Rafael Coninck Teigão
Julio Henrique Morimoto

Projeto Lógico
BananaKernel*: Um Sistema Operacional Didático*

Curitiba - PR

Junho 2004

Rafael Coninck Teigão
Julio Henrique Morimoto

Projeto Lógico
BananaKernel*: Um Sistema Operacional Didático*

Trabalho apresentado pelos alunos do 7º período do curso de Bacharelado em Ciência da Computação para a disciplina Projeto Final I.

Orientador:
Prof. Dr. Carlos Alberto Maziero

Pontifícia Universidade Católica do Paraná

Curitiba - PR

Junho 2004

Sumário

Lista de Figuras p. iv

Lista de Tabelas p. v

1 Introdução p. 1

1.1 Objetivo . . . . . p. 1

1.2 Escopo . . . . . p. 2

2 Modelagem do Projeto p. 4

2.1 Justificativa da Escolha da Abordagem . . . . . p. 4

2.2 Declaração de Objetivos e Escopo . . . . . p. 4

2.3 Levantamento de Requisitos . . . . . p. 4

2.4 Modelo Ambiental . . . . . p. 5

2.4.1 Diagrama de Contexto . . . . . p. 6

2.4.2 Lista de Eventos . . . . . p. 6

2.5 Modelo Comportamental . . . . . p. 8

2.5.1 DFD Individual de Resposta aos Eventos . . . . . p. 8

2.5.1.1 Ação #1: Alocar memória OU Configurar *errno* . . . . p. 8

2.5.1.2 Ação #2: Re-alocar memória OU Configurar *errno* . . p. 9

2.5.1.3 Ação #3: Liberar memória alocada OU Configurar *errno* p. 9

2.5.1.4 Ação #4: Informar memória disponível . . . . . p. 10

2.5.1.5 Ação #5: Informar tamanho de bloco de memória alocada OU Configurar *errno* . . . . . p. 11

2.5.1.6 Ação #6: Abrir arquivo OU Configurar *errno* . . . . . p. 11

2.5.1.7 Ação #7: Ler arquivo aberto OU Configurar *errno* . . p. 12

2.5.1.8 Ação #8: Escrever em arquivo aberto OU Configurar *errno* . . . . . p. 13

2.5.1.9 Ação #9: Fechar arquivo OU Configurar *errno* . . . . p. 13

2.5.1.10 Ação #10: Incluir novo processo OU Configurar *errno* p. 14

2.5.1.11 Ação #11: Remover processo OU Configurar *errno* . . p. 15

2.5.1.12 Ação #12: Trocar processo . . . . . p. 15

2.5.1.13 Ação #13: Imprimir mensagem de erro . . . . . p. 16

2.5.2 DFD Detalhado de Resposta aos Eventos . . . . . p. 16

2.5.3 Dicionário de Dados . . . . . p. 16

2.5.4 Modelo Lógico de Dados . . . . . p. 16

2.5.5 Diagrama de Transição de Estados . . . . . p. 17

2.5.6 Especificação dos Processos Primitivos . . . . . p. 19

2.5.6.1 Ação #1: Alocar memória OU Configurar *errno* . . . . p. 20

2.5.6.2 Ação #2: Re-alocar memória OU Configurar *errno* . . p. 22

2.5.6.3 Ação #3: Liberar memória alocada OU Configurar *errno* p. 24

2.5.6.4 Ação #4: Informar memória disponível . . . . . p. 26

2.5.6.5 Ação #5: Informar tamanho de bloco de memória alocada OU Configurar *errno* . . . . . p. 27

2.5.6.6 Ação #6: Abrir arquivo OU Configurar *errno* . . . . . p. 28

2.5.6.7 Ação #7: Ler arquivo aberto OU Configurar *errno* . . p. 30

2.5.6.8 Ação #8: Escrever em arquivo aberto OU Configurar *errno* . . . . . p. 31

2.5.6.9 Ação #9: Fechar arquivo OU Configurar *errno* . . . . p. 32

2.5.6.10 Ação #10: Incluir novo processo OU Configurar *errno* p. 34

2.5.6.11 Ação #11: Remover processo OU Configurar *errno* . . p. 35

2.5.6.12 Ação #12: Trocar processo . . . . . p. 36

2.5.6.13 Ação #13: Imprimir mensagem de erro . . . . . p. 38

2.6 Protótipo . . . . . p. 39

3 Riscos p. 40

4 Cronograma p. 43

5 Documentos Adicionais p. 44

5.1 CD-ROM . . . . . p. 44

5.2 Manual do Usuário . . . . . p. 45

5.3 Plano de Testes . . . . . p. 46

6 Conclusão p. 47

7 Responsabilidades p. 48

Anexo A – Descrição Detalhada do Escopo p. 49

A.1 Escopo da Gerência de Memória . . . . . p. 49

A.2 Escopo da Gerência de Arquivos . . . . . p. 49

A.3 Escopo do Escalonador de Processos . . . . . p. 50

Referências Bibliográficas p. 51

Índice Remissivo p. 52

Lista de Figuras

1 Definição do Escopo do Projeto . . . . . p. 2

2 Diagrama de Contexto . . . . . p. 6

3 Diagrama **`bnnk_malloc()`** . . . . . p. 8

4 Diagrama **`bnnk_realloc()`** . . . . . p. 9

5 Diagrama **`bnnk_free()`** . . . . . p. 10

6 Diagrama **`bnnk_avail()`** . . . . . p. 10

7 Diagrama **`bnnk_getsize()`** . . . . . p. 11

8 Diagrama **`bnnk_open()`** . . . . . p. 12

9 Diagrama **`bnnk_read()`** . . . . . p. 12

10 Diagrama **`bnnk_write()`** . . . . . p. 13

11 Diagrama **`bnnk_close()`** . . . . . p. 14

12 Diagrama **`bnnk_addproc()`** . . . . . p. 14

13 Diagrama **`bnnk_delproc()`** . . . . . p. 15

14 Diagrama **`bnnk_swapproc()`** . . . . . p. 15

15 Diagrama **`bnnk_perror()`** . . . . . p. 16

16 Diagrama Entidade-Relacionamento . . . . . p. 17

17 Diagrama de Transição de Estados . . . . . p. 18

18 Cronograma . . . . . p. 43

Lista de Tabelas

1 Lista de Eventos . . . . . p. 7

2 Tabela de Transição de Estados . . . . . p. 18

3 Nomenclatura dos valores de Risco . . . . . p. 40

4 Quantificação dos Riscos . . . . . p. 41

5 Riscos X Percentagens . . . . . p. 42

1 Introdução

A criação de um sistema operacional, além de ser um tópico de pesquisa muito interessante, pois envolve muitos conceitos desenvolvidos em cursos de Ciência e Engenharia de Computação, é também uma oportunidade para criar uma ferramenta de ensino para a disciplina *Sistemas Operacionais*.

Como será visto a seguir, na seção 1.1, este projeto tem como intenção criar um sistema operacional que irá explicar o que está sendo feito internamente, para que o aluno tenha uma maior facilidade de visualização dos processos internos deste tipo de sistema, além de possuir várias funções (vide seção 1.2) para facilitar a modificação e inclusão de novas partes criadas pelo aluno.

Este documento encontra-se dividido em 8 partes, sendo elas:

1. Esta introdução, contendo, também, os objetivos e o escopo do projeto;
2. a modelagem do projeto;
3. uma descrição dos riscos e contingências;
4. o cronograma;
5. a conclusão;
6. a descrição dos documentos adicionais;
7. a data de conclusão e a assinatura dos responsáveis; e
8. anexos.

1.1 Objetivo

O propósito do sistema é proporcionar uma ferramenta para auxiliar no ensino da disciplina *Sistemas Operacionais*, demonstrando alguns dos mecanismos internos de fun-

cionamento de um SO [1] moderno.

O sistema deve apresentar mensagens explicando partes do funcionamento, deve possuir uma documentação e um desenvolvimento que possibilite aos alunos e professores a criação de novos módulos para substituir ou acrescentar funcionalidades.

1.2 Escopo

Existe uma necessidade no meio acadêmico de um sistema que facilite o ensino de conceitos de SO para alunos, principalmente dos cursos de Bacharelado em Ciência da Computação e Engenharia de Computação.

Este sistema deve possuir uma documentação que explique o funcionamento de conceitos importantes, como gerência de arquivos e memória, por exemplo, e ainda permita que o aluno ou o professor faça substituições em módulos responsáveis pela apresentação desses conceitos.

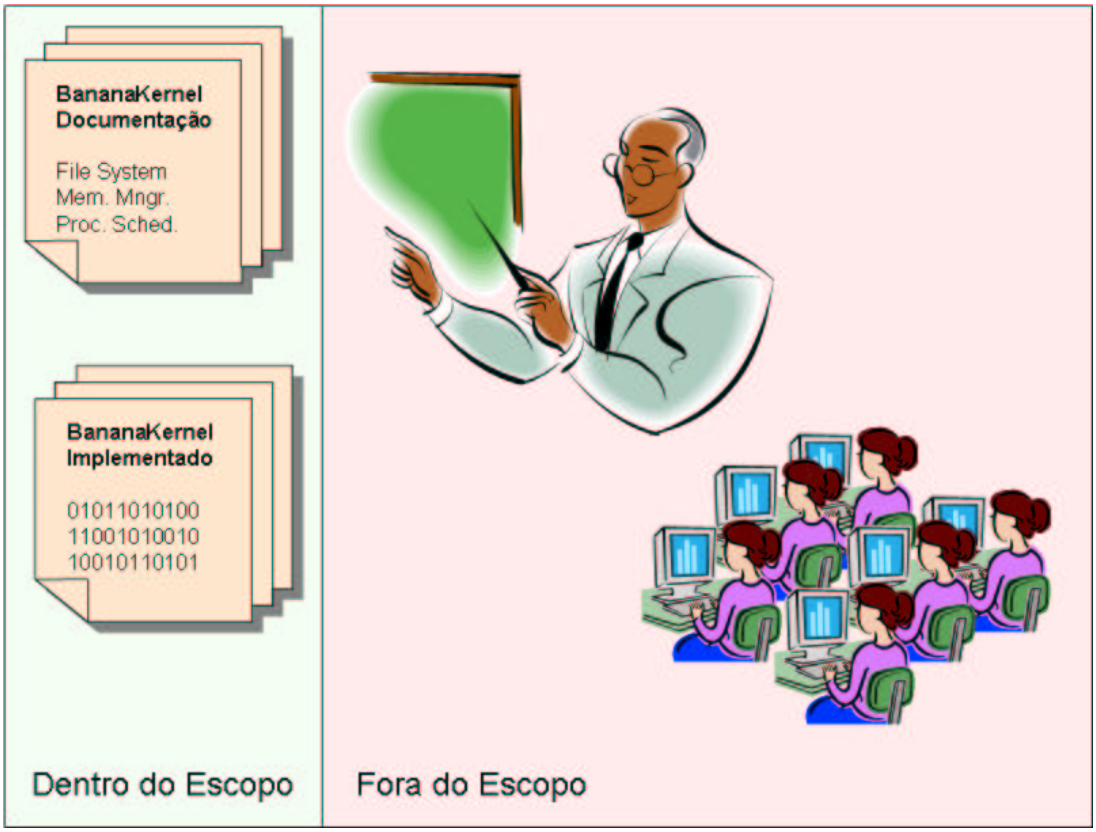

Figura 1: Definição do Escopo do Projeto

O escopo deste projeto, ilustrado na figura 1, cobre a criação de um sistema ope-

[1] Sistema Operacional

racional simples, contendo módulos de gerência de memória (vide Anexo A.1), gerência de arquivos (vide Anexo A.2) e escalonador de processos (vide Anexo A.3), sendo que esses módulos devem apresentar mensagens na tela do computador explicando o funcionamento interno de cada subsistema (na figura, ilustrado com o título "BananaKernel Implementação").

O projeto cobre, também, a criação de um manual de usuário que explique como incluir ou substituir módulos, além de uma breve introdução à tecnologia e ao método de desenvolvimento utilizado na criação do sistema, e uma função **`bnnk_perror()`** como descrita em (American National Standards Institute, 1989), responsável pela apresentação das mensagens de erro (na figura, ilustrado com o título "BananaKernel Documentação").

Este projeto não cobre, porém, ensinar aos usuários linguagens de programação, nem pretende montar um curso de sistemas operacionais, como mostra a figura 1 na página precedente. O fornecimento do equipamento necessário para implantação, as demais ferramentas para a execução do sistema ou para condução das aulas e os *software* e sistemas necessários para o desenvolvimento ou modificação de módulos também não estão cobertos (lado direito da figura, mostrando o professor, os alunos e os equipamentos utilizados).

Assume-se, como premissas, que os usuários têm conhecimento e capacidade de criar programas nas linguagens de programação adequadas à alternativa escolhida, bem como capacidade de compreender textos escritos em inglês técnico, além de estarem sendo acompanhados por um professor capacitado da disciplina *Sistemas Operacionais*.

2 *Modelagem do Projeto*

Este projeto está sendo modelado utilizando a abordagem da **Análise Essencial** e, neste capítulo, serão expostos os artefatos criados durante esta análise. Nas tabelas e diagramas a seguir, o uso de "/" indica "OU" e "+" indica "E" .

2.1 Justificativa da Escolha da Abordagem

A programação orientada a objetos envolve incertezas oriundas dos mecanismos de resolução dinâmica de sobrecargas e polimorfismos. Tais incertezas não são desejáveis no âmbito de um núcleo de sistema operacional, que deve possuir um comportamento tão determinístico e previsível em tempo de compilação quanto possível.

Desta forma, entende-se mais eficiente modelar o projeto utilizando análise essencial, mantendo o mesmo próximo da realidade de implementação, sem que sejam necessárias traduções em representações mais abstratas, como modelar estruturas em linguagem C via objetos em modelagem orientada a objetos.

2.2 Declaração de Objetivos e Escopo

A Declaração de Objetivos e o Escopo encontram-se, respectivamente, nas seções 1.1 e 1.2. Entende-se que a repetição dos mesmos aqui seria redundante.

2.3 Levantamento de Requisitos

O levantamento dos requisitos do sistema foi conduzido através de diversas reuniões, presenciais ou por meio eletrônico, entre os dois alunos responsáveis pelo projeto e seu orientador.

Nas primeiras reuniões, o orientador informou quais características seriam desejáveis

em um sistema operacional didático, sem definir explicitamente funções específicas.

Posteriormente, foi proposto pela equipe o desenvolvimento de uma única versão de três módulos básicos: *gerenciador de memória*, *gerenciador de arquivos* e *escalonador de processos*. Todavia, o desenvolvimento de apenas uma versão de cada módulo mostrou-se insuficiente para cumprir os objetivos didáticos, sendo, então, requisitada pelo orientador a criação de duas versões para cada módulo:

1. versão simples, em que a utilização da memória e do sistema de arquivos deve ser contínua, e em que o escalonador não possui prioridades ou envelhecimento; e
2. versão intermediária, utilizando algum tipo de particionamento em blocos da memória, acesso ao disco através de um sistema de arquivos mais elaborado, e escalonador com prioridades e envelhecimento.

Ficou clara também a necessidade da criação de um manual do usuário que permita uma fácil assimilação do sistema por alunos participando na disciplina *Sistemas Operacionais*.

Em trocas de *e-mail*, utilizando uma lista de discussão[1] criada especificamente para este projeto, foram descritas mais detalhadamente as funções básicas do sistema. Esta lista é utilizada também para, juntamente com um servidor de CVS[2], permitir um acompanhamento mais detalhado do projeto pelo orientador.

Tanto a lista de discussão, quanto o servidor de CVS, são mantidos por uma entidade externa, conhecida como *SourceForge*[3]. Esta entidade é responsável pelo armazenamento e *backup* destes dados; em caso de falha neste serviço, existem cópias de segurança armazenadas localmente.

2.4 Modelo Ambiental

Os fluxos de dados entre as entidades externas e o sistema estarão a seguir representados no **Diagrama de Contexto** e na **Lista de Eventos**.

[1] bananakernel-developers@lists.sourceforge.net
[2] *Concurrent Versions System* - Sistema de Versões Concorrentes
[3] `http://sourceforge.net/`

2.4.1 Diagrama de Contexto

A figura 2 mostra o Diagrama de Contexto para o sistema. Neste diagrama, o termo "cliente" é utilizado como referência a um programa, ou função, sendo executado sobre o *BananaKernel*[4].

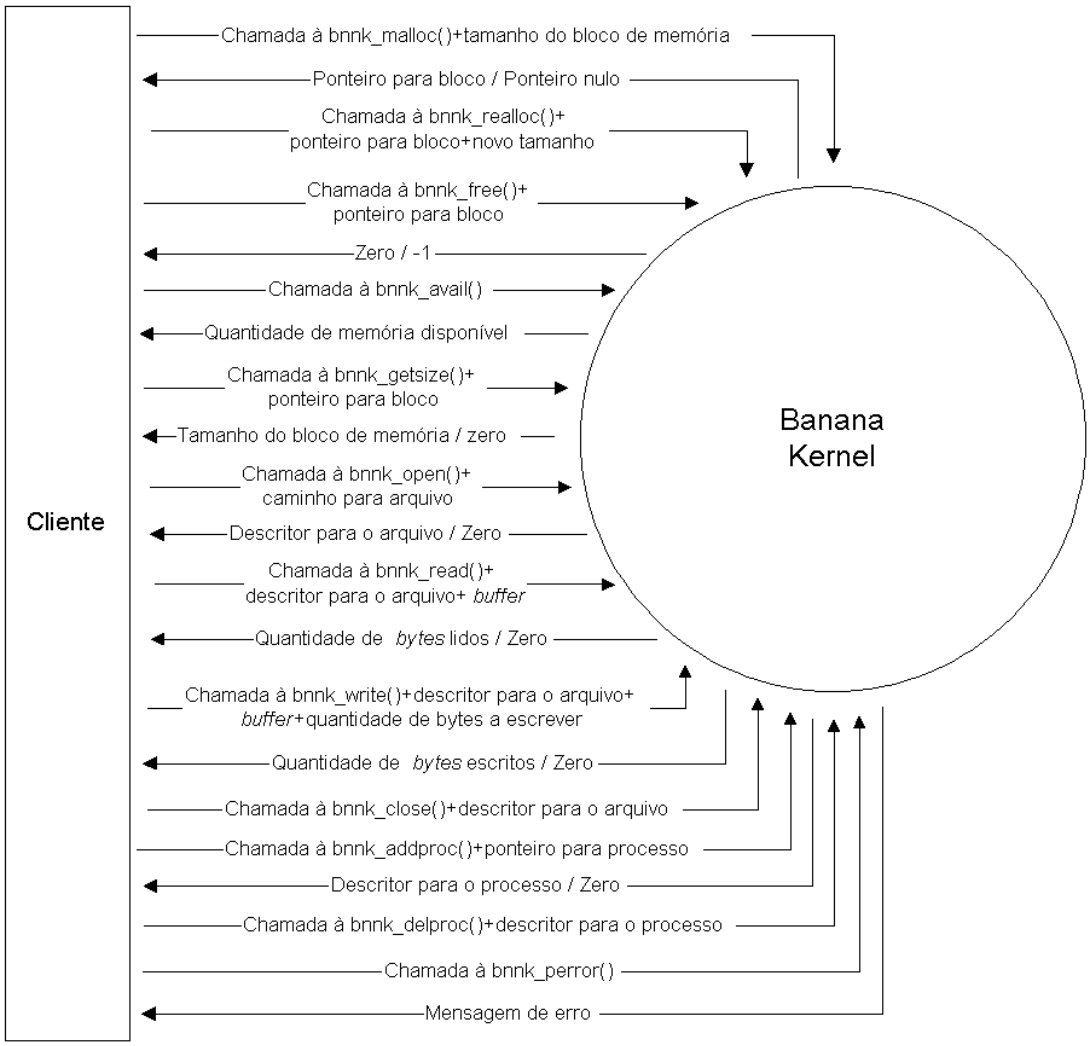

Figura 2: Diagrama de Contexto

2.4.2 Lista de Eventos

A tabela 1 na próxima página mostra a Lista de Eventos para o sistema. Novamente, o termo "cliente" é utilizado para representar um programa ou função sendo executado sobre o *BananaKernel*.

[4]É comum chamar programas de "clientes do sistema operacional".

Tabela 1: Lista de Eventos

#	Nome	Estímulo	Ação	Resposta	Tipo
1	Cliente solicita alocação de memória	Chamada a `bnnk_malloc()` + tamanho do bloco de memória	Alocar memória / Configurar *errno*	Ponteiro para bloco / Ponteiro nulo	F
2	Cliente solicita realocação de memória	Chamada a `bnnk_realloc()` + ponteiro para bloco + novo tamanho	Re-alocar memória / Configurar *errno*	Ponteiro para bloco / Ponteiro nulo	F
3	Cliente solicita liberação de memória alocada	Chamada a `bnnk_free()` + ponteiro para bloco	Liberar memória alocada / Configurar *errno*	Zero / -1	F
4	Cliente pede informação sobre memória disponível	Chamada a `bnnk_avail()`	Informar memória disponível	Quantidade de memória disponível	F
5	Cliente pede informação sobre tamanho de bloco de memória alocada	Chamada a `bnnk_getsize()` + ponteiro para bloco	Informar tamanho de bloco de memória alocada / Configurar *errno*	Tamanho do bloco de memória / Zero	F
6	Cliente solicita abertura de arquivo	Chamada a `bnnk_open()` + caminho para arquivo	Abrir arquivo / Configurar *errno*	Descritor para o arquivo / Zero	F
7	Cliente solicita leitura de arquivo	Chamada a `bnnk_read()` + descritor para o arquivo + *buffer*	Ler arquivo aberto / Configurar *errno*	Quantidade de *bytes* lidos / Zero	F
8	Cliente pede escrita de arquivo	Chamada a `bnnk_write()` + descritor para o arquivo + *buffer* + quantidade de *bytes* a escrever	Escrever em arquivo aberto / Configurar *errno*	Quantidade de *bytes* escritos / Zero	F
9	Cliente pede fechamento de arquivo	Chamada a `bnnk_close()` + descritor para o arquivo	Fechar arquivo / Configurar *errno*	Zero / -1	F
10	Cliente solicita inclusão de um novo processo	Chamada a `bnnk_addproc()` + ponteiro para processo	Incluir novo processo / Configurar *errno*	Descritor para o processo / Zero	F
11	Cliente solicita remoção de um processo	Chamada a `bnnk_delproc()` + descritor para o processo	Remover processo / Configurar *errno*	Zero / -1	F
12	Hora de trocar processo em execução	-	Trocar processos	Perda de execução do processo corrente + inicio de execução do próximo processo	T
13	Cliente solicita mensagem de erro	Chamada a `bnnk_perror()`	Imprimir mensagem de erro	Mensagem de erro	F

2.5 Modelo Comportamental

Nesta seção, cada evento será individualmente apresentado, especificando-se o **DFD individual** de resposta, o **DFD detalhado**, caso seja necessário, e o **Modelo Lógico de Dados** para cada depósito de dados definido nos DFDs.

Serão também apresentados o **Diagrama de Transição de Estados** e a **Especificação dos Processos Primitivos**.

O **Dicionário de Dados** poderá ser encontrado no CD-ROM entregue com esta documentação.

2.5.1 DFD Individual de Resposta aos Eventos

Cada ação será abaixo apresentada separadamente.

2.5.1.1 Ação #1: Alocar memória OU Configurar *errno*

A figura 3 mostra o DFD individual para a ação "Alocar memória OU Configurar *errno*".

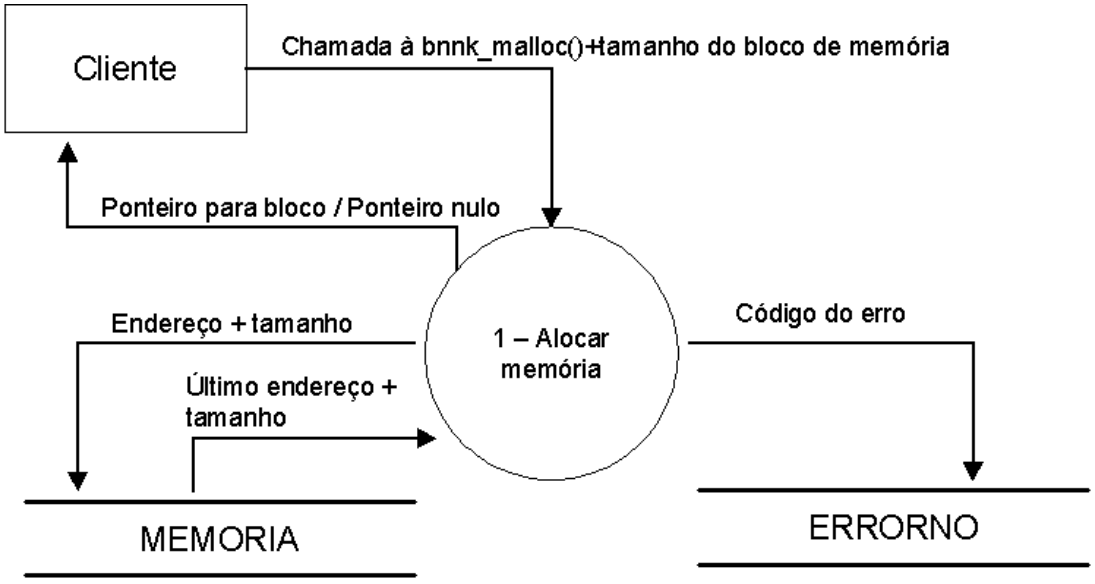

Figura 3: Diagrama **`bnnk_malloc()`**

O cliente faz uma chamada à **`bnnk_malloc()`**, passando o tamanho do bloco de memória desejado. O sistema cadastra este novo bloco no depósito "MEMÓRIA" e retorna o ponteiro para este bloco, ou configura o registro *errno* e retorna 0 (zero).

2.5.1.2 Ação #2: Re-alocar memória OU Configurar *errno*

A figura 4 mostra o DFD individual para a ação "Re-alocar memória OU Configurar *errno*".

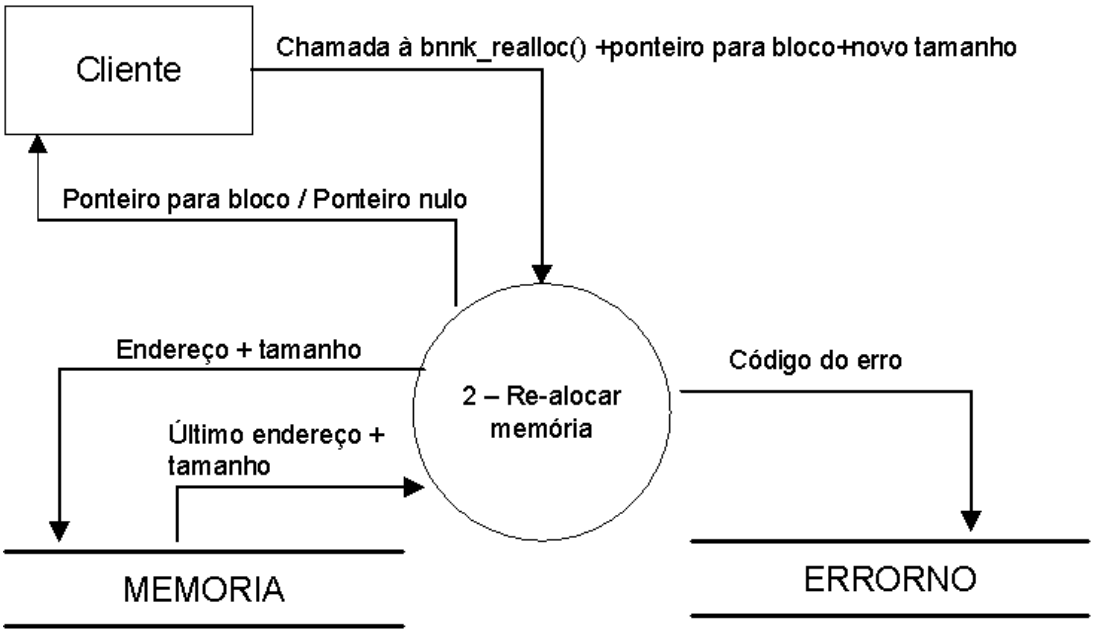

Figura 4: Diagrama **`bnnk_realloc()`**

O cliente faz uma chamada ao **`bnnk_realloc()`**, passando o bloco que se deseja alterar e o novo tamanho deste bloco. O sistema atualiza as informações do bloco no depósito "MEMÓRIA" e retorna a nova posição do bloco, ou configura o registro *errno* e retorna 0 (zero).

2.5.1.3 Ação #3: Liberar memória alocada OU Configurar *errno*

A figura 5 na página seguinte mostra o DFD individual para a ação "Liberar memória alocada OU Configurar *errno*".

O cliente faz uma chamada ao **`bnnk_free()`**, passando o bloco que deseja ser liberado. O sistema faz a alteração no depósito "MEMÓRIA" e retorna 0 (zero), ou configura o registro *errno* e retorna -1.

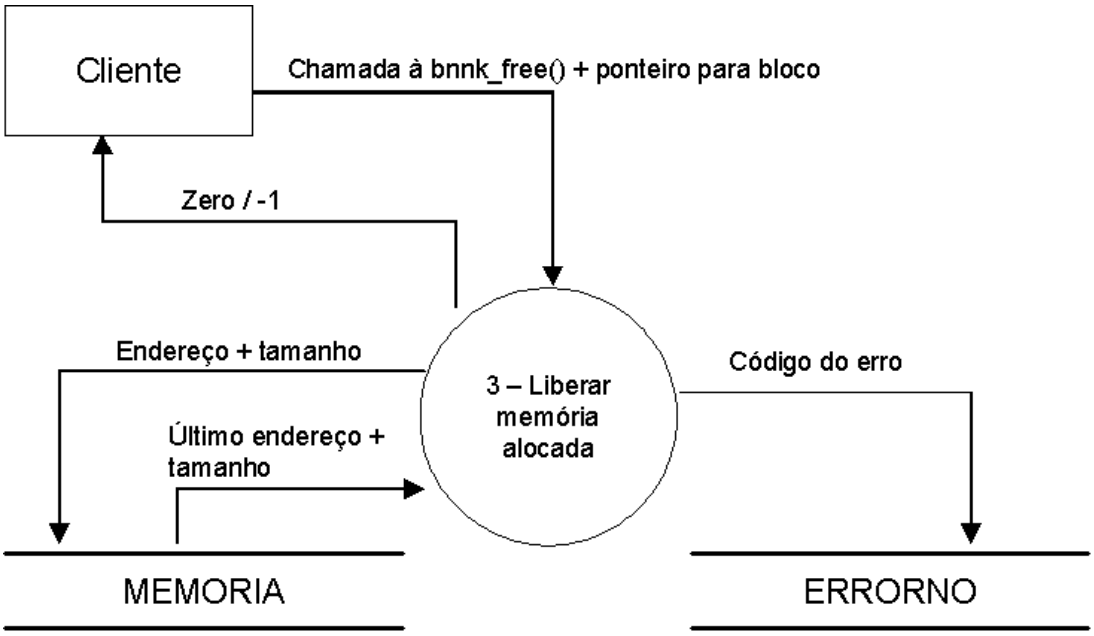

Figura 5: Diagrama **`bnnk_free()`**

2.5.1.4 Ação #4: Informar memória disponível

A figura 6 mostra o DFD individual para a ação "Informar memória disponível".

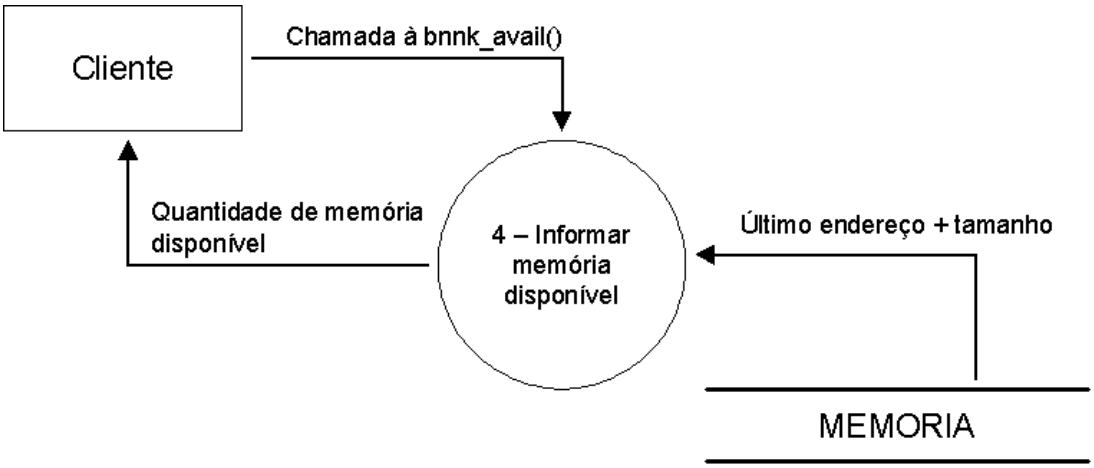

Figura 6: Diagrama **`bnnk_avail()`**

O cliente faz uma chamada ao **`bnnk_avail()`**, e o sistema busca no depósito "MEMÓRIA" as informações necessárias para calcular a memória livre, e retorna a quantidade disponível.

2.5.1.5 Ação #5: Informar tamanho de bloco de memória alocada OU Configurar *errno*

A figura 7 mostra o DFD individual para a ação "Informar tamanho de bloco de memória alocada OU Configurar *errno*".

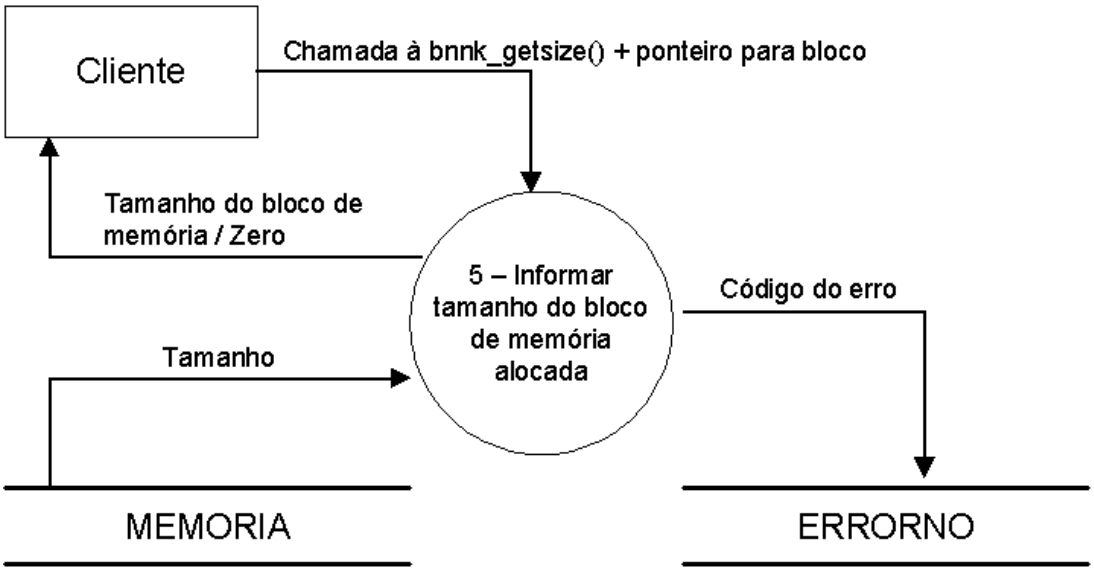

Figura 7: Diagrama **`bnnk_getsize()`**

O cliente faz uma chamada ao **`bnnk_getsize()`**, passando o bloco que se deseja conhecer o tamanho. O sistema busca os dados do bloco no depósito "MEMÓRIA" e retorna o tamanho do bloco, ou configura o registro *errno* e retorna 0 (zero).

2.5.1.6 Ação #6: Abrir arquivo OU Configurar *errno*

A figura 8 na próxima página mostra o DFD individual para a ação "Abrir arquivo OU Configurar *errno*".

O cliente faz uma chamada ao **`bnnk_open()`**, passando o caminho para o arquivo. O sistema busca os dados do arquivo no depósito "TABELA_ARQUIVOS", inclui os dados do arquivo no depósito "ARQUIVOS_ABERTOS" e retorna o descritor para o arquivo, ou configura o registro *errno* e retorna 0 (zero).

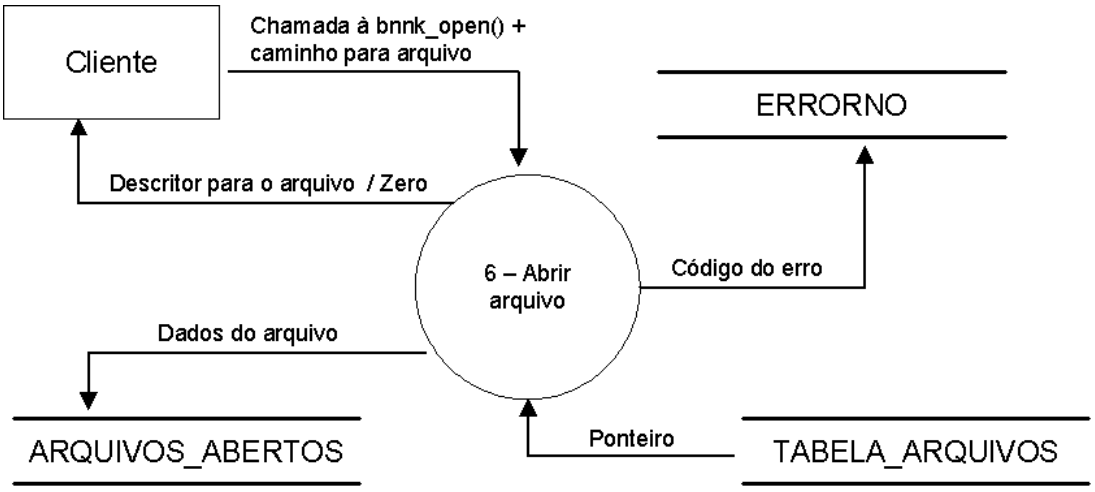

Figura 8: Diagrama **bnnk_open()**

2.5.1.7 Ação #7: Ler arquivo aberto OU Configurar *errno*

A figura 9 mostra o DFD individual para a ação "Ler arquivo aberto OU Configurar *errno*".

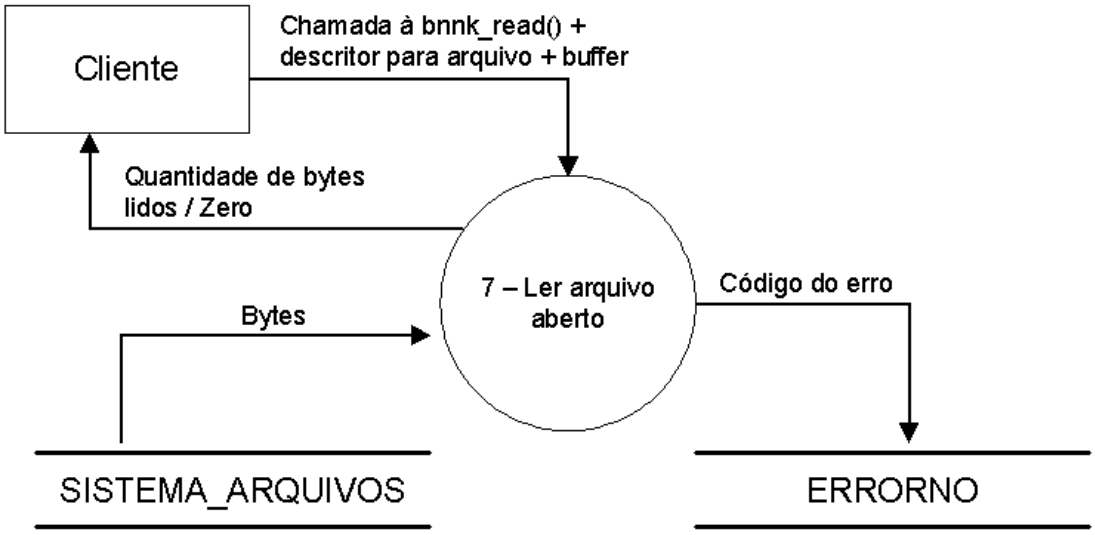

Figura 9: Diagrama **bnnk_read()**

O cliente faz uma chamada ao **bnnk_read()**, passando o descritor para o arquivo e o *buffer*. O sistema busca os dados no depósito "SISTEMA_ARQUIVOS", armazenado-os internamente no *buffer*, e retorna a quantidade de *bytes* lidos, ou configura o registro *errno* e retorna 0 (zero).

2.5.1.8 Ação #8: Escrever em arquivo aberto OU Configurar *errno*

A figura 10 mostra o DFD individual para a ação "Escrever em arquivo aberto OU Configurar *errno*".

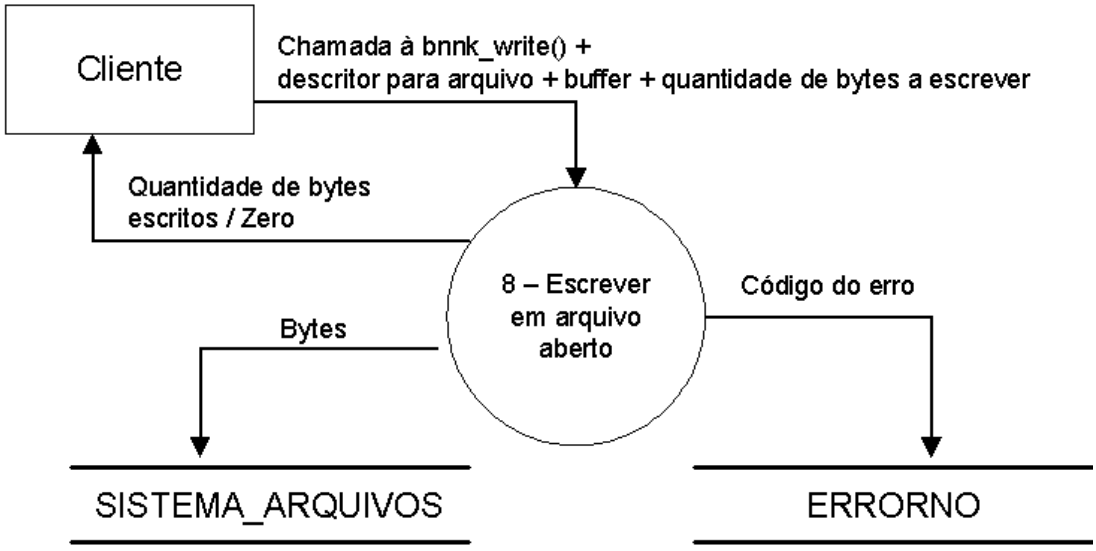

Figura 10: Diagrama **`bnnk_write()`**

O cliente faz uma chamada ao **`bnnk_write()`**, passando o descritor para o arquivo, o *buffer* e a quantidade de bytes a escrever. O sistema escreve os dados do *buffer* no depósito "SISTEMA_ARQUIVOS" e retorna a quantidade de *bytes* escritos, ou configura o registro *errno* e retorna 0 (zero).

2.5.1.9 Ação #9: Fechar arquivo OU Configurar *errno*

A figura 11 na página seguinte mostra o DFD individual para a ação "Fechar arquivo OU Configurar *errno*".

O cliente faz uma chamada ao **`bnnk_close()`**, passando o descritor para o arquivo. O sistema atualiza o depósito "ARQUIVOS_ABERTOS" e retorna 0 (zero), ou configura o registro *errno* e retorna -1.

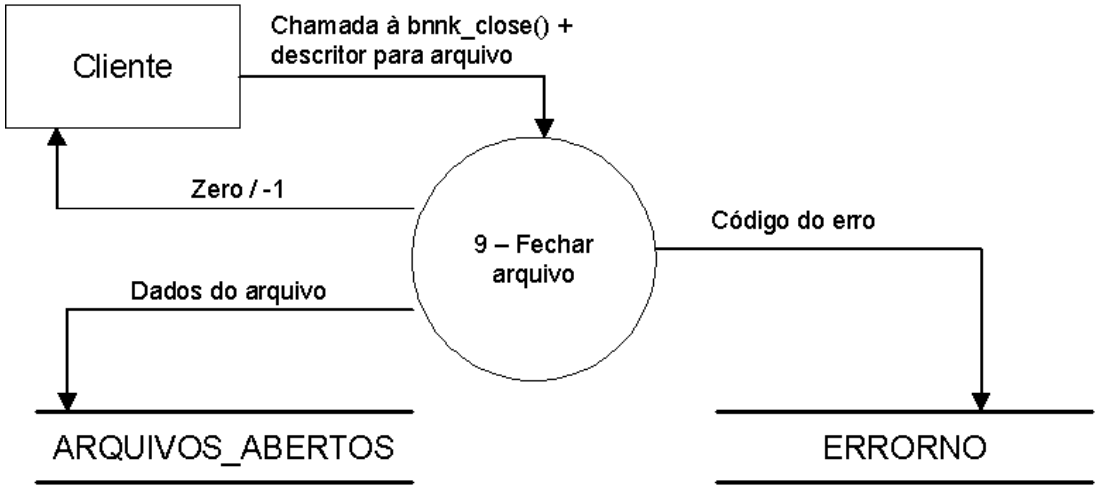

Figura 11: Diagrama **`bnnk_close()`**

2.5.1.10 Ação #10: Incluir novo processo OU Configurar *errno*

A figura 12 mostra o DFD individual para a ação "Incluir novo processo OU Configurar *errno*".

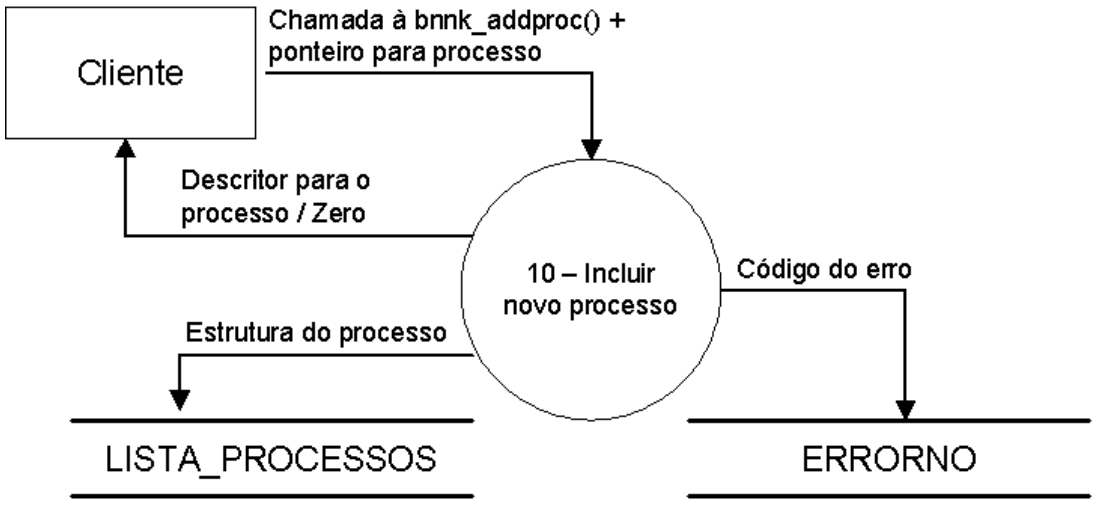

Figura 12: Diagrama **`bnnk_addproc()`**

O cliente faz uma chamada ao **`bnnk_addproc()`**, passando o ponteiro para processo. O sistema passa a estrutura do processo para o depósito "LISTA_PROCESSOS" e retorna o descritor para o processo, ou configura o registro *errno* e retorna 0 (zero).

2.5.1.11 Ação #11: Remover processo OU Configurar *errno*

A figura 13 mostra o DFD individual para a ação "Remover processo OU Configurar *errno*".

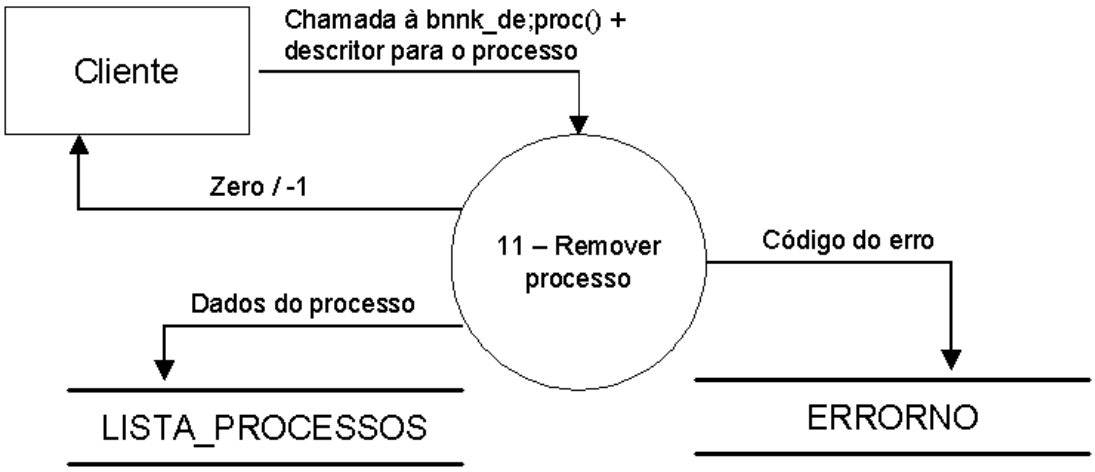

Figura 13: Diagrama **`bnnk_delproc()`**

O cliente faz uma chamada ao **`bnnk_delproc()`**, passando o descritor para o processo. O sistema atualiza o depósito "LISTA_PROCESSOS" e retorna 0 (zero), ou configura o registro *errno* e retorna -1.

2.5.1.12 Ação #12: Trocar processo

A figura 14 mostra o DFD individual para a ação "Trocar processo".

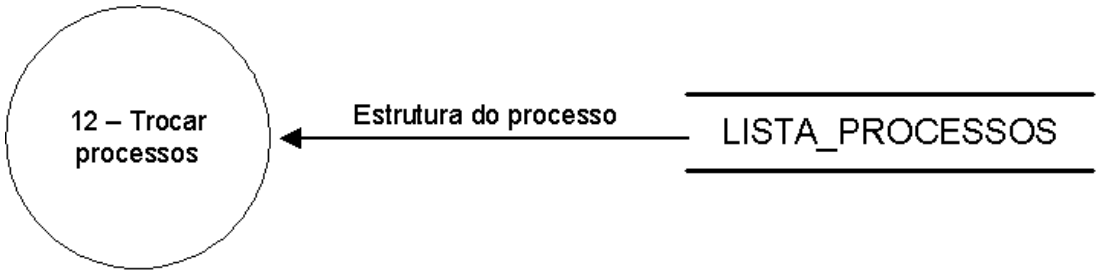

Figura 14: Diagrama **`bnnk_swapproc()`**

Quando chega o momento de trocar o processo em execução, através de uma interrupção de *hardware* o sistema chama **`bnnk_swapproc()`**, que busca no depósito "LISTA_PROCESSOS" os dados do próximo processo a entrar em execução.

2.5.1.13 Ação #13: Imprimir mensagem de erro

A figura 15 mostra o DFD individual para a ação "Imprimir mensagem de erro".

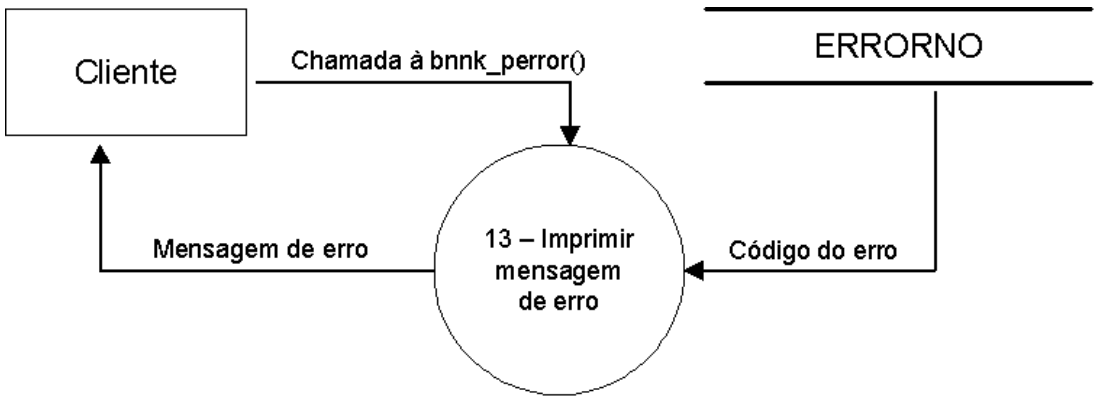

Figura 15: Diagrama **`bnnk_perror()`**

O cliente faz uma chamada ao **`bnnk_perror()`**. O sistema busca no depósito "ERRORNO" qual foi o último código entrado em *errno*, e imprime a mensagem de erro correspondente.

2.5.2 DFD Detalhado de Resposta aos Eventos

O usuário/orientador indicou a não-necessidade de detalhamento dos Diagramas de Fluxo de Dados.

2.5.3 Dicionário de Dados

O **Dicionário de Dados** encontra-se no CD-ROM entregue com esta documentação. Mais informações podem ser encontradas no capítulo 5, *Documentos Adicionais*.

2.5.4 Modelo Lógico de Dados

O **Diagrama Entidade-Relacionamento** (DER) para o sistema encontra-se na figura 16. Ele contém os depósitos "ARQUIVOS_ABERTOS", "LISTA_PROCESSOS", "TABELA_ARQUIVOS", "ERRNO" e "MEMORIA".

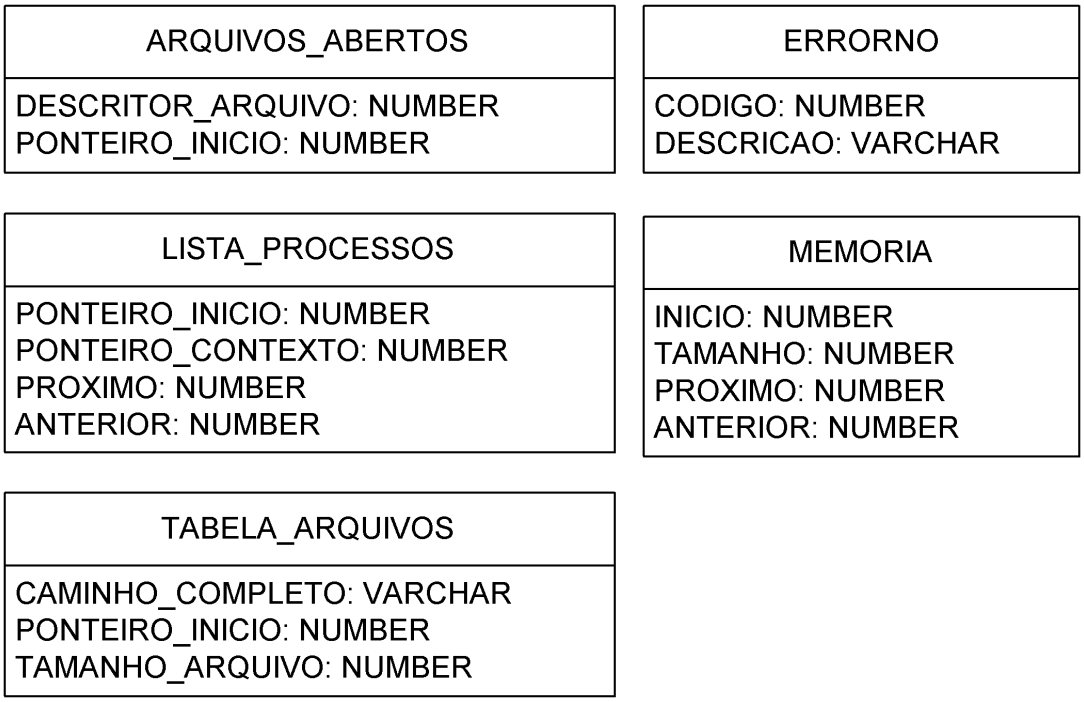

Figura 16: Diagrama Entidade-Relacionamento

2.5.5 Diagrama de Transição de Estados

As transições de estado do sistema encontram-se na **Tabela de Transição de Estados** (tabela 2 na próxima página) e também, ilustradas no **DTE** (figura 17 na página seguinte). Para uma melhor visualização, foi incluída no CD-ROM entregue com esta documentação uma simulação utilizando redes de Petri (vide capítulo 5).

Tabela 2: Tabela de Transição de Estados

Estado Inicial	Estado Final	Evento	Ação
-	Aguardando chamada de função	*boot*	Carregar o *kernel*
Aguardando chamada de função	Trocando entre processos	**bnnk_addproc()** chamada	Adicionar processo
Aguardando chamada de função	Aguardando chamada de função	função $\neq$ **bnnk_addproc()**	Executar função
Trocando entre processos	Trocando entre processos	função $\neq$ **bnnk_delproc()**	Executar função
Trocando entre processo	Trocando entre processos	**bnnk_delproc()** chamada com número de processos > 1	Remover processo
Trocando entre processos	Aguardando chamada de função	**bnnk_delproc()** chamada com número de processos $= 1$	Remover processo

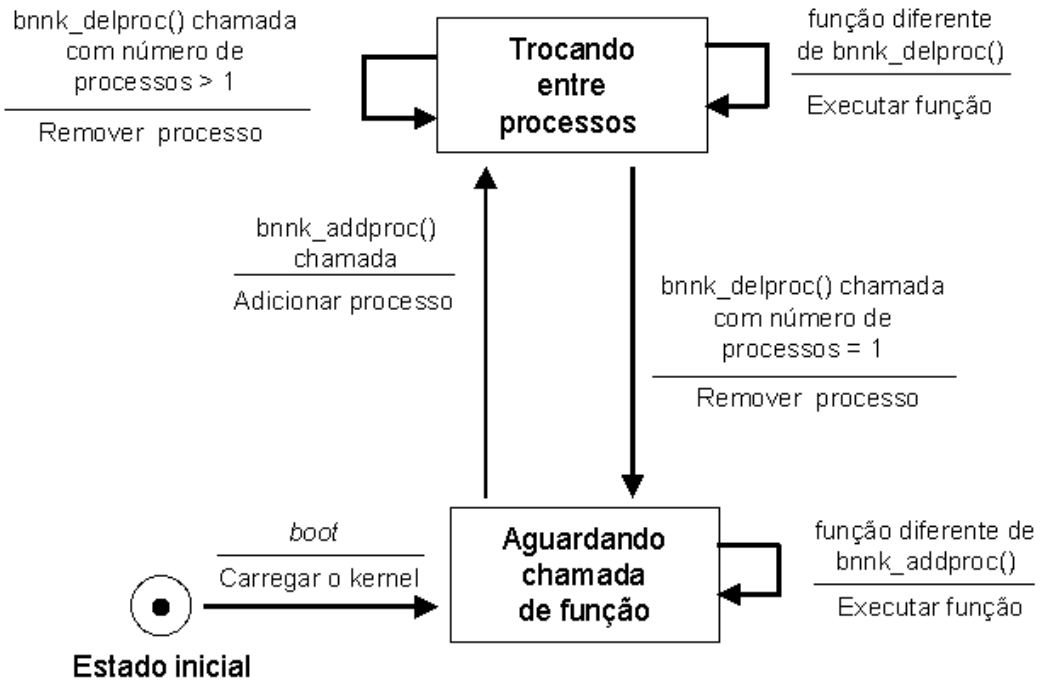

Figura 17: Diagrama de Transição de Estados

2.5.6 Especificação dos Processos Primitivos

Nesta subseção serão apresentadas as especificações dos processos primitivos para as versões simples e intermediária, em pseudo-código comentado na linguagem C.

As funções chamadas dentro dos processos primitivos que começam com `bnnk_` fazem parte das funções definidas por outros processos primitivos. Aquelas que começam com `oskit_` fazem parte das funções fornecidas pelas bibliotecas do sistema OSKit e suas especificações podem ser encontradas em (The Flux Research Group, 2002). As demais funções são parte das bibliotecas da linguagem C, definidas em (American National Standards Institute, 1989), ou relatam procedimentos que devem ser criados dentro do *BananaKernel*, mas que não fazem parte dos processos primitivos.

2.5.6.1 Ação #1: Alocar memória OU Configurar *errno*

Pseudo-código comentado em C para a versão simples:

```
/*
 * Esta funcao recebe o tamanho do bloco desejado e retorna um ponteiro para
 * o bloco. Em caso de erro, retorna nulo e configura o errno
 */

void            *
bnnk_malloc(int tamanho)
{
        int             posicao = 0;

        posicao = ultima_posicao + ultimo_tamanho;      /* calcula a posicao atual
                                                         * somando o endereco da
                                                         * ultima posicao alocada com
                                                         * o ultimo tamanho */

        if (posicao + tamanho < top_da_memoria) {       /* se a posicao nao
                                                         * ultrapassar a memoria
                                                         * disponivel... */
                ultima_posicao = posicao;               /* atualize ultima_posicao */
                ultimo_tamanho = tamanho;               /* atualize ultimo_tamanho */
                adiciona_bloco(posicao, tamanho);       /* adiciona o bloco a lista de
                                                         * blocos alocados */
        } else {
                errno = codigo_erro_malloc;             /* senao, configure errno */
                return null;                            /* e retorne null */
        }

        return posicao;                                 /* retorna o ponteiro para a
                                                         * posicao */

}
```

Pseudo-código comentado em C para a versão intermediária:

```
/*
 * Esta funcao recebe o tamanho do bloco desejado e retorna um ponteiro para
 * o bloco. Em caso de erro, retorna nulo e configura o errno
 */

void            *
bnnk_malloc(int tamanho)
{
        struct pos_mem *bloco = 0;

        bloco = encontra_posicao(tamanho);              /* encontra_posicao percorre a
                                                         * lista encadeada de posicoes
                                                         * vazias e procura uma que
                                                         * possua o tamanho desejado ou
                                                         * maior */

        if (!bloco) {                                   /* se nao encontrou nenhuma... */
                errno = codigo_erro_malloc;             /* configure errno */
                return null;                            /* retorne null */
        } else {
                if (bloco->tamanho > tamanho) {         /* se o tamanho for maior que
                                                         * o escolhido */
                        cria_novo_bloco(bloco, tamanho); /* cria um novo bloco do
                                                         * tamanho total menos o
                                                         * tamanho escolhido */
                }
                remove_bloco(bloco);                    /* remove o bloco da lista de
                                                         * posicoes vazias */
                adiciona_bloco(bloco->posicao, tamanho); /* adiciona o bloco a lista de
                                                         * blocos alocados */
        }

        return bloco->posicao;                          /* retorna o ponteiro para a
                                                         * posicao */

}
```

2.5.6.2 Ação #2: Re-alocar memória OU Configurar *errno*

Pseudo-código comentado em C para a versão simples:

```
/*
 * Esta funcao recebe o tamanho para um bloco de memoria e o novo tamanho, e
 * retorna um ponteiro para um novo bloco.
 */

void            *
bnnk_realloc((void *) bloco, int tamanho)
{
        int             posicao = 0;

        if (bnnk_getsize(bloco) > tamanho) {        /* se o bloco novo eh menor
                                                     * que o atual */
                errno = codigo_erro_realloc;        /* configura errno */
                return null;                        /* retorne null */
        }
        posicao = bnnk_malloc(tamanho);             /* aloca um novo bloco com o
                                                     * tamanho desejado */

        if (!posicao) {                             /* se nao encontrou memoria
                                                     * disponivel... */
                errno = codigo_erro_realloc;        /* configura errno */
                return null;                        /* retorne null */
        }
        memcpy(posicao, bloco, bnnk_getsize(bloco)); /* copia o conteudo do bloco
                                                     * para o novo bloco */
        bnnk_free(bloco);                           /* libera o bloco realocado */
        adiciona_bloco(posicao, tamanho);           /* adiciona bloco a lista de
                                                     * blocos alocados */

        return bloco;                               /* retorna o ponteiro para a
                                                     * nova posicao */
}
```

Pseudo-código comentado em C para a versão intermediária:

```
/*
 * Esta funcao recebe o tamanho para um bloco de memoria e o novo tamanho, e
 * retorna um ponteiro para um novo bloco.
 */

void            *
bnnk_realloc((void *) bloco, int tamanho)
{
        int             novo_bloco = 0;

        if (bnnk_getsize(bloco) > tamnaho) {            /* se o novo bloco eh menor
                                                         * que o atual */
                errno = codigo_erro_realloc;            /* configura errno */
                return null;                            /* retorne null */
        }
        novo_bloco = bnnk_malloc(tamanho);              /* aloca um novo bloco com o
                                                         * tamanho desejado */

        if (!novo_bloco) {                              /* se nao encontrou memoria
                                                         * disponivel... */
                errno = codigo_erro_realloc;            /* configura errno */
                return null;                            /* retorne null */
        }
        /* copia o conteudo do bloco para o novo bloco */
        memcpy(novo_bloco->posicao, bloco, bnnk_getsize(bloco));

        bnnk_free(bloco->posicao);                      /* libera o bloco realocado */
        adiciona_bloco(novo_bloco, tamanho);            /* adiciona bloco a lista de
                                                         * blocos alocados */

        return bloco->posicao;                          /* retorna o ponteiro para a
                                                         * posicao */

}
```

2.5.6.3 Ação #3: Liberar memória alocada OU Configurar *errno*

Pseudo-código comentado em C para a versão simples:

```
/*
 * Esta funcao recebe o ponteiro para o bloco a ser liberado e retorna 0.
 * Em caso de erro, retorna -1 e configura o errno
 */

int
bnnk_free((void *) bloco)
{
        int             existe = 0;

        existe = remove_bloco(bloco);                   /* remove bloco da lista de
                                                         * blocos alocados */

        if (!existe) {                                  /* se o bloco nao existe na
                                                         * lista... */
                errno = codigo_erro_free;               /* configure errno */
                return -1;                              /* retorne -1 */
        }
        /*
         * nesta versao o bnnk_free() nao faz alteracoes na alocacao continua, pois
         * nao ha gerencia de pedacos (chunks) liberados, e toda alocacao
         * acontece sempre no final do ultimo bloco alocado, sem gerencia de
         * fragmentacao
         */

        return 0;
}
```

Pseudo-código comentado em C para a versão intermediária:

```
/*
 * Esta funcao recebe o ponteiro para o bloco a ser liberado e retorna 0.
 * Em caso de erro, retorna -1 e configura o errno
 */
int
bnnk_free((void *) bloco)
{
        struct pos_mem *novo_bloco = 0;
        struct pos_mem *bloco_vazio = 0;
        int             offset = 0;
        int             erro = 0;

        offset = bloco + bnnk_getsize(bloco);           /* o final do bloco pode ser o
                                                         * comeco de um bloco vazio */
        if ((bloco_vazio = encontra_bloco(offset))) {   /* se ha um bloco vazio no
                                                         * final do bloco */
                novo_bloco->posicao = bloco;            /* o novo_bloco comeca no
                                                         * nosso bloco */
                novo_bloco->tamanho = bloco_vazio->posicao +
                        bloco_vazio->tamanho;           /* e termina no final do
                                                         * proximo bloco vazio */
                erro = cria_novo_bloco(novo_bloco);     /* adicona o novo bloco a
                                                         * lista de blocos vazios */
        } else {
                novo_bloco->posicao = bloco;            /* o novo bloco eh igual ao
                                                         * bloco */
                novo_bloco->tamanho = offset;           /* e o tamanho eh o mesmo */

                erro = cria_novo_bloco(novo_bloco);     /* adicona bloco a lista de
                                                         * blocos vazios */
        }
        if (erro) {                                     /* se nao conseguiu adicionar
                                                         * o novo bloco */
                errno = codigo_erro_free;               /* configure errno */
                return -1;                              /* retorne -1 */
        }
        remove_bloco(bloco);                            /* remove bloco da lista de
                                                         * blocos alocados */
        return 0;
}
```

2.5.6.4 Ação #4: Informar memória disponível

Pseudo-código comentado em C para a versão simples:

```
/*
 * Esta funcao retorna a quantidade de memoria disponivel
 */

int
bnnk_avail(void)
{
        /*
         * basta subtrair o valor do ultimo bloco alocado do total de memoria
         * encontrado durante o boot, que estah na variavel phys_mem_max encontrada
         * em oskit/x86/pc/phys_lmm.h
         */

        return phys_mem_max - (ultima_posicao + ultimo_tamanho);
}
```

Pseudo-código comentado em C para a versão intermediária:

```
/*
 * Esta funcao retorna a quantidade de memoria disponivel
 */

int
bnnk_avail(void)
{
        int             disponivel = 0;
        struct pos_mem *bloco;

        bloco = primeiro_bloco;                 /* encontra o primeiro bloco
                                                 * vazio */

        while (bloco) {                         /* enquanto a lista nao acabar */
                disponivel += bloco->tamanho;   /* soma o tamanho do bloco */
                bloco = bloco->proximo;         /* e muda para o proximo */
        }
        return disponivel;                      /* retorna o tamanho
                                                 * disponivel */
}
```

2.5.6.5 Ação #5: Informar tamanho de bloco de memória alocada OU Configurar *errno*

Pseudo-código comentado em C para as versões simples e intermediária:

```
/*
 * Esta funcao retorna o tamanho de um bloco
 */

int
bnnk_getsize((void *) bloco)
{
        struct pos_mem *lista_blocos = primeiro_bloco;      /* recebe o endereco do
                                                             * primeiro bloco da lista
                                                             * encadeada de blocos
                                                             * alocados */

        while (lista_blocos) {
                if (&lista_blocos == &bloco) {              /* se o bloco percorrido eh o
                                                             * bloco desejado... */
                        break;
                } else {
                        lista_blocos = lista_blocos->proximo;
                }
        }

        if (!lista_blocos) {                                /* se percorremos todos os
                                                             * blocos e nao encontramos o
                                                             * bloco desejado, quer dizer
                                                             * que o endereco estah errado */
                errno = codigo_erro_getsize;                /* configure errno */
                return 0;                                   /* retorne 0 */
        }
        return lista_blocos->tamnaho;
}
```

2.5.6.6 Ação #6: Abrir arquivo OU Configurar *errno*

Pseudo-código comentado em C para a versão simples:

```
/*
 * Esta funcao recebe o caminho completo para um arquivo e devolve um descritor
 * para arquivo. Em caso de erro, retorna zero e configura o errno
 */

int
bnnk_open((char *) caminho)
{
        int             descritor = 0;
        struct pos_arq  arquivos = primeiro_arquivo;

        while (arquivos != NULL) {                          /* percorre a tabela de
                                                             * arquivos */
                if (!(strcmp(arquivos->caminho, caminho))) {  /* se encontrar... */
                        descritor = arquivos->descritor;    /* salva o descritor */
                        break;                              /* e sai do while */
                }
                arquivos = arquivos->proximo;
        }

        if (!descritor) {                                   /* se nao encontrou o
                                                             * arquivo */
                if (descritor = cria_novo_arquivo(caminho)) {  /* se nao conseguir
                                                             * criar um arquivo novo
                                                             * no final da tabela */
                        errno = codigo_erro_open;           /* configure errno */
                        return null;                        /* e retorne null */
                }
        }
        return descritor;                                   /* retorna o descritor
                                                             * para o arquivo */

}
```

Pseudo-código comentado em C para a versão intermediária:

```
/*
 * Esta funcao recebe o caminho completo para um arquivo e devolve um descritor
 * para arquivo. Em caso de erro, retorna zero e configura o errno.
 * Assume-se que o sistema de arquivos jah estah aberto
 */

int
bnnk_open((char *) caminho)
{
        oskit_blkio_t     *descritor = 0;
        oskit_error_t      erro;

        /*
         * o fsread_ffs_open recebe o descritor para o disco a ser utilizado,
         * o caminho para o arquivo e o descritor em que se deve escrever.
         * Retirado de
         * http://www.cs.utah.edu/flux/oskit/html/oskit-wwwch34.html#x48-29880
         * 0034.4.2
         */
        erro = fsread_ffs_open(hdb1, caminho, &descritor);

        if (erro != 0) {
                errno = codigo_erro_open;                /* configure errno */
                return null;                             /* e retorne null */
        }
        return descritor                                 /* retorna o descritor para o
                                                          * arquivo */
}
```

2.5.6.7 Ação #7: Ler arquivo aberto OU Configurar *errno*

Pseudo-código comentado em C para as versões simples e intermediária:

```
/*
 * Esta funcao recebe o descritor para um arquivo e um buffer para escrever o
 * conteudo deste arquivo, e retorna a quantidade de bytes lidos, ou 0 em erro.
 */

int
bnnk_read(int descritor, (char *) buffer)
{
        char            entrada;
        int             i = 0;
        int             lidos = 0, atual = 0;
        oskit_error_t   erro;

        /*
         * utiliza-se a funcao do oskit para ler diretamente do disco. Esta
         * funcao recebe o descritor, o buffer para onde escrever, o offset,
         * o tamanho do buffer e um ponteiro para um indicador da posicao
         * atual
         */
        if (!(erro = oskit_blkio_read(descritor, entrada, lidos, 1, &atual))) {
                lidos += atual;                         /* lidos aumenta com a leitura */
        } else {
                errno = codigo_erro_read;               /* configure errno */
                return 0;                               /* e retorne null */
        }
        while (entrada = !EOF) {                        /* enquanto nao chegamos ao
                                                         * fim do arquivo */
                buffer[i] = entrada;                    /* copia a entrada para buffer */
                if (!(erro = oskit_blkio_read(descritor, entrada, lidos, 1, &atual))) {
                        lidos += atual;                 /* lidos aumenta com a leitura */
                } else {
                        errno = codigo_erro_read;       /* configure errno */
                        return 0;                       /* e retorne null */
                }
        }
        return lidos;                                   /* retorna a quantidade de
                                                         * bytes lidos */
}
```

2.5.6.8 Ação #8: Escrever em arquivo aberto OU Configurar *errno*

Pseudo-código comentado em C para as versões simples e intermediária:

```
/*
 * Esta funcao recebe o descritor para um arquivo, um buffer de onde ler o
 * conteudo e a quantidade de bytes a escrever, e retorna a quantidade de bytes
 * lidos. Em caso de erro, retorna zero e configura o errno
 */

int
bnnk_write(int descritor, (char *) buffer, int bytes)
{
        int             i = 0;
        int             escritos = 0;
        oskit_error_t   erro;

        /*
         * utiliza-se a funcao do oskit para escrever diretamente no disco. Esta
         * funcao recebe o descritor, o buffer de onde ler, o offset,
         * o tamanho do buffer e um ponteiro para um indicador da posicao
         * atual
         */
        erro = oskit_blkio_write(descritor, buffer, 0, bytes, &escritos)
                if (erro) {                             /* se nao conseguiu escrever */
                errno = codigo_erro_write;              /* configure errno */
                return 0;                               /* e retorne zero */
        }
        return escritos;                                /* retorna a quantidade de
                                                         * bytes lidos */
}
```

2.5.6.9 Ação #9: Fechar arquivo OU Configurar *errno*

Pseudo-código comentado em C para a versão simples:

```
/*
 * Esta funcao recebe o descritor para arquivo e retorna 0.
 * Em caso de erro, retorna -1 e configura o errno
 */

int
bnnk_close(int descritor)
{
        /*
         * o bnnk_close() na verdade nao faz nada em gravacao continua de
         * disco, pois a escrita eh direta e o descritor eh controlado pela
         * tabela de alocacao. Entao, deve-se apenas garantir que todos os
         * buffers tenham sido escritos
         */

        fflush();                                   /* faz um flush de todos os
                                                     * buffers */

        return 0;                                   /* retorna zero */

}
```

Pseudo-código comentado em C para a versão intermediária:

```
/*
 * Esta funcao recebe o descritor para arquivo e retorna 0.
 * Em caso de erro, retorna -1 e configura o errno
 */

int
bnnk_close(int descritor)
{
        int             erro;

        /*
         * oskit_blkio_release() recebe um descritor e escreve todos os
         * metadados pertinentes a esse descritor no filesystem adequado
         * (neste casso ext2fs)
         */
        erro = oskit_blkio_release(descritor);

        if (erro != 0) {
                errno = codigo_erro_close;          /* configure errno */
                return -1;                          /* retorne -1 */
        }
        return 0;                                   /* retorna zero */

}
```

2.5.6.10 Ação #10: Incluir novo processo OU Configurar *errno*

Pseudo-código comentado em C para as versões simples e intermediária:

```
/*
 * Esta funcao recebe um ponteiro para processo e retorna um descritor para
 * este processo
 */

int
bnnk_addproc(struct proc * processo)
{
        /*
         * processo eh uma estrutura contendo, entre outras coisas, um
         * apontador para o processo carregado na memoria
         */

        /* posiciona-se o novo processo no final da fila */
        ultimo_processo->proximo = processo;
        processo->anterior = ultimo_processo;
        processo->descritor = (ultimo_processo->descritor) + 1

        /*
         * nao ha erro configurado, o processo eh sempre adicionado no final
         * da fila. Se faltar memoria, quem irah cuidar disso eh o
         * bnnk_malloc, quando for ser criado o espaco para este processo
         */

                return processo->descritor;
}
```

2.5.6.11 Ação #11: Remover processo OU Configurar *errno*

Pseudo-código comentado em C para as versões simples e intermediária:

```
/*
 * Esta funcao recebe um descritor para processo e o remove da lista de
 * processos
 */

int
bnnk_delproc(int processo)
{
        struc proc        processos = primeiro_processo;

        while (processos->descritor != processo)          /* busca na lista de processos */
                processos = processos->proximo;

        if (!processos) {                                 /* se nao encontramos um
                                                           * processo com esse descritor */
                errno = codigo_erro_addproc;              /* configure errno */
                return -1;                                /* retorne -1 */
        } else {
                /* remove-se o processo da lista de processos */
                processos->anterior->proximo = processos->proximo;
                processos->proximo->anterior = processos->anterior;

        }

        /*
         * o processo nao eh removido da memoria, isto eh de responsabilidade
         * da funcao que chamou bnnk_delproc()
         */

        return 0;                                         /* retorne zero */

}
```

2.5.6.12 Ação #12: Trocar processo

Pseudo-código comentado em C para a versão simples:

```
/*
 * Esta funcao troca o processo em execucao
 */

void
bnnk_swapproc()
{
        desliga_interrupt();                    /* esta funcao nao pode ser
                                                 * interrompida */

        salva_contexto(corrente->contexto);     /* salva o contexto do
                                                 * processo em uma regiao da
                                                 * memoria, e aponta
                                                 * corrente->contexto para
                                                 * este endereco */

        corrente = corrente->proximo;           /* vai para o proximo processo
                                                 * da lista circular encadeada */

        carrega_contexto(corrente->contexto);   /* carrega o novo contexto e
                                                 * faz um ljmp ou uma lcall
                                                 * para o far pointer do novo
                                                 * EIP, liberando os
                                                 * interrupts */

}
```

Pseudo-código comentado em C para a versão intermediária:

```
/*
 * Esta funcao troca o processo em execucao
 */

void
bnnk_swapproc()
{
        struct proc     processos = corrente->proximo;
        struct proc     proximo = corrente->proximo;

        desliga_interrupt();                            /* esta funcao nao pode ser
                                                         * interrompida */

        salva_contexto(corrente->contexto);             /* salva o contexto do
                                                         * processo em uma regiao da
                                                         * memoria, e aponta
                                                         * corrente->contexto para
                                                         * este endereco */

        while (processos != corrente) {                 /* varre a lista de processos */
                processos->idade++;                     /* envelhece o processo */
                if ((proximo->idade + proximo->prioridade) >
                    (processos->idade + processos->prioridade))
                        proximo = processos;            /* encontra o processo de
                                                         * maior prioridade */
                processos = processos->proximo;         /* muda de processo */
        }

        corrente = proximo;                             /* vai para o proximo processo
                                                         * de maior prioridade da
                                                         * lista circular encadeada */

        carrega_contexto(corrente->contexto);           /* carrega o novo contexto e
                                                         * faz um ljmp ou uma lcall
                                                         * para o far pointer do novo
                                                         * EIP, liberando os
                                                         * interrupts */

}
```

2.5.6.13 Ação #13: Imprimir mensagem de erro

Pseudo-código comentado em C para as versões simples e intermediária:

```
/*
 * Esta funcao imprime a ultima mensagem de erro associada ao errno
 */

void
bnnk_perror(void)
{
        /*
         * basta imprimir a mensagem de erro na posicao correta do vetor de
         * (char *) que contem todas as mensagens de erro
         */
        printf("Erro %i: %s", errno, erros[errno]);

        /*
         * pode ser implementada tambem para retornar a mensagem de erro para
         * o cliente que a chamou, mas estah implementada desta forma para
         * ter uma semelhanca maior a perror() do C, sem valor de retorno
         */

}
```

2.6 Protótipo

O protótipo deste sistema encontra-se no documento entregue com este, e também disponível em `http://bananakernel.sourceforge.net/manual/`, intitulado ***BananaKernel*: Um Sistema Operacional Didático - Manual do Usuário**, contendo um *draft* do conteúdo que irá compor o manual do usuário. Mais informações podem ser encontradas no capítulo 5, *Documentos Adicionais*.

3 Riscos

O quadro de riscos foi efetuado com base na metodologia de probabilidade e impacto dos riscos envolvidos no projeto. Para calcular um fator de risco para o projeto, foram adotados valores de grandeza para cada risco, tornando mais compreensiva a probabilidade do risco vir a ocorrer e o impacto (peso) que o mesmo causaria no andamento do projeto. A nomenclatura dos valores utilizados encontra-se na tabela 3.

Tabela 3: Nomenclatura dos valores de Risco

Valor Associado	Significado
1	Muito baixo
2	Baixo
3	Médio
4	Alto
5	Muito alto

Dentro desta nomenclatura, podemos definir cada risco associado ao projeto, como na tabela 4 na página seguinte. O risco total do projeto é dado pela média ponderada dos riscos. A média ponderada é calculada pelo somatório da probabilidade de ocorrência, multiplicada pelo impacto do risco de cada ítem relacionado na tabela 4, dividido pelo somatório do impacto do risco destes ítens, ou $\frac{\sum P*I}{\sum I}$, em que P é a probabilidade de ocorrência, I é o impacto do risco e S é a severidade da ocorrência, calculado como $P * I$.

Tabela 4: Quantificação dos Riscos

#	**Risco**	**P**	**I**	**S**	**Explicação**	**Medida Preventiva**	**Medida Corretiva**
1	A equipe de desenvolvimento não domina a tecnologia utilizada	2	3	6	A falta de domínio da tecnologia pode atrasar, ou impossibilitar a execução do projeto	Levantar documentação necessária e efetuar treinamento na tecnologia empregada	Procurar ajuda de especialistas em listas de discussão relacionadas ao tópico envolvido
2	Impossibilidade de prever quais as necessidades do projeto	1	3	3	Não prever todas as necessidades pode levar a muitas mudanças	Efetuar verificações constantes com o orientador dos dados coletados no levantamentos de requisitos	Marcar reunião com o orientador para verificar quais alterações devem ser feitas nos requisitos levantados para adequá-los às necessidades do projeto
3	Os requisitos do projeto mudam constantemente ao longo do seu ciclo de vida	1	3	3	Mudanças bruscas nos requisitos podem aumentar o custo e o tempo de execução	Ter um escopo claro e bom levantamento de requisitos	Executar uma rodada de levantamento de requisitos, preparar um documento relatando os requisitos encontrados, e pedir para o orientador assinar este documento, indicando aceitação dos requisitos levantados
4	A equipe de desenvolvimento não possui experiência prévia neste tipo de projeto	1	2	2	A falta de experiência da equipe pode levar a um mal entendimento técnico do projeto	Garantir que um orientador experiente supervisione o projeto	Procurar ajuda de um co-orientador com a experiência necessária
5	A equipe de desenvolvimento não possui treinamento nas ferramentas utilizadas	1	2	2	A falta de treinamento pode interferir no planejamento e execução	Levantar os documentos necessários e efetuar treinamento na ferramenta empregada	Procurar ajuda de especialistas em listas de discussão relacionadas à ferramenta envolvida
6	Existem projetos em paralelo que podem afetar o desenvolvimento deste projeto	1	1	1	Projetos em paralelo podem modificar as ferramentas necessárias para a execução	Manter-se atualizado com as listas de discussão e de notificação de mudanças das ferramentas	Conseguir, com os desenvolvedores terceiros, a lista de modificações e adequar o projeto a elas
						Risco total do projeto (média ponderada)	**2,833**

Com base na metodologia de análise de risco adotada, conclui-se que o risco do projeto é suficientemente baixo, como mostra a tabela 5.

Tabela 5: Riscos X Percentagens

Valor Associado	Faixa de Percentagem	Significado
1	0% - 5%	Muito baixo
2	5% - 10%	Baixo
3	10% - 30%	Médio
4	30% - 60%	Alto
5	60% - 100%	Muito alto

O valor total (2,833) posiciona o projeto na faixa de risco próxima a 10%.

Com um melhor conhecimento sobre o projeto, foi possível encontrar um valor mais próximo ao risco real. Na primeira análise (TEIGÃO; MORIMOTO, 2004b), o valor encontrado foi 1,334, posicionando o projeto na área de risco "muito baixo". Nesta revisão, o risco subiu para 2,833, posicionando-o na área "baixo", porém já muito próximo à área "médio".

Com um risco próximo a 10%, espera-se que algumas ações de contingência necessitem ser acionadas.

O fator de risco com maior severidade, que posiciona este projeto próximo ao risco médio, é o "Risco #1", com severidade 6. Muitos documentos foram levantados na fase de preparação para a construção deste projeto lógico, porém, o domínio de toda a tecnologia necessária só se provará completo durante a fase de implementação.

Então, em preparação para acionar a ação de contingência do "Risco #1", foi reservada uma semana no cronograma (vide capítulo 4) para levantar as listas de discussão freqüentadas pelos principais especialistas nos tópicos mais relevantes ao projeto.

4 Cronograma

A figura 18 contém o cronograma simplificado para este projeto. Por se tratar de um cronograma extenso, neste documento foram incluídos apenas os ítens que compõem as principais data do cronograma, além dos períodos utilizados para as medidas de prevenção de riscos.

Para visualização do cronograma completo, que inclui o Diagrama de Gantt, consulte o CD-ROM entregue com este documento (vide capítulo 5 na próxima página).

	Responsável	Tarefa	Início	Fim
0	Rafael & Julio	**Estudo de Viabilidade**	12/02/2004	01/04/2004
1	Rafael & Julio	**Modelo Ambiental**	02/04/2004	12/04/2004
2	Julio	*Diagrama de contexto*	02/04/2004	07/04/2004
3	Rafael	*Lista de eventos*	07/04/2004	12/04/2004
4	Rafael & Julio	**Modelo Comportamental**	12/04/2004	24/05/2004
5	Julio	*DFD Individual de resposta aos eventos*	12/04/2004	20/04/2004
6	Julio	*DFD Detalhado de resposta aos eventos*	20/04/2004	02/05/2004
7	Julio	*Dicionário de Dados*	03/05/2004	09/05/2004
8	Julio	*Modelo Lógico de Dados*	10/05/2004	19/05/2004
9	Julio	*Diagrama de Transição de Estados*	20/05/2004	24/05/2004
10	Rafael	*Especificação dos Processos Primitivos*	02/05/2004	12/05/2004
11	Rafael	**Correções nos riscos sugerida pela banca**	13/05/2004	24/05/2004
12	Rafael	**Redação do texto**	24/05/2004	28/05/2004
13	Rafael & Julio	**Cronograma**	23/05/2004	24/05/2004
14	Julio	**Documentos adicionais**	24/05/2004	07/06/2004
15	Julio	*CD-ROM Interativo*	24/05/2004	07/06/2004
16	Rafael	*Manual do Usuário*	28/05/2004	07/06/2004
17	Rafael & Julio	**MP: Levantar doc. necessária e efetuar treinamento**	27/04/2004	29/04/2004
18	Rafael & Julio	**MP: Reuniões de acompanhamento c/ orientador**	27/04/2004	22/10/2004
19	Rafael & Julio	**MP: Reuniões de esclarecimentos c/ prof. da disc.**	27/04/2004	22/10/2004
20	Rafael & Julio	**MP: Eslcarecer escopo e levantar requisitos**	30/04/2004	02/05/2004
21	Rafael & Julio	**MP: Manter-se atualizado com as listas de discussão da ferramenta**	27/04/2004	22/10/2004
22	Rafael & Julio	**Entrega dos documentos no Eureka**	07/06/2004	07/06/2004
23	Rafael & Julio	**Defesa da Modelagem e do Plano de Testes**	17/06/2004	17/06/2004
24	Rafael & Julio	**Revisão sugerida nos documentos**	17/06/2004	02/07/2004
25	Rafael & Julio	**Implementação**	12/07/2004	06/09/2004
26	Rafael & Julio	**Testes e correção de bugs**	07/09/2004	22/10/2004

Figura 18: Cronograma

5 *Documentos Adicionais*

Foram entregues, com este Projeto Lógico, três documentos adicionais: um CD-ROM, um Manual do Usuário e um Plano de Testes.

5.1 CD-ROM

O CD-ROM contém:

- o **Dicionário de Dados**, em que os fluxos de dados encontrados em cada **DFD** individual podem ser clicados com o *mouse* para visualizar suas definições, além da Especificação do Dicionário de Dados, que fornece as definições primitivas utilizadas, e outras possibilidades de navegação, como pela **Tabela de Eventos** ou **Diagrama de Contexto**;
- a simulação, em redes de Petri, no formato *Audio Video Interleave* (AVI) para o **DTE**;
- o **Cronograma** completo, com o **Diagrama de Gantt**; e
- o **Manual do Usuário** no formato *Hypertext Markup Language* (HTML).

Para acessar os dados contidos no CD-ROM, é necessário um equipamento com as seguintes características mínimas:

- leitor de CD-R (CD-ROM gravável);
- 64 MB de RAM;
- Sistema Operacional Microsoft Windows 95/98/Me/XP/2000;
- *browser* de Internet:

  - Microsoft Internet Explorer 5.5 ou superior; ou
  - Netscape Navigator 4.7 ou superior;

- Microsoft Windows Media Player, para a visualização da simulação no formato *Audio Video Interleave*, pode ser encontrado em (Microsoft Corporation, 2003), ou outro decodificador deste formato.

O CD-ROM deve inicializar o *browser* e carregar o menu principal automaticamente, após ser inserido no leitor. Caso isso não ocorra, deve-se acessar a unidade do leitor (e.g. D:) através do Microsoft Windows Explorer, e efetuar um clique duplo no arquivo `bnnk.exe`, para carregar o menu principal.

A partir do menu, é possível acessar o **Dicionário de Dados**, as duas simulações em redes de Petri e o **Manual do Usuário**.

O arquivo `README.txt` incluso no CD-ROM fornece essas instruções sobre o seu funcionamento e fornece contatos que podem sanar eventuais dúvidas ou auxiliar na resolução de problemas quanto ao acesso aos dados.

5.2 Manual do Usuário

Destinado ao aluno, o Manual do Usuário (TEIGÃO; MORIMOTO, 2004a) deve servir como um guia para encontrar as funções que o *BananaKernel* oferece, e como um tutorial passo-a-passo para a criação de um ambiente de desenvolvimento e execução.

O sumário resumido do manual é:

- Conceitos
  - Introdução
  - Estrutura do *BananaKernel*
  - Alterando Partes do Sistema
- Ambiente de Desenvolvimento e Execução
  - Criando um Ambiente de Desenvolvimento
  - Executando

- Principais Funções do *OSKit*
  - Inicialização
  - Gerência de Memória
  - Sistemas de Arquivos
  - Funções Gerais
- Apêndices

Este manual trata apenas de questões referentes ao sistema, não trazendo material específico para a aula de *Sistemas Operacionais* e não é destinado para o ensino desta disciplina, mas, sim, como um apoio à utilização do sistema.

É importante enfatizar que o manual será disponibilizado em dois formatos: no formato livro, em *Portable Document Format* (PDF), para ser impresso frente-e-verso, e no formato *web*, em *Hypertext Markup Language* (HTML), para servir de consulta rápida.

5.3 Plano de Testes

O Plano de Testes fornece a especificação para a execução de testes sobre os requisitos funcionais e não-funcionais.

Os testes descritos no Plano de Testes têm como objetivo assegurar que:

- as funções descritas neste Projeto Lógico, sendo chamadas em um ambiente adequado e recebendo os parâmetros corretos, executam o código esperado e fornecem a resposta correta em retorno; e
- um aluno de um curso universitário de computação, cursando a disciplina *Sistemas Operacionais* e com domínio da linguagem C, possa executar o *kernel* exemplo e alterar e criar módulos, utilizando como documentação do *BananaKernel* apenas o **Manual do Usuário** (TEIGÃO; MORIMOTO, 2004a).

6 Conclusão

A criação de um sistema operacional é um projeto que depende de diversas áreas de conhecimento da Ciência da Computação, e envolve escolhas de algoritmos e métodos que são claramente mais interessantes do que suas traduções, com otimizações e simplificações, para uma linguagem de programação.

Uma das vantagens encontrada na utilização de um sistema operacional didático é o comprometimento com a teoria, e não com a execução. Isso permite que o usuário concentre-se no algoritmo e no método empregado, e não nas suas propriedades de execução. É irrelevante para o ensino do funcionamento básico de um gerenciador de memória, por exemplo, se um determinado código permite o uso de *trampolins* ao ser compilado.

A criação deste projeto lógico levou em consideração essa característica importante que deve estar presente em um SO didático: a clareza da implementação.

Para a preparação da **Especificação dos Processos Primitivos**, subseção 2.5.6, vários conceitos do sistema foram testados, programando chamadas independentes às implementações parciais das ações, o que levou a um melhor conhecimento do projeto e dos requisitos de desenvolvimento.

A ação que mostrou-se mais trabalhosa foi a **Trocar processo**, porém, após estudar o capítulo 6 de (Intel Corporation, 1997), pode-se ter uma melhor compreensão sobre o funcionamento da troca de contexto dentro do processador, o que permitiu o desenvolvimento do pseudo-código para esta ação.

A análise de risco apresentada em (TEIGÃO; MORIMOTO, 2004b) foi revisada, e a apresentação no capítulo 3, considerando o que foi aprendido durante o desenvolvimento deste projeto lógico, mostrou que o projeto é interessante.

Considerando esses pontos apresentados e os novos riscos levantados, este projeto continua sendo considerado viável.

7 *Responsabilidades*

Curitiba, 09 de junho de 2004.

São responsáveis por este Projeto Lógico:

Alunos:

Rafael Coninck Teigão

Julio Henrique Morimoto

Orientador:

Prof. Dr. Carlos Alberto Maziero

ANEXO A – Descrição Detalhada do Escopo

Neste anexo estão descritos em detalhes os requisitos dos gerenciadores de memória e arquivos e do escalonador.

A.1 Escopo da Gerência de Memória

O gerenciador de memória deve disponibilizar as funções:

`bnnk_malloc()` responsável pela alocação de memória;

`bnnk_realloc()` responsável pela re-alocação de memória;

`bnnk_free()` responsável pela liberação de uma parte da memória;

`bnnk_avail()` retorna a quantidade de memória disponível; e

`bnnk_getsize()` retorna o tamanho de um volume de memória reservado.

Os nomes, apesar de estarem em inglês, mantém uma coerência com aqueles empregados na linguagem C, e são familiares aos alunos.

Essas funções devem estar disponíveis em duas versões:

- alocação contínua; e
- alocação baseada em listas (TANENBAUM, 2001).

A.2 Escopo da Gerência de Arquivos

O gerenciador de arquivos deve disponibilizar as funções:

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; e

bnnk_close() fecha o arquivo aberto em disco.

Novamente, os nomes estão em inglês para manter coerência com aqueles utilizados na linguagem C.

Essas funções devem estar disponíveis em duas versões:

- escrita contínua em disco; e
- escrita utilizando o *ext2fs* (CARD; TS'O; TWEEDIE, 1994).

A.3 Escopo do Escalonador de Processos

O escalonador de processos deve fornecer duas funções:

bnnk_addproc() inclui um novo processo á lista de processos;

bnnk_delproc() remove um processo desta lista; e

bnnk_swapproc() muda o contexto para o próximo processo.

Os nomes foram mantidos em inglês para estarem coerentes com as demais funções implementadas.

O escalonador de processos também estará disponível em duas versões:

- escalonador *FIFO*[1], sem prioridade ou envelhecimento;
- escalonador com prioridade e envelhecimento.

[1] *First In - First Out* (primeiro a entrar, primeiro a sair).

Referências Bibliográficas

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

CARD, R.; TS'O, T.; TWEEDIE, S. Design and implementation of the second extended filesystem. In: STATE UNIVERSITY OF GRONIGEN. *Proceedings of the First Dutch International Symposium on Linux.* 1994. Disponível em: <http://e2fsprogs.sourceforge.net/ext2intro.html>. Acesso em: 28 de abril 2004.

Intel Corporation. *Intel Architecture Software Developer's Manual.* P.O. Box 7671 Mt. Prospect IL 60056-7641 USA, 1997. v. 3: System Programming Guide. Disponível em: <http://www.intel.com/design/mobile/manuals/24319201.pdf>. Acesso em: 11 de maio de 2004.

Microsoft Corporation. *Microsoft Windows Media Player.* 2003. Disponível em: <http://www.microsoft.com/windows/windowsmedia/default.aspx>. Acesso em: 17 de maio de 2004.

TANENBAUM, A. Memory management with linked lists. In: VRIJE UNIVERSITEIT. *Modern Operating Systems.* 2^a. ed. [S.l.]: Prentice Hall, 2001. cap. 4.2.2, p. 200–201.

TEIGÃO, R. C.; MORIMOTO, J. H. BananaKernel*: Um Sistema Operacional Didático - Manual do Usuário.* [S.l.], 2004. Disponível em: <http://bananakernel.sourceforge.net/manual/>. Acesso em: 20 de maio de 2004.

TEIGÃO, R. C.; MORIMOTO, J. H. *Estudo de Viabilidade* - BananaKernel*: Um Sistema Operacional Didático.* [S.l.], marco 2004. Disponível em: <http://bananakernel.sourceforge.net/docs/EV.pdf>. Acesso em: 11 de maio de 2004.

The Flux Research Group. *The OSKit: The Flux Operating System Toolkit.* Version 0.97 (snapshot 20020317). Salt Lake City, UT, USA 84112, March 2002. Disponível em: <http://www.cs.utah.edu/flux/oskit/html/oskit-www.html>. Acesso em: 10 de maio de 2004.

Índice Remissivo

Diagrama
de Contexto, 6
de Fluxo de Dados, 8, 44
de Gantt, 43, 44
de Transição de Estados, 17, 44
Entidade-Relacionamento, 16

Escalonador de Processos
funções, 50

Função bnnk_
addproc(), 14, 18, 34, 50
avail(), 10, 26, 49
close(), 13, 32, 50
delproc(), 15, 18, 35, 50
free(), 9, 24, 49
getsize(), 11, 27, 49
malloc(), 8, 20, 49
open(), 11, 28, 50
perror(), 3, 16, 38
read(), 12, 30, 50
realloc(), 9, 22, 49
swapproc(), 15, 36, 47, 50
write(), 13, 31, 50

Gerenciador de Arquivos
funções, 49
Gerenciador de Memória
funções, 49

Levantamento de Requisitos
reunião, 4

Manual do Usuário, 3, 5, 39, 44–46

Redes de Petri, 17