo a oportunidade de consolidar suas noções teóricas e entender aspectos técnicos que as complementam. Exemplos amplamente conhecidos dessa abordagem incluem o sistemas operacionais MINIX [Tanenbaum 2000], originariamente desenvolvido com finalidades didáticas, e GNU/Linux, crescentemente utilizado como estudo de caso em função de sua relevância e licença Livre. Embora produtiva, a experiência de alterar partes de um SO é ainda limitada a uma visão fragmentada do tema, não ressaltando as relações e interdependências entre os tópicos estudados. Este trabalho tem em vista ampliar as possibilidades desse tipo de prática didática, conduzindo o aluno através de um programa de aprendizado baseado em projeto, exercitado mediante a construção de um sistema operacional completo, desde as primeiras linhas de código até os subsistemas essenciais, fornecendo uma experiência prática e efetiva das competências e habilidades necessárias para atuação profissional na área. Não obstante o vasto e riquíssimo material disponível online, este encontra-se substancialmente esparsos em artigos independentes, relatos informais e exemplos específicos. O presente trabalho introduz uma compilação sistematizada desses conhecimentos em um programa de instrução estruturado.

1.1. Objetivos

Este artigo apresenta uma plataforma de ensino e treinamento no desenvolvimento de Sistemas Operacionais fundada na abordagem de aprendizado baseado em projeto (*project-based learning*). A plataforma é constituída de uma especificação de arquitetura de SO cuja estrutura em nível de implementação mapeia uma correspondência simples e coerente com um roteiro de conteúdo instrucional baseado em módulos teóricos sequencialmente conectados. A arquitetura de SO, portanto, deve minimizar o acoplamento funcional entre módulos e orientar-se pela construção/desenvolvimento incremental do código seguindo uma sequência intuitivamente ligada ao material didático. Fazem parte da plataforma uma clara documentação que explica a utilização e construção passo-a-passo, na forma de um tutorial e documentos de referência, de um Sistema Operacional básico para Arquitetura Intel IA-32 (x86 32 bits), além de todo software de suporte para configuração, compilação e edição do código-fonte.

2. Trabalhos Relacionados

As ementas dos cursos de Berkeley e do MIT, tomadas como referências importantes, apresentam a teoria de SO abordando a arquitetura x86 e sua programação, arquitetura PC, sistemas de arquivos de alta performance, Multics e o UNIX, Minix, máquinas virtuais, etc. A parte prática também focaliza kernels (núcleos) de sistemas operacionais reais como o Minix e sistemas/simuladores de intuito educacional como o Nachos[nac 2010], específico para disciplinas de SO. O Minix é o tema do livro-texto clássico em cursos de computação "Operating System Design and Implementation"[Tanenbaum 2000] (primeira edição em 1987). Além desse, já há alguns anos existem livros de teoria de SO que elaboram os tópicos com base no código-fonte do sistema Linux[O'Gorman 2001].

Eqnaunto abordagens teóricas são mais frequentes, experiências práticas descritas na literatura são realizadas com o auxílio de SOs de intuito educativo. Diversos exemplos são baseados em simulação, i.e. em algoritmos que são avaliados em um emulador de arquitetura de hardware. É o caso, por exemplo, do Nachos[Christopher et al. 1993], um SO educacional desenvolvido na Universidade da Califórnia, Berkeley. O núcleo do Nachos roda sobre outro SO, simulando uma arquitetura MIPS R2000/3000.Na utilização descrita, os estudantes recebem apenas um esqueleto do código, e durante as práticas vão desenvolvendo as funcionalidades não implementadas. Baseado no Nachos, o OS/161[Holland et al. 2002], desenvolvido na Universidade Harvard visa superar algumas de suas limitações. O OS/161 é executado em um simulador chamado *System/161*, que emula um processador MIPS R2000 e mais alguns dispositivos de hardware tais como discos e portas seriais. Os alunos desenvolvem práticas que incluem sincronização, chamadas de sistema e sistema de arquivos.

Dentre os sistemas operacionais educativos não-emulados, i.e. que são executados diretamente pelo hardware real, encontram-se diversos exemplos de SO para arquiteturas embarcadas. O Xinu[Carissimo 1995] foi desenvolvido na década de 1980 na Purdue University; apesar de ser tema de dois livros textos, atualmente não esta mais em desenvolvimento[Anderson and Nguyen 2005]. O Topsy[Fankhauser et al. 1996] é um Sistema Operacional baseado em *microkernel* desenvolvido no Swiss Federal Institute of Technology in the Computer Engineering and Networks Laboratory. Segundo [Anderson and Nguyen 2005], seu uso parece estar restrito apenas alguns campi europeus. O Topsy roda em um processador MIPS R3000, tanto no hardware físico quanto em simuladores. Para plataformas mais elaboradas, como a da arquitetura PC convencional existe um dos mais famosos projetos de Sistemas Operacionais educacionais: o Minix[Tanenbaum 1987], um clone do Unix escrito por Andrew S. Tanenbaum entre 1984 e 1897.

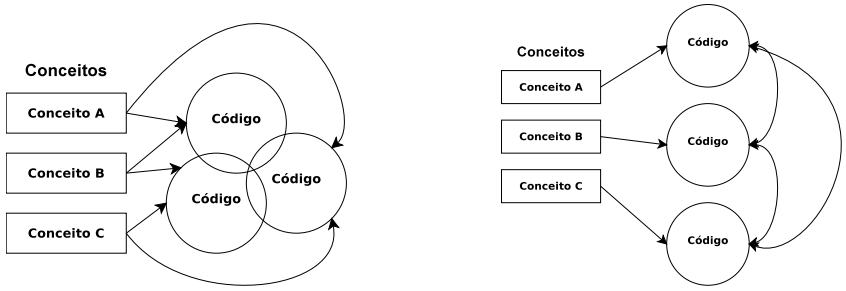
No conjunto dos casos estudados, a utilização de sistemas operacionais em cursos de graduação baseia-se ou em simulação, ou na modificação de SOs existentes, ou na implementação pontual de funções específicas.

3. A plataforma TempOS

Uma dificuldade em conduzir-se um curso de Sistemas Operacionais através de programação passo-a-passo de um código completo é que as instanciações das diversas funcionalidades, em um sistema complexo, podem ser realizadas de modo "fragmentado" em diversas partes do código fonte, ao invés de serem implementadas de modo coeso dentro de módulos bem distintos em termos construtivos. Questões referentes ao desempenho, aspecto essencial aos SOs, podem recomendar decisões arquiteturais que resultem em uma forte interdependência de diferentes componentes, de maneira que determinada função é realizada por interações complexas entre muitos procedimentos e funções, que compartilham estados. Se por um lado é possível aplicar conceitos de modelagem orientada a objetos no projeto arquitetônico, a implementação real ainda tem de ser mapeada na visão de sistema em baixo nível, o que significa interagir com a arquitetura de hardware e suas próprias abstrações – nem sempre bem encapsuladas. Assim, é em geral difícil elaborar um roteiro de aulas que mapeie conceitos às respectivas implementações de modo simples e que refiram

o conteúdo abordado a partes auto-contidas do código fonte (Figura 1(a)).

A estratégia da plataforma de ensino proposta, na qual consiste o diferencial de modelo adotado, baseia-se na concepção e especificação de uma arquitetura de SO em nível de implementação na qual os módulos componentes são projetados de modo a minimizar o acoplamento de código fonte, e a apresentar uma correspondência simples e direta com os tópicos conceituais destacadas (Figura 1(b)).



(a) Alta interdependência entre conceito e implementação.

(b) Menor interdependência entre conceito e implementação.

Figura 1. Característica chave da estratégia da arquitetura do TempOS

Idealmente seguindo um programa institucional com tópicos sequenciais, esse princípio facilita o estudo do código fonte em trechos concisos e de menor interação com outras partes, especialmente com aquelas a serem abordadas em etapas posteriores do roteiro de estudo. Isso também é verdade para a tarefa de implementação se a arquitetura completa puder ser construída incrementalmente, a partir de partes que se agregam ao sistema, minimizando o esforço de recorrer à fragmentos de código espalhados pelo programa todo, e não intuitivamente conexos.

A plataforma proposta compõe-se do sistema-modelo, das especificações e da descrição do método de ensino e treinamento aplicáveis a seu uso na prática. De posse desse recursos, o aluno deve ser capaz de reimplementar ele mesmo toda arquitetura, tendo a possibilidade de exercitar seus conhecimentos incorporando modificações e extensões de funcionalidade. São requisitos gerais dos módulos acima relacionados o desenho arquitetural simples e compreensivo, código legível e didático, documentação precisa e completa.

3.1. Componentes da Plataforma

A plataforma, denominada TempOS, é composta pelos seguintes componentes:

- Especificação de uma arquitetura completa de um Sistema Operacional funcional, de kernel monolítico e baseado no Unix, para arquitetura IA-32 (x86), compondo-se de suas diversas partes:
  - Kernel do sistema, que fornece a API interna para os drivers e externa (chamadas ao sistema, seguindo padrão POSIX), além de gerenciar todos os recursos básicos da máquina.
  - Device drivers para dispositivos como teclado, discos IDE, e periféricos específicos, como o temporizador (PIT) e controlador de interrupções (PIC).

- Código fonte do Sistema Operacional da especificação, exemplificando abordagens de projeto e técnicas de implementação para cada um dos componentes do sistema.
- Ambiente de desenvolvimento, configurado e otimizado, incluindo software de edição de código e depuração (*debugging*), emulador, ferramentas de teste.
- Roteiro na forma de tutorial passo-a-passo dividido em distintas práticas de laboratório que, recorrendo ao modelo, conduza o aluno através do processo de construção de um sistema operacional completo e funcional, desde as primeiras linhas de código até os módulos mais complexos incluindo subsistemas de carregamento, acesso ao hardware, gerenciamento de memória e demais funções de alto nível. A sequência de aula é projetada de modo a servir como base para um programa de disciplina de graduação em cursos de computação.

O modelo é inicialmente baseado na plataforma PC, mas tendo em mente a portabilidade para outras arquiteturas de hardware, em vista da crescente relevância das arquiteturas embarcadas e a demanda por profissionais capacitados ao desenvolvimento de sistemas operacionais para tecnologias de hardware emergentes e requisitos críticos. O material didático ainda encontra-se em elaboração, entretanto, toda a especificação e mais o código do SO da plataforma já foram desenvolvidos. O SO implementado foi denominado TempOS (TempOS is an educational and multi purpose Operating System).

3.2. Arquitetura da Plataforma

A arquitetura do TempOS (Figura 2) corresponde a arquitetura do Unix mostrada em Bach (1986)[Bach 1986] com pequenas adaptações. Assim, o TempOS compreende um núcleo monolítico que segue o padrão POSIX de chamadas ao sistema. A camada VFS (Virtual File System) permite o suporte a vários sistemas de arquivos, sendo baseada na camada VFS do Linux.

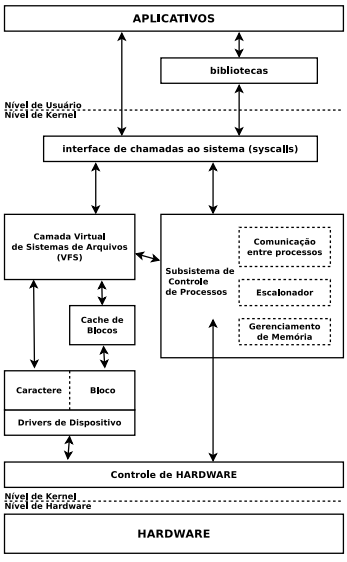


Figura 2. Arquitetura da plataforma TempOS. Figura adaptada de [Bach 1986].

A arquitetura é composta por seis principais módulos:

- **Interface de chamadas ao sistema**: Implementa o mecanismo de chamada ao sistema (atendimento da interrupção, transição entre kernel/userspace e vice-versa) e as chamadas ao sistema POSIX.
- **Camada Virtual de Sistema de Arquivo**: Implementa a camada que permite o suporte a vários sistemas de arquivos através da abstração de um sistema de arquivos comum, baseado no EXT2. O suporte a determinado sistema de arquivo é feito através de um drivers, contidos neste módulo, que também implementam funções internas for drivers do kernel.
- **Cache de blocos**: Responsável por fazer cache de blocos de dispositivos de armazenamento, como por exemplo discos rígidos, otimizando a leitura/escrita dos blocos.
- **Drivers de dispositivo (caractere / bloco)**: Contém os drivers para dispositivos de caractere e bloco.
- **Controle de Hardware**: Contém drivers de dispositivos de hardwares mais específicos, como por exemplo o controlador de interrupções (PIC) e temporizador (PIT).
- **Subsistema de controle de processos**: Este módulo contém o gerenciador de memória de alto nível, responsável por alocar memória para os processos de usuário e de kernel, o escalonador, que implementa o chaveamento de processos baseado em uma política de escalonamento e funções para comunicação inter-processos, como semáforos, filas de mensagens e pipes.

3.3. O Sistema Operacional TempOS

Um dos elementos centrais da plataforma é o sistema operacional *TempOS* (acrônimo recursivo de *TempOS is an educational and multi purpose Operating System*). O intuito do seu design projeto é apresentar um sistema operacional muito simples do ponto de vista estrutural, mas completamente funcional, exaustivamente documentado maneira ser possível sua utilização como toolkit em um curso de desenvolvimento de sistemas operacionais.

O *TempOS* foi escrito para a arquitetura IA-32 (x86 32 bits), 93,33% em Linguagem C, sendo o Assembly (sintaxe AT&T) utilizado onde estritamente imprescindível para partes dependentes de arquitetura. Com um robusto sistema de compilação (desenvolvido através de scripts atuando em conjunto com o utilitário *make*), o *TempOS* pode ser compilado e testado muito facilmente a partir de qualquer sistema GNU/Linux juntamente com o emulador QEMU[1], ou executando-o diretamente em um PC comum. O código está extensivamente comentado (os comentários representam 50,19% dos fontes) e contém referências para que o usuário possa pesquisar e correlacionar conceitos teóricos com a implementação em código, incluindo referências para os manuais da arquitetura IA-32 (disponibilizados na internet gratuitamente pela Intel).

O código fonte foi estruturado tendo em mente seu desacoplamento em relação aos tópicos teóricos envolvidos no código. Assim, buscou-se uma organização que tenta alinhar cada arquivo de código fonte com um ou poucos conceitos teóricos, e que estejam diretamente relacionados, facilitando o compreendimento pelo

[1]Open source processor emulator, disponível http://www.qemu.org

estudante. A Figura 3 mostra a organização dos fontes em relação a cada módulo da arquitetura. Os arquivos estão indicados por elipses, diretórios pelo polígono e os módulos da arquitetura pelos retângulos. O diretório *arch* contém todo código dependente de arquitetura. Outros arquivos não ligados diretamente aos módulos foram omitidos para melhor visualização.

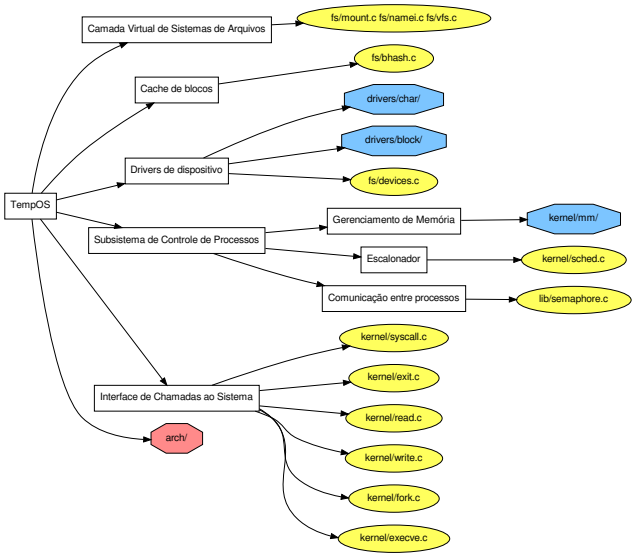


Figura 3. Organização do código-fonte do TempOS com correlação entre módulos.

O código fonte está disponível no sítio *http://tempos-project.org* sob a licença GPL, sendo totalmente modularizado, isolando-se a parte dependente da Arquitetura x86, de modo a permitir que o *TempOS* seja portado para outras arquiteturas sem grande dificuldade, inclusive para plataformas embarcadas. O *TempOS* possui um sistema de boot multiestágio (via GRUB), funções da biblioteca C, realocação dinâmica do kernel (linkado em 3GB), modelo de memória plana, compartilhamento de IRQs, drivers para controladores de teclado, PATA (IDE), chamadas ao sistema POSIX, Camada Virtual de Sistema de Arquivo, suporte parcial ao sistema de arquivos EXT2, suporte completo a tabela de partições (primárias e extendidas), threads de kernel, escalonador com política *round robin* e demais recursos fundamentais em SOs de uso real. Depois de inicializar, o *TempOS* carrega o programa *init* para a memória e passa a executá-lo em espaço de usuário. A maioria da funcionalidade encontra substancialmente desenvolvida, sendo o atual trabalho concentrado na implementação das system calls, sistemas de arquivos e de console. O código é documentado com o auxílio do software doxygen[2] conversível para HTML ou PDF.

O *TempOS* segue a especificação da arquitetura. A partir desta, o material de instrução (em desenvolvimento) descreve o processo pelo qual o SO-exemplo é desenvolvido. O roteiro de estudo é elaborado como uma sequência de tópicos conceituais logicamente encadeados, referenciando os exercícios de implementação. Seguindo esses passos, o aluno é guiado pelo processo iterativo de desenvolvimento do SO, passo a passo, evoluindo incrementalmente o projeto, sendo a estrutura

[2]Disponível em http://www.doxygen.org

geral da arquitetura consistente com as arquiteturas de SOs convencionais para PC, monolítica, inspirada no modelo Unix/Linux e padrão POSIX.

4. TempOS X Linux X Minix

Linux e Minix são dois SOs utilizados em cursos de abordagem mais prática em Sistemas Operacionais. Entretanto, a versão 2.6.30 do kernel do Linux contém mais de 11 milhões de linhas de código[Greg Kroah-Hartman 2009] e o Minix, a partir da versão 3.0 deixou de ter intuitos educacionais[3]. Mesmo assim, versões anteriores (e menores) ainda podem ser utilizadas.

Para investigar a complexidade do TempOS, foi feita uma comparação entre os códigos da versão 0.99.15 do Linux e a versão 2.0.0 do Minix (que ainda tinha cunho educional). É importante notar que ambos os sistemas, mesmo em suas versões mais modestas, possuem maiores funcionalidades que o TempOS, por exemplo como a maior quantidade de drivers de dispositivo e pilha TCP/IP. Portanto, para efetuar uma comparação justa, diversos arquivos de código do Linux e Minix foram removidos, tais como arquivos de drivers de dispositivos e sistemas de arquivos não suportados pelo TempOS, sistemas de rede (pilha TCP/IP) e arquivos de chamadas ao sistemas ainda não implementados no TempOS. Como o Minix possui uma arquitetura microkernel (diferente da arquitetura monolítica do TempOS e do Linux), portanto, na análise foi considerado o código do microkernel, gerenciador de memória e sistema de arquivo. Demais serviços, biblioteca C, etc, não foram considerados na análise. O software LocMetricts[4] foi utilizado para fazer a aferição do código. Foram considerados dois parâmetros para a análise: Logical sources lines of code (SLOC-L) e McCabe Ciclomatic Complexity (MVG).

Há diversas definições para computar a métrica SLOC-L. O LockMetrics computa esse valor somando os ponto-e-virgulas e chaves terminais das linhas de código-fonte. A Complexidade Ciclomática McCabe (MVG)[McCabe 1976] é um método clássico para medir a complexidade de software, mediante a contagem do número de fluxos através de um trecho de código. Cada vez que um desvio ocorre (`if, for, while, case`, etc.), a métrica é incrementada. A tabela 1 contém os valores obtidos através do LocMetricts para o TempOS, Linux e Minix.

Tabela 1. Resultados do LocMetrics

	TempOS	Linux 0.15.99	Minix 2.0.0
SLOC-L	3954	14538	17309
MVG	663	4462	3910

A figura 4(a) apresenta o gráfico do número total de linhas de código (incluindo comentários e linhas em branco) e o número total de linhas de comentário para cada um dos Sistemas Operacionais analisados. A porcentagem indica a quantidade total de comentários em relação a todo código fonte. Nos códigos analisados, o Minix foi o maior kernel, com 52667 linhas de código, seguido pelo Linux, com 39364

[3]Afirmação feita no FAQ do site oficial do Minix. Disponível em http://www.minix3.org/other/readmore.html

[4]Disponível em http://www.locmetrics.com

e o TempOS, com 13556 linhas de código. Quanto aos comentários, o código do TempOS foi o que apresentou a maior porcentagem, sendo 50,19% de todo código fonte somente em comentários, seguido pelo Minix, com 19,40% e Linux, com 17,24%. A figura 4(b) apresenta o gráfico das quantidade de linhas escritas em Linguagem C e Assembly, e suas respectivas porcentagens em relação a todo código fonte. Os valores foram obtidos através do software *SLLOCCount*[5].

(a) Número de Linhas de Código e Comentários.

(b) Linguagens de Programação utilizadas.

Figura 4. Alguns dados comparativos do código-fonte

O Linux apresentou a maior porcentagem de código C (94,32%), seguido pelo TempOS (93,33%) e Minix (88,50%). Quanto ao código Assembly, o Linux apresentou apenas 5,68% de código escrito em Assembly, entretanto, este valor corresponde a 1549 linhas de código. Já o TempOS apresentou 6,67% de código Assembly, o que corresponde a apenas 416 linhas. O Minix apresentou 11,50% correspondentes a 3556 linhas de código.

5. Conclusões

Este artigo introduz uma plataforma para ensino de SO cujo diferencial é a metodologia de aprendizagem baseada em projeto e uma correspondente implementação exemplo. Esta, denominada TempOS, é estruturalmente mapeada em um programa de treinamento em desenvolvimento, cuja característica chave é o isolamento dos conceitos em blocos concisos de implementação, referenciando os módulos de ensino sequenciais com pouca interdependência.

Uma versão anterior do TempOS, ainda menos desenvolvida do que a apresentada neste artigo, foi utilizada um experimento piloto realizado em ano anterior, na condução da disciplina de Sistemas Operacionais II, para o curso de Engenharia de Computação ICMC/EESC USP. A iniciativa teve como objetivo avaliar a estratégia e angariar experiência para seu desenvolvimento subsequente. Uma turma de cerca de 15 alunos completou o programa, que abrangeu o desenvolvimento de 60% da plataforma, da qual foram excluídos os subsistemas ainda incompletos na ocasião. Dentre os resultados anotados destacam-se, dentre as dificuldades, a elevada carga de trabalho propostas aos estudantes, resultante das lacunas no material didático ainda em elaboração, com ênfase para as ferramentas de desenvolvimento e especificidades da arquitetura de hardware — a constatação motivou um aprofundamento nessa área que, potencialmente, sugere que o material possa ser bem empregado em uma abordagem integrada com um curso de organização de computadores. Dentre

[5]Disponível em http://www.dwheeler.com/sloccount/

as conclusões positivas, anota-se que a turma participante do experimento, inicialmente cética quanto à própria competência para desenvolver um SO, completou o programa com sucesso e considerando-se apta a uma reedição mais completa do desafio — o que corrobora a expectativa assinalada nos objetivos do projeto. Uma avaliação mais objetiva da aplicação da plataforma será possível no prazo de alguns meses, quando o material estiver finalizado.

Agradecimentos

Os autores agradecem à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), ao Conselho Nacional de Pesquisa e Desenvolvimento (CNPq), à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), ao Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos INCT-SEC, e à Universidade de São Paulo.

Referências

(2010). Nachos operating system. http://www.cs.washington.edu/homes/tom/nachos/.

Anderson, C. L. and Nguyen, M. (2005). A survey of contemporary instructional operating systems for use in undergraduate courses. *J. Comput. Small Coll.*, 21(1):183–190.

Bach, M. J. (1986). *The design of the Unix Operating System*. Prentice-Hall.

Carissimo, J. (1995). XINU–an easy to use prototype for an operating system course. *ACM SIGCSE Bulletin*, 27(4):56.

Christopher, W., Procter, S., and Anderson, T. (1993). The Nachos instructional operating system. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, page 4. Usenix Association.

Fankhauser, G., Conrad, C., Zitzler, E., and Plattner, B. (1996). Topsy–a teachable operating system. *Computer Engineering and Networks Laboratory, ETH Zurich*, 2001.

Greg Kroah-Hartman, Jonathan Corbet, A. (2009). Who writes linux: How fast it is going, who is doing it, what they are doing, and who is sponsoring it: An august 2009 update. Technical report, Linux Foundation.

Holland, D., Lim, A., and Seltzer, M. (2002). A new instructional operating system. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 111–115. ACM.

McCabe, T. (1976). A complexity measure. *Software Engineering, IEEE Transactions on*, (4):308–320.

O'Gorman, J. (2001). *Operating Systems with Linux*. Palgrave Macmillan.

Tanenbaum, A. (1987). A UNIX clone with source code for operating systems courses. *ACM SIGOPS Operating Systems Review*, 21(1):29.

Tanenbaum, A. S. (2000). *Sistemas Operacionais: projeto e implementação*. Bookman.