BananaKernel: Um Sistema Operacional para Ensino

Rafael Coninck Teigão[1], **Julio Henrique Morimoto**[1] **e Carlos Maziero**[1]

[1]Bacharelado em Ciência da Computação
Centro de Ciências Exatas e de Tecnologia
Pontifícia Universidade Católica do Paraná

{rcteigao, juliohm}@gmail.com, maziero@ppgia.pucpr.br

Abstract. *The purpose of this project is to develop a didactical Operating System to provide students an environment for learning concepts such as Memory Management, Process Scheduling, and so on. Its structure should allow students to easily develop and integrate new modules into the system and/or substitute existing ones.*

Resumo. *O propósito deste projeto é desenvolver um Sistema Operacional didático para proporcionar aos estudantes um ambiente para o aprendizado de conceitos como Gerência de Memória, Escalonador de Processos e outros. Sua estrutura deve permitir aos estudantes facilmente desenvolver e integrar novos módulos ao sistema e/ou substituir os existentes.*

1. Introdução

No meio acadêmico existe a necessidade de um ambiente que facilite o ensino de conceitos de sistemas operacionais para alunos, principalmente dos cursos de Bacharelado em Ciência da Computação e Engenharia de Computação [Maziero 2002]. Esse ambiente deve possuir uma documentação *on-line*, que apresente o funcionamento interno do sistema, e um manual do usuário, que possibilite alunos e professores utilizarem e alterarem o sistema facilmente.

O *BananaKernel* é uma ferramenta para auxiliar no ensino da disciplina de Sistemas Operacionais, demonstrando alguns dos mecanismos internos de funcionamento de um SO moderno, com documentação em português e um manual do usuario [Teigão and Morimoto 2004a] que permite uma fácil utilização e modificação dos seus módulos.

Com o intuito de proporcionar um entendimento gradual do funcionamento interno de um sistema operacional, duas versões foram criadas, sendo chamadas de *simples* e *intermediária*. Na versão simples, procurou-se evitar otimizações no uso dos recursos disponíveis no sistema (e.g. o gerenciador de memória faz alocações contíguas e seqüenciais). Já na versão intermediária, algumas otimizações foram utilizadas (e.g. o gerenciador de memória utiliza um sistema de lista encadeada de blocos disponíveis, permitindo a escolha do melhor bloco na alocação).

Este artigo descreve a concepção e implementação de um sistema operacional didático. O texto está dividido em 10 seções. A seção 2 apresenta um diagnóstico atual do ensino de sistemas operacionais; a seção 3 discorre sobre a metodologia aplicada ao

desenvolvimento do sistema; as seções 4, 5 e 6 detalham as implementações do gerenciador de memória, gerenciador de arquivos e escalonador de processos; a seção 7 relata os testes efetuados com os alunos, em sala de aula; nas seções 8 e 9 são sugeridos trabalhos futuros e apresentados trabalhos correlatos, respectivamente; a seção 10 contém a conclusão do artigo.

2. Diagnóstico Atual

Muitos professores de Sistemas Operacionais recorrem a implementações para demonstrar alguns dos conceitos aprendidos em sala. Porém, essa forma de aprendizado encontra vários problemas, entre eles:

- falta de uma documentação clara que auxilie os alunos;
- falta de um ambiente dedicado para a implementação, o que obriga os alunos a lidarem com sistemas completos e muito mais complexos;
- muitas vezes o SO base não fornece mensagens suficientemente didáticas de erro e funcionamento ao aluno;
- dificuldades para resolver problemas, uma vez que não se tem controle do SO em que se implementa.

Com isso, o desenvolvimento de uma simples biblioteca de *threads*, por exemplo, demanda muito tempo dos alunos, torna-se uma tarefa extremamente dependente do SO base, e fornece conhecimento apenas superficial do processo.

Um SO didático, que permita uma fácil troca de módulos e seja bem documentado, aceleraria este processo de desenvolvimento, ao fornecer um ambiente de desenvolvimento propício; melhoraria o controle sobre os módulos, pois a complexidade do SO base seria muito menor; e auxiliaria no aprendizado do aluno através de informações úteis na tela.

3. Metodologia de Desenvolvimento

O ambiente aqui proposto foi desenvolvido utilizando como base o sistema *Flux OSKit* [Ford et al. 1997], composto por um conjunto de bibliotecas que auxiliam a criação de um sistema operacional. O *OSKit* permitiu que o desenvolvimento fosse concentrado nos conceitos fundamentais de gerência de memória, escalonador de processos e gerência de arquivos, evitando o desenvolvimento de código de *boot* e *device drivers*. O compilador e o depurador utilizados foram o *GNU project C and C++ Compiler* (GCC) e *GNU Debugger* (GDB), respectivamente.

Para cada módulo do sistema, foram preparados algoritmos [Teigão and Morimoto 2004c] e casos de teste [Teigão and Morimoto 2004b], que foram implementados separadamente e depois unidos em um único sistema, em correspondência ao requisito de modularidade. A figura 1 ilustra a relação entre o *BananaKernel* e os demais sistemas.

Após a implementação, foram criadas imagens de disco contendo o *kernel* de cada uma das versões, preparadas com os casos de teste, e executadas no emulador *Bochs* [Lawton et al. 2004].

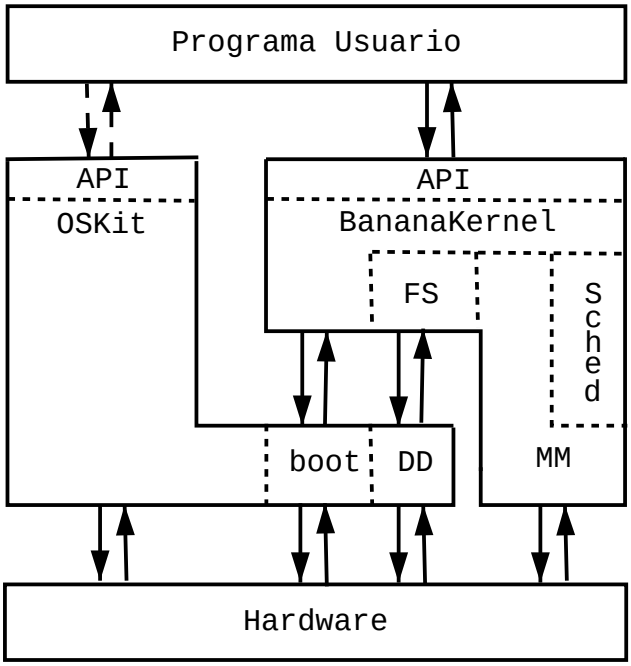

Figura 1. Interfaces do *BananaKernel*

3.1. Escolha da linguagem de programação

A linguagem de programação escolhida foi o ANSI C [ANSI 1989]. Foi dada preferência a uma linguagem imperativa, ao invés de uma orientada a objetos, como C++, porque uma linguagem orientada a objetos envolve incertezas oriundas dos mecanismos de resolução dinâmica de sobrecargas e polimorfismo. Essas incertezas são indesejáveis no âmbito de um núcleo de sistema operacional, que deve possuir um comportamento tão determinístico e previsível em tempo de compilação quanto possível.

Deve-se notar, também, que muitos cursos superiores de computação dão grande ênfase à programação em C, então esta linguagem foi escolhida por ser mais familiar para a maioria dos alunos.

4. Gerenciador de Memória

O gerenciador de memória fornece as funções utilizadas para alocar, realocar e liberar blocos de memória, e também para mostrar a quantidade de memória disponível e o tamanho de um bloco.

`bnnk_malloc()` – alocação de memória;
`bnnk_realloc()` – re-alocação de memória;
`bnnk_free()` – liberação de uma parte da memória;
`bnnk_avail()` – informa a quantidade de memória disponível;
`bnnk_getsize()` – informa o tamanho de um volume de memória reservado e
`bnnk_memstat()` – apresenta na tela uma barra mostrando a situação atual de preenchimento dos blocos de memória.

4.1. Detalhes da versão simples

A versão simples do gerenciador de memória esta baseada em um esquema de alocação contígua, em que o próximo bloco alocado estaá localizado ao final do último bloco.

Para que o endereço tenha um crescimento positivo para o usuário, ou seja, para que uma posição no bloco possa ser expressa como $endereco + offset$, o sentido da alocação deve ser do maior endereço disponível para o menor. Sendo assim, o endereço de um novo bloco pode ser encontrado pela função `bnnk_malloc()` por:

$$novo\_endereco = endereco\_ultimo\_bloco - tamanho\_requerido$$

Cada bloco alocado é representado por uma estrutura contendo sua posição e tamanho, armazenada em um vetor de tamanho fixo (1024), o que limita a quantidade de blocos que podem ser alocados simultaneamente. Como nesta versão não há gerência de blocos liberados e de fragmentos de memória, um bloco alocado não pode ser retirado da memória, mas ele é removido deste vetor.

4.2. Detalhes da versão intermediária

Como esta versão gerencia blocos livres e alocados de memória, é possível devolver o tamanho de um bloco para ser reutilizado pelo sistema. Assim como na versão simples, a versão intermediária possui um vetor de tamanho fixo para os blocos alocados, mas conta também com um vetor contendo os blocos livres.

Inicialmente, o sistema possui um único bloco livre, cujo tamanho é equivalente a toda memória disponível. Quando o usuário solicita um bloco, a função `bnnk_malloc()` procura o primeiro bloco disponível com tamanho igual ou maior que o tamanho solicitado. Então este bloco é fracionado, caso seja maior, e o tamanho solicitado é retirado da memória disponível.

Quando o bloco é liberado, a função `bnnk_free()` verifica se os blocos imediatamente anterior ou posterior estão livres e, em caso afirmativo, ele faz a junção desses blocos, criando um espaço livre maior.

4.3. Testes efetuados

O caso de teste do gerenciador de memória procurou verificar sua capacidade de alocar blocos de memória com permissão de escrita, re-alocar blocos já preenchidos com dados, efetuar cálculos para encontrar o volume disponível de memória, calcular o tamanho de um bloco e liberar blocos alocados.

Todas essas funções foram executadas sem apresentar erros, e o conteúdo utilizado para preencher a memória manteve-se intacto após todas as operações.

5. Gerenciador de Arquivos

O gerenciador de arquivos fornece as funções comuns de acesso a arquivos, como *open*, *read*, *write* e *close*, e também funções para formatar uma imagem de disco.

`bnnk_open()` – abre um arquivo em disco;
`bnnk_read()` – faz a leitura de um arquivo aberto no disco para a memória;
`bnnk_write()` – escreve em um arquivo aberto;
`bnnk_close()` – fecha o arquivo aberto em disco; e
`bnnk_fs_format()` – formata o disco de acordo com o sistema de arquivos com alocação contígua (apenas na versão simples).

5.1. Detalhes da versão simples

Nesta versão, não há gerência de blocos, os arquivos são escritos de forma contígua e seqüencial no disco, sendo então impossível utilizar um formato de alocação pré-existente (como *FAT*, por exemplo). Para que seja possível utilizar um disco, deve-se antes formatá-lo. Na formatação, é criada uma área no início do disco que contém a tabela de alocação. Nesta tabela estão os dados dos arquivos, como posição base (aonde o arquivo começa no disco) e tamanho. A tabela permite a inclusão de um número máximo fixo de arquivos (1000).

5.2. Detalhes da versão intermediária

A versão intermediária do gerenciador de arquivos baseou-se no sistema de arquivos *ext2fs* [Card et al. 1994], baseado em *inodes*, implementado apenas parcialmente: é possível criar arquivos na raiz do sistema de arquivos e abrir arquivos criados em discos formatados em *ext2fs* por outros sistemas operacionais, como o Linux. Porém, as manipulações estão limitadas à raiz do sistema de arquivos, sem que o *BananaKernel* tenha uma compreensão de sub-diretórios ou *inodes* que façam referências a outros *inodes*.

O número de índices por *inode* foi limitado a 12 (desconsidera-se os últimos três índices, reservados na definição do sistema para apontar para outros *inodes*), sendo que cada um deles aponta para um bloco de 1024 *bytes* de dados. Então, cada arquivo pode ter, no máximo, $1024 * 12 = 12288$ *bytes*.

5.3. Testes efetuados

O objetivo destes testes foi verificar a capacidade do gerenciador de arquivos em manter e alterar uma tabela de alocação de arquivos e permitir que dados sejam lidos e escritos em arquivos armazenados no sistema de arquivos do disco rígido.

O teste conseguiu abrir dois novos arquivos, salvar dados diferentes dentro deles, fechá-los e reabrí-los em ordem inversa (para testar a tabela de alocação) e mostrou corretamente os seus dados na tela.

6. Escalonador de Processos

Para simplificar a visualização do escalonador de processos, definiu-se como processo qualquer função cujas variáveis sejam estáticas e que tenha sido adicionada à lista de processos. Assim, evita-se códigos complexos para carregar um binário nos formatos comuns de saída dos compiladores, como *ELF* e *a.out*.

O módulo do escalonador de processos fornece funções para adicionar e remover processos, além de uma função para troca de contexto.

`bnnk_addproc()` – inclui um novo processo à lista de processos;
`bnnk_delproc()` – remove um processo desta lista;
`bnnk_swapproc()` – muda o contexto para o próximo processo; e
`bnnk_procstat()` – imprime na tela uma tabela contendo o identificador do processo, o seu endereço na memória, sua prioridade (na versão intermediária), entre outras informações.

Na versão simples, os processos são escolhidos em ordem *FIFO* (*First In - First Out*), sem prioridades ou envelhecimento. A versão intermediária conta com prioridades (estática de dinâmica) e um mecanismo de envelhecimento. A definição do próximo processo a entrar em execução é feita escolhendo o processo que tiver a menor a prioridade dinâmica. Cada vez que um processo entra em execução, sua prioridade dinâamica é reiniciada (volta a ser igual à sua prioridade estática); enquanto isso, os demais processos "envelhecem", decrementando duas respectivas prioridades diâmicas.

A lista de processos é uma lista encadeada, em que cada nodo possui o endereço base do processo, um *flag* indicando se ele já entrou em execução, o seu descritor e o seu contexto salvo e, no caso da versão intermediária, sua prioridade estática (*nice level*) e sua idade.

Quando a troca de contexto é acionada, o próximo processo da lista é selecionado. Então, é verificado se já houve uma execução deste processo que foi interrompida. Caso o processo nunca tenha entrado em execução, é efetuada uma operação *call* para o seu endereço base:

```
__asm__ __volatile__("call *%0" :: "m"(endereco));
```

Se este for um processo que já entrou em execução, então é chamado a operação *longjmp*[1] para o seu contexto.

As versões simples e intermediária são não-preemptivas (ou colaborativas), ou seja, um processo deve pedir para sair, salvando o seu contexto e depois chamando o escalonador. Esse procedimento de requisição para a troca de processo está bem detalhado no código-fonte do caso de teste do escalonador de processos, que acompanha a distribuição.

6.1. Testes efetuados

Para testar a capacidade do escalonador de processos de adicionar processos e intercalar a execução entre eles, foram criados dois processos que imprimem mensagens na tela e, a cada execução, invocam o escalonador.

Os processos entraram em execução na ordem correta, ou seja, *FIFO* para o escalonador simples e respeitando a relação entre a prioridade e o envelhecimento na intermediária, e foram corretamente removidos da fila de processos ao término de sua execução.

7. Testes em Sala de Aula

Os testes em sala de aula têm como objetivo validar a utilidade de uma ferramenta de auxílio ao ensino de Sistemas Operacionais, verificar a facilidade de uso do sistema e a capacidade do Manual do Usuário de expressar correta e claramente os mecanismos de alteração, construção e execução do núcleo do sistema.

Para a execução destes testes, foram escolhidos pelo professor orientador (Carlos Maziero) 7 alunos de uma turma da disciplina Sistemas Operacionais do 4^o período do

[1] `void longjmp(jmp_buf env, int val)` permite retornar a um contexto previamente salvo através da função `int setjmp(jmp_buf env)`

curso de graduação em Ciência da Computação da PUCPR, que se mostraram dispostos a aprender sobre o *BananaKernel*. Estes alunos possuem um bom domínio da linguagem C, e já haviam cursado 80% da disciplina.

Os testes consistem em contruir e executar o núcleo exemplo que acompanha o sistema e fazer pequenas modificações em módulos prontos, como mostra a tabela 1.

Tabela 1. Tarefas a Serem Executadas pelos Alunos

#	identificação	descrição	pontos
1	**modbr**	O aluno deve construir e criar a imagem do núcleo exemplo, movê-la para uma imagem de disco vazia e executá-la dentro do emulador Bochs	40
2	**modmm**	O aluno deve alterar a função `bnnk_free()` da versão simples, para devolver a memória do bloco liberado ao conjunto de memória disponível, quando este bloco for o último alocado	30
3	**modfs**	o aluno deve, utilizando o código da função `bnnk_open()` da versão simples, criar uma função **ls()** que deverá listar todos os arquivos da tabela de alocação	20
4	**modsc**	o aluno deve alterar a função `bnnk_swaproc()` da versão intermediária para envelhecer mais rapidamente os processos na fila de espera	10

7.1. Resultado dos testes em sala de aula

O teste foi dividido em duas partes, de 100 minutos cada uma. Na primeira parte, os alunos foram apresentados ao *BananaKernel* e lhes foi pedido para fazer o *download* do projeto, construir uma imagem do *kernel* e executá-la em uma imagem de disco dentro do emulador.

Entretanto, a configuração do emulador *Bochs* se mostrou excessivamente complicada para os alunos executarem durante a aula, e praticamente toda a primeira parte foi gasta apenas nesta configuração, com pouco sucesso.

Levando esse problema em consideração, foi incluído com o sistema um arquivo de configuração do *Bochs*, em que apenas as primeiras linhas precisariam ser editadas pelos alunos, porque representavam os caminhos para os arquivos de *ROM* do emulador, dependentes de cada instalação.

Essa mudança mostrou-se muito eficiente, e, na segunda parte do teste, gastou-se menos de 10 minutos para ter o *kernel* exemplo em execução.

Outro problema encontrado foi a necessidade de permitir um acesso privilegiado (como *root* ou usando *sudo*) ao alunos, para que eles pudessem montar no *Linux* as imagens de disco para copiar o *kernel* para dentro delas.

Esse segundo problema foi resolvido utilizando uma imagem de *CD-ROM*, ao invés de uma imagem de disco. Uma imagem deste tipo pode ser criada diretamente pelo aluno, sem a necessidade de privilégios que poderiam inviabilizar a utilização do *BananaKernel* em alguns laboratórios e, possivelmente, causar falhas de segurança no *Linux* utilizado.

Tendo sido resolvidos estes dois problemas, foi solcitado aos alunos que fizessem os testes que exigem alguma modificação no código-fonte. O primeiro teste foi sobre o gerenciador de memória (**modmm**) e levou cerca de 20 minutos para ser resolvido pelos alunos; um desses alunos foi além do esperado, modificando o código de teste de memória para melhorar a visualização da alteração que ele tinha implementado.

O segundo teste efetuado contemplou o escalonador de processos (**modsc**), tomando cerca de 10 minutos para ser completado. Novamente, todos os alunos conseguiram fazer as modificações necessárias e executar o *kernel*.

O teste do gerenciador de arquivos não foi solicitado aos alunos, pois exigiria, pelo menos, mais uma etapa de 100 minutos, que não nos estava disponível. Porém, como os alunos conseguiram terminar os outros 3 testes, consideramos os resultados como suficientes para considerar o sistema como uma ferramente da fácil utilização.

7.1.1. Comentários dos alunos

Após os testes com o sistema, foram distribuídas folhas para os alunos expressarem anonimamente as suas opiniões. Todos os alunos acharam muito difícil realizar a configuração do ambiente *Bochs* como tinha sido proposto inicialmente, mas que, com o arquivo de configuração, o uso se tornou muito mais fácil.

Os alunos também acharam fácil fazer as modificações, sendo que um dos alunos sugeriu solicitarmos modificações mais difíceis. Mais da metade dos alunos indicou ter sentido uma grande satisfação ao ver uma modificação feita por eles executando dentro de um sistema operacional.

Dois pontos importantes destacados pelos alunos foram a simplicidade do código que, segundo eles, permitiu um claro entendimento do funcionamento dos conceitos apresentados, e a qualidade dos comentários, explicando o funcionamento interno do sistema e facilitando as modificações.

8. Trabalhos Futuros

Este trabalho contempla as versões simples e intermediária do sistema e, nesta seção serão discutidas sugestões para a versão avançada, levando em consideração cada módulo do sistema.

8.1. Gerenciador de memória

As versões simples e intermediária não possuem controle de acesso à memória, e o acesso é feito com endereços absolutos. É interessante criar, na versão avançada, um mecanismo de proteção para os espaços de memória de cada processo, e tornar os endereços relativos ao endereço base de cada processo.

8.2. Gerenciador de arquivos

A versão intermediária do gerenciador de arquivos considera apenas arquivos que se encontram na raiz do sistema de arquivos, e não compreende *inodes* que apontam para outros *inodes*[2]. Assim, não é possivel a utilização de diretórios, nem de arquivos maiores que 12 blocos.

Uma alteração interessante para a versão avançada seria a implementação de um código que trate essas questões, permitindo a utilização de diretórios e de arquivos com qualquer tamanho, limitados apenas pela capacidade do disco.

8.3. Escalonador de processos

Nas versões atuais, o usuário deve criar uma variável global que contém os dados do processo. Além disso, não há preempção, e cada processo deve salvar o seu contexto e chamar a função `bnnk_swapproc()` para liberar o processador e permitir que outros processos entrem em execução.

Para a versão avançada, sugere-se manter todos os dados do processo dentro do espaco de memória deste processo, evitando o uso da variável global. Sugere-se, também, a utilização da interrupção de *timer* para salvar o contexto do processo atual e fazer a chamada à função `bnnk_swapproc()`, criando, assim, a preempção.

9. Trabalhos Correlatos

Existem alguns projetos que são freqüentemente utilizados para o ensino de sistemas operacionais, como o MINIX [Tannenbaum 2006] e o Nachos [Anderson 1996]. Apesar destes possuírem implementações mais completas, aproximando-se mais de SOs de uso geral, eles não possuem documentação significativa em português. Além disso, a quantidade e o detalhamento dos comentários em seus códigos-fonte parecem aquém do necessário para nossa realidade pedagógica.

10. Conclusão

Ficou claro para nós, após os testes em sala de aula, que o *BananaKernel* cumpre os seus objetivos como um sistema operacional para o ensino. Os alunos puderam, sem a necessidade de muitas explicações, implementar modificações no sistema e visualizar o seu funcionamento.

Acreditamos que um professor da disciplina de Sistemas Operacionais poderá utilizar este sistema como uma ferramenta de ensino, pedindo aos alunos modificações e novas funcionalidades, sem ter que se preocupar com as limitações que um sistema operacional de produção imporia.

Todavia, a facilidade na utilização só foi alcançada retirando-se do sistema partes muito complexas, mas interessantes, presentes em sistemas operacionais reais. Por exemplo, não existem mecanismos para carregar executáveis nos formatos binários mais populares (como *ELF* e *a.out*) nem um código-fonte, facilmente compreensível, para a etapa de *boot*.

[2] Os índices 13, 14 e 15 de cada *inode* são reservados para apontarem para outros *inodes*

Todo o código-fonte, os documentos necessários para a utilização e demais informações (como listas de discussão), estão disponíveis no endereço http://bananakernel.sf.net/, sob a licença *Mozilla Public License Version 1.1*[3].

Referências

Anderson, T. (1996). *README for the Nachos Release*. UC Berkeley. http://www.cs.washington.edu/homes/tom/nachos/README.

ANSI (1989). *American National Standard Programming Language C, ANSI X3.159-1989*. American National Standards Institute, 1430 Broadway, New York, NY 10018, USA.

Card, R., Ts'o, T., and Tweedie, S. (1994). Design and implementation of the second extended filesystem. In *Proceedings of the First Dutch International Symposium on Linux*. State University of Gronigen.

Ford, B., Back, G., Benson, G., Lepreau, J., Lin, A., and Shivers, O. (1997). The Flux OSKit: A substrate for OS and language research. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Saint-Malo, France.

Lawton, K., Denney, B., Guarneri, N. D., Ruppert, V., and Bothamy, C. (2004). *Bochs User Manual*. http://bochs.sourceforge.net/doc/docbook/user/book1.html.

Maziero, C. A. (2002). Reflexões sobre o ensino prático de sistemas operacionais. In *Anais do X Workshop Sobre Educação em Computação da SBC*, pages 1–12. SBC - Sociedade Brasileira de Computação.

Tannenbaum, A. (2006). The Minix operating system. http://www.minix3.org.

Teigão, R. C. and Morimoto, J. H. (2004a). *BananaKernel*: Um sistema operacional didático – manual do usuário. Technical report, Pontifícia Universidade Católica do Paraná.

Teigão, R. C. and Morimoto, J. H. (2004b). *BananaKernel*: Um sistema operacional didático – plano de teste. Technical report, Pontifícia Universidade Católica do Paraná.

Teigão, R. C. and Morimoto, J. H. (2004c). *BananaKernel*: Um sistema operacional didático – projeto lógico. Technical report, Pontifícia Universidade Católica do Paraná.

[3]Disponí vel em http://www.mozilla.org/MPL/MPL-1.1.html