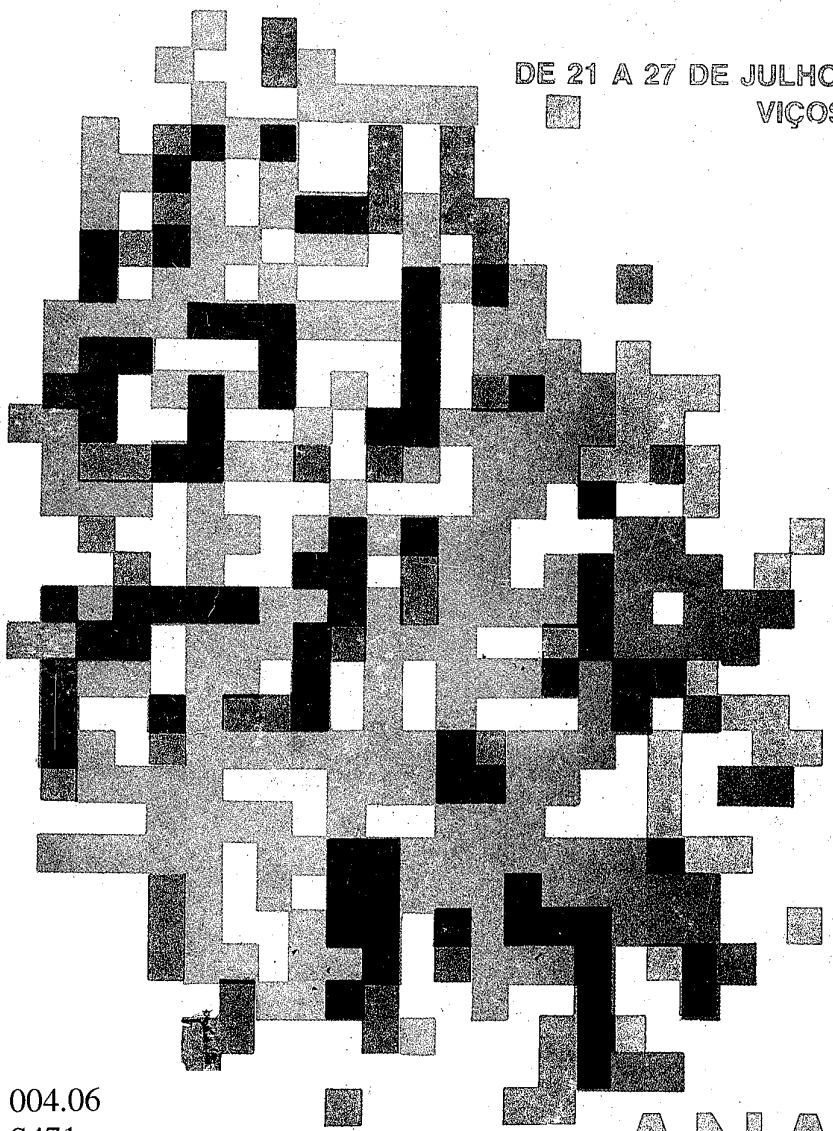
IV CONGRESSO

DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO

DE 21 A 27 DE JULHO DE 1984

VIÇOSA — MG



ANAIS

VOL. I

IV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO
VIÇOSA 21 a 27 de julho de 1984

ANAIS

VOLUME I

TRABALHOS APRESENTADOS

XI SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE

EDITORES:
R.S. BIGONHA, L.J. BRAGA-FILHO, A.M. OLIVEIRA & C.E. RECH

PROMOÇÃO: SOCIEDADE BRASILEIRA DE COMPUTAÇÃO
UNIVERSIDADE FEDERAL DE VIÇOSA - UFV

PATROCÍNIO: CAPES CNPq CTI FINEP SEI

OPERAÇÕES DE ABSTRAÇÃO PARA RECUPERAÇÃO DE INFORMAÇÃO: UMA IMPLEMENTAÇÃO UTILIZANDO ÁRVORES B^+

EWERTON A. P. VIEIRA *

MARIO A. M. GUIMARÃES **

SUMÁRIO

O trabalho descreve o projeto e implementação de tres níveis de operações de abstração:

1. Método de Acesso Paginado (MAP), responsável pelo mapeamento em memória secundária de páginas lógicas;
2. Método de Acesso Indexado (MAI), responsável pela manipulação de registros de tamanho variável contendo chaves únicas também de tamanho variável; e
3. Método de Acesso Invertido (MAV), responsavel pela criação e manipulação de listas invertidas.

O ambiente destino foi o sistema operacional CP/M nas versões 8 e 16 (CP/M-86) bits e a linguagem utilizada, C. O primeiro nível, o Método de Acesso Paginado, gerencia a alocação de novas páginas e a liberação de páginas descartadas, promovendo sua reutilização quando necessário. O algoritmo de alocação de áreas em memória principal é do tipo LRU minimizando o número de acessos à memória secundária. O segundo é construido sobre o primeiro, implementando as primitivas de acesso indexado. A estrutura de armazenamento escolhida, árvores B^+, minimiza o número de acessos na pesquisa direta a um registro e permite também uma eficiente pesquisa sequencial nos registros que ficam naturalmente em ordem crescente. O tàmanho variável de registros e chaves permite também uma maior eficiência no que diz respeito à utulização de espaço e, portanto, na altura da árvore de acesso. Por fim, o terceiro nível, o Método de Acesso Invertido, construido sobre o primeiro e segundo níveis, implementa operações primitivas definidas sobre listas invertidas armazenadas em memória de masssa.

Palavras Chave: Arquivo indexado, Arvores B^+, CP/M, Linguagem C, Método de acesso e Lista invertida.

ABSTRACT

This paper describes the design and implementation of three levels of abstract operations:

1. A Paged Access Method (PAM), responsible for mapping the logical pages in secondary memory.
2. Indexed Access Method (IAM), responsible for manipulating variable length records containing unique keys whose lengths are also variable.
3. Inverted Access Method (VAM), responsible for creating and manipulating inverted lists.

The environment aimed for were the 8 and 16 bit versions of the CP/M operating system and the language used was C. The first level, the Paged Access Method, manages the allocation of mew pages and the freeing of unwanted pages for their reutilization when necessary. The algorithm for allocating the areas in main memory is LRU, minimizing the number of secondary memory accesses. The second level is built over the first , implementing the primitives of indexed access. The storage organization chosen was the B+ trees, minimizing the number of accesses in the direct queries to one record and permitting an efficient sequential search for the records that are in ascending order. The variable length records and keys permit greater efficiency whith respect to space utilization, thus at the height of the tree. The third level, the Inverted Access Method , is built over the first two levels, and inplements the primitive operation defined over inverted lists stored in mass memory.

* Engenheiro de Sistemas (PUC/RJ, 1983) : Mestrando em Informática (PUC/RJ);
Professor Auxiliar do Departamento de Informática da PUC/RJ.
**Mestre em Informática (PUC/RJ).

Pontifícia Universidade Católica do Rio de Janeiro
Departamento de Informática - PUC/RJ
Rua Marquês de São Vicente 225
22245 - Gávea - Rio de Janeiro - RJ

I) INTRODUÇÃO

O trabalho surgiu da necessidade de criar um método de acesso que permitisse trabalhar com um grande volume de dados. Tratando-se de sistemas interativos, o tempo de resposta deveria ser baixo. Outra característica comum às nossas aplicações, era a necessidade de trabalhar com acesso direto e sequencial às informações. Depois de um levantamento bibliográfico (1, 2, 3, 4) e orientados pelo professor Daniel A. Menascé, optamos individualmente, por um método de acesso utilizando árvores B+.

Utilizou-se a estrutura conhecida como árvore B+ na implementação. O grande problema da árvore B, refere-se à pesquisa sequencial. Para resolver este problema, foi desenvolvida uma variação da árvore B, denominada pelos autores de B+ (5) ou B*, onde os níveis mais altos das árvores são apenas índices, enquanto que as folhas possuem as chaves e as informações associadas. As folhas constituem uma lista encadeada, permitindo um eficiente acesso sequencial.

Vários Sistemas de Gerência de Banco de Dados utilizam árvores B+ ou alguma variação da mesma (5, 6). Entre eles podemos citar:

- Sistema R (BD relacional p/maq. grande porte)
- INGRES (BD relacional p/ maq. médio porte)
- O IMS através do VSAM (método de acesso p/o BD hierárquico p/maq. de grande porte)
- dBASEII (BD relacional p/ microcomputador)
- MDBSIII (BD de redes p/ microcomputador)
- SBD/TS (BD de redes, brasileiro, para micros).

Discute-se no trabalho, cada um dos tres níveis da implementação, as opções tomadas, restrições e dificuldades encontradas ao longo do desenvolvimento, além das possíveis alterações e melhoramentos que poderiam ser realizados.

II) ARQUITETURA

O ambiente onde os sistemas seriam inicialmente utilizados consistiria de um microcomputador baseado no microprocessador 8086, construido no LESC (laboratório de Engenharia e Sistemas de Computação) da PUC/RJ. Este computador está configurado com 128K bytes de memória e dois acionadores de discos flexíveis de oito polegadas, densidade simples, face simples. O sistema operacional disponível é o CP/M-86 que oferece algumas primitivas de acesso a disco bastante limitadas. Também disponíveis para o CPM-86 encontran-se o ASM86, montador de linguagem de máquina e a linguagem de alto nível C (8) . Foi esta última a escolhida para a implementação efetiva dos sistemas de recuperação de informação, devido às suas interessantes características de transportabilidade, estruturação, modularidade e eficiência.

Inicialmente seria produzida então uma biblioteca de funções primitivas que estenderiam a linguagem C para que se pudesse manipular arquivos indexados (níveis 1 e 2) com o mínimo de dificuldades possível. Além disso, seria interessante que o método de acesso fosse

o mais independente possível do sistema operacional hospedeiro, fator fundamental determinante de sua transportabilidade.

As árvores B+ trabalham com unidades de acesso à memória secundária chamadas páginas.Uma página pode conter índices ou, quando localizada em uma folha, registros. A página corresponde a uma área de tamanho fixo, um segmento de arquivo.

Tendo em vista as características acima mencionadas do MAI, dividimo-lo em duas partes distintas, dois níveis de construção:

1. Método de Acesso Paginado, reponsável pela manipulação de páginas pertencentes a um arquivo, em memória principal ou em disco.
2. Método de Acesso Indexado propriamente dito, utilizando-se das primitivas oferecidas pelo Método de Acesso Paginado e de árvores B+.

Graficamente temos vários ambientes, em sucessivas camadas ou níveis, dispostos da seguinte maneira:

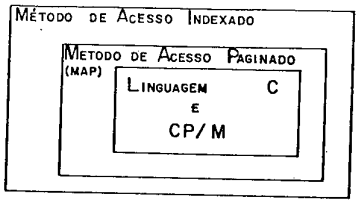


Fig.1

De forma a atender os requisitos do MAI, o Método de Acesso Paginado oferece as seguintes primitivas:

a) Criar um arquivo paginado;
b) Abrir um arquivo paginado;
c) Fechar um arquivo paginado;
d) Alocar uma página;
e) Liberar uma página;
f) Ler uma página;
g) Gravar uma página;
h) Fixar uma página; e
i) Desfixar uma página.

O Método de Acesso Indexado, por sua vez, oferece o conjunto de primitivas abaixo:

a) Criar um arquivo indexado:
b) Abrir um arquivo indexado;
c) Fechar um arquivo indexado:
d) Incluir um registro com uma chave única, inexistente;
e) Ler um registro dada uma chave já existente;

f) Alterar um registro;
g) Excluir um registro;
h) Posicionar o apontador de próximo registro em função de uma chave e de uma relação;
i) Ler o proximo registro na ordem;
j) Ler o registro anterior ao corrente;
k) Posicionar o apontador de próximo registro no início do arquivo;
l) Posicionar o apontador de próximo registro no fim do arquivo.

Uma vez implementado o Método de Acesso Indexado (MAI), constatamos uma restrição : as chaves são únicas. O gerenciamento de chaves duplicatas utilizando árvores B torna-se difícil devido à possibilidade de ocorrer uma quebra dividindo duas chaves idênticas, uma em cada página. Qual seria a chave separadora?

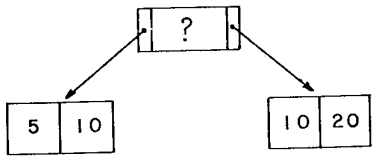


Fig.2

Assim sendo, ficamos impossibilitados de fazer pesquisas baseadas em chaves secundarias , cuja utilidade em sistemas aplicativos é incontestável (2, 3). Resolvemos então implementar um Método de Acesso que permitisse a manipulação de listas invertidas, de forma mais simples e abstrata.

As primitivas do Método de Acesso Invertido podem ser divididas em tres grupos:

1) Inicialização / Encerramento
   1.a) Abre um arquivo invertido;
   1.b) Cria um arquivo invertido;
   1.c) Fecha um arquivo invertido;

2) Manipulação de Listas
   2.a) Criar uma lista invertida;
   2.b) Abrir uma lista invertida;
   2.c) Abrir a próxima lista na ordem;
   2.d) Posicionar o apontador de próxima lista em função de uma chave e de uma relação;
   2.4) Excluir uma lista;

3) Manipulação dos Elementos de uma Lista
   3.a) Lê o elemento corrente de uma lista;
   3.b) Lê o próximo elemento de uma lista na ordem;
   3.c) Inclui um elemento na lista;

3.d) Exclui o elemento corrente de uma lista;

3.e) Posiciona o primeiro elemento como o corrente de uma lista;

Passamos então a ter mais uma camada de abstração, com a seguinte configuração:

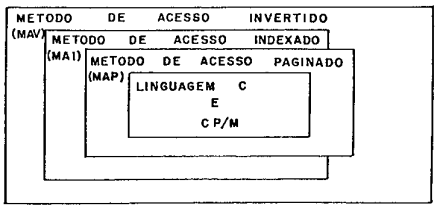


Fig.3

III) MÉTODO DE ACESSO PAGINADO

O Método de Acesso Paginado tem como objetivo gerenciar a manutenção de uma quantidade de páginas de um arquivo em memória principal. Naturalmente, quanto maior o número de páginas em memória, maior a velocidade de acesso média alcançada pelo sistema. Visamos exatamente reduzir o tempo médio de acesso que depende diretamente do número de referências à memória secundária (um recurso caro).

Na criação ou abertura de um arquivo paginado se recebe um ponteiro chamado identifica - dor de arquivo paginado, que deverá ser usado para referenciar todo o acesso seguinte ao arquivo. Além disso, é alocada uma área de trabalho para conter as páginas que ficarão em memória e a tabela que controla estas páginas. Cada página da tabela tem associados a "hora" em que ocorreu o último acesso a ela, e o arquivo a que pertence. Além disso, como a página pode ser fixada na memória quando a aplicação assim o exigir, torna-se necessário também manter uma informação dizendo se a página é removível ou não.

A informação da "hora" do último acesso é usada na implementação do mecanismo de seleção para remoção da página menos recentemente utilizada (LRU em inglês). Isto permite que se libere a página que foi acessada pela última vez a mais tempo entre as removíveis. A liberação da página se dará quando for necessário espaço em memória para outra página, lida ou criada (alocada). Existe um contador associado ao arquivo paginado que indica a "hora" atual do MAP e que é incrementado a cada acesso de leitura ou gravação.

As primitivas de fixação e desfixação de páginas na memória vem atender, basicamente, às necessidades de manipular várias páginas de memória ao mesmo tempo. Observa que uma página só estará garantidamente na memória desde após algum acesso a ela e até o próximo acesso à outra página qualquer, a menos que ela esteja explicitamente fixada.

Para contornar a incapacidade do CP/M de receber de volta as áreas em disco previamente alocadas porém não mais utilizadas, o MAP deverá também, gerenciar uma pilha com as páginas liberadas, de onde serão retiradas quando se tornar necessário, isto é, na alocação de páginas.

A Utilização do Método de Acesso Paginado se dá através de dez rotinas para o usuário e mais rotinas internas ao MAP. As primeiras nove rotinas correspondem exatamente às primitivas oferecidas e previstas no projeto do MAP, sendo que uma décima foi colocada posteriormente visando dar mais uma facilidade: ela retorna o número da página dado o seu endereço na memória. Além disso, existe uma função auxiliar que retorna o número de páginas possíveis de serem alocadas com o espaço de memória restante, e que permite a máxima utilização da memória disponível independente da configuração da máquina utilizada.

IV) MÉTODO DE ACESSO INDEXADO

O objetivo do Método de Acesso Indexado é fornecer um conjunto de operações primitivas ao sistema de aplicação para a manipulação de registros de tamanho variável. Descrevemos abaixo as características da nossa implementação.

O MAI é construido utilizando-se das primitivas do MAP. Sua única ligação com o meio físico é através do parâmetro que define o tamanho da página. Conseguiu-se desta forma, também, uma independência neste nível em relação ao Sistema Operacional.

Basicamente, existem dois tipos de páginas no mai: as páginas de índice e as páginas de folha. As primeiras são responsáveis pelo armazenamento dos índices que apontam em última análise para as páginas do tipo folha. Um arquivo vazio não possui páginas índice. As páginas folha são as que contêm os registros. Elas formam uma lista duplamente encadeada utilizada para a pesquisa sequencial.

Uma página do tipo índice tem a seguinte disposição interna:

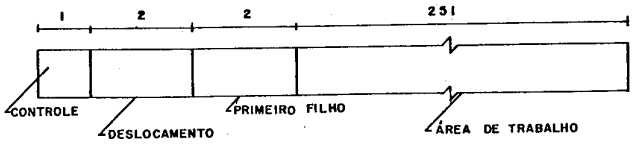


Fig.4

e uma página do tipo folha:

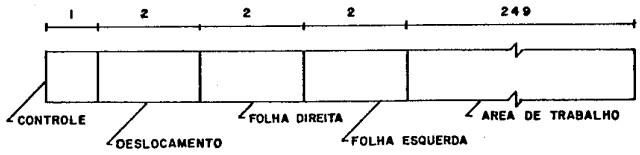


Fig. 5

As páginas podem, ainda, ser raiz ou não. Usa-se os 2 bits menos significativos do byte de controle da página para identificá-las. O bit 0 indica se é índice (1) ou folha (0) e o bit 1 se a página é raiz (1) ou não (0). Assim, temos 4 tipos de página: FOLHA (00), ÍNDICE (01), ÚNICA (10) e RAIZ (11). A página única é raiz e folha ao mesmo tempo. A criação de uma árvore gera uma página única, vazia.

Um registro de tamanho variável, como utilizado nesta implementação, tem no primeiro byte seu tamanho total. Imediatamente após este byte, segue-se o campo da chave de acesso,

e, em seguida vem os demais campos, se houver. Observe que esta escolha limita o tamanho total de registro a 256 bytes, o que é mais do que suficiente para as aplicações pretendidas. A alteração deste parâmetro, contudo, é trivial.

A chave de acesso do registro, por sua vez, também é um campo de tamanho variável. Assim sendo, o seu primeiro byte é utilizado para armazenar o tamanho total do campo e é seguido da chave propriamente dita. Sempre que falamos da chave de acesso como parâmetro de entrada ou no interior de um registro, estamos nos referenciando ao campo de tamanho variável que possui a chave real e o tamanho do campo.

Fig.6

Nesta implementação, escolheu-se por colocar os registros e os índices no mesmo arquivo para tornar a gerência das estruturas de dados e suas representações físicas mais simples. Por outro lado isto impacta na velocidade de acesso visto que a altura da árvore será em geral maior, pois mais folhas serão necessárias para armazenar os dados ao invés de armazenarmos só ponteiros em um arquivo e os dados em outro.

Nada impede, porém, utilizarmo-nos do método de acesso atual para realizar esta modificação. Basta acrescentar-se mais uma "casca" externa ao MAI, oferecendo-se assim estas mesmas operações com um único arquivo, para os índices, e outro para os dados.

Na criação e abertura, a raiz é fixada na memória com o objetivo de reduzir o número de acessor à memória secundária, visto que todo o acesso direto é feito através dela. Assim, se a árvore tiver altura quatro, por exemplo, serão necessários no máximo, tres acessos, no caso de nenhuma página estar presente na memória.

Sob o CP/M e com páginas de 2 registros físicos de 128 bytes, o número de páginas é de (2**15 - 1) o que implica em um total de 8 Mbytes. A altura da árvore está limitada, neste caso, portanto, a 16.

O procedimento de exclusão garante à árvore tornar-se no pior dos casos uma árvore binária balanceada. Se a árvore tiver altura H teremos no mínimo, portanto, 2** (H-1) registros podendo acessá-los em um máximo de (H-1) acessos. Na exclusão a concatenação só será efetuada quando a página estiver totalmente vazia. A sub árvore que ficar pendente é reincluida em uma página "irmã" (de mesmo nível). Conseguimos, com isso diminuir a complexidade da operação de exclusão, que prevê a redistribuição de índices ou registros nas baixas ocupações das páginas. Naturalmente, tal decisão implica em comprometimento com a utilização de espaço em memória secundária, mas por outro lado, diminui a frequência de alterações da árvore. No caso de nossas aplicações, onde a exclusão ocorrerá raramente, não se perderá muito.

Na inclusão a quebra será efetuada quando a página não puder suportar a inclusão, pois a página é de tamanho fixo. O conteúdo da página será dividido com uma nova página alocada, e a inserção então realizada na página conveniente. A divisão será feita da seguinte forma:

- na quebra de folhas, a chave separadora do nível acima (índice) será o prefixo mínimo (9) entre a última chave da página anterior e a primeira chave da página seguinte. O último caracter da chave separadora será ØXØØ se a última chave estiver contida na primeira, caso contrário será o sucessor do caracter pertencente à primeira chave que os difere;
- na quebra de índices um dos índices da página será escolhido para separar as duas páginas. Seu filho à direita será o primeiro filho à esquerda da nova página.

Existe um mecanismo que percebe que se está fazendo inclusões sequenciais, possibilitando fazer com que neste caso, a divisão dos registros não seja feita buscando obter níveis idênticos de ocupação, mas deixando a página cheia intacta e realizando a inclusão na nova página alocada. O efeito obtido é que arquivos criados pela inclusão sequencial e ordenada pelas chaves será mais compacto que se fosse criado através de inclusões aleatórias.

O emprego de técnicas de compactação para o armazenamento dos prefixos mínimos que formam os índices, poderia aumentar a densidade dos mesmos por página e, por conseguinte, diminuir a altura da árvore de acesso.

V) MÉTODO DE ACESSO INVERTIDO

O objetivo do Método de Acesso Invertido é de fornecer um conjunto operações primitivas , sob a forma de rotinas, ao sistema de aplicação para manipulação de listas invertidas em memória auxiliar, armazenadas em um objeto que chamaremos arquivo invertido. Um arquivo invertido consiste em uma organização onde ocorre a mudança de papéis entre o registro e o atributo (14). Existem dois tipos básicos de consultas que o usuário deseja:

1) Pesquisas realizadas em chaves primárias, quais propriedades/caracteristicas uma determinada entidade tem. Exemplo: Funcionários com número de identificação igual a 6. Funcionário é a chave primária.
2) Pesquisas baseadas em chaves secundárias, quais entidades possuem determinada característica. Exemplo: Todos os funcionários do Estado de São Paulo. Estado é a chave secundária.

A maneira mais eficiente de implementar-se um método de acesso baseado em chaves secundárias (5, 6) é através de um arquivo invertido, porque uma vez localizada a chave, os acessos subsequentes são diretos.

Um arquivo invertido lógico consiste em 2 arquivos CP/M. Um, o arquivo indexado, gerenciado pelo Método de Acesso Indexado, contém as chaves das listas e ponteiros para seus respectivos elementos. O outro, o arquivo paginado, gerenciado pelo do Método de Acesso Paginado, contém os elementos das listas.

As primitivas do Método de Acesso Invertido podem ser divididas em 3 grupos: Inicializa -

ção e Encerramento, Manipulação de listas e Manipulação dos Elementos de uma Lista. Para acessar uma lista (primitivas de manipulação de listas) basta ao usuário fornecer a cadeia, que é a chave da lista. Isto difere do Método de Acesso Indexado, onde os parâmetros passados são o registro montado ou a chave montada. O resultado consiste num ganho de simplicidade para o usuário, acarretando porém a restrição de a chave não poder conter o caracter 00H, que é o delimitador de uma cadeia em C.

Da mesma forma que no Método de Acesso Paginado (MAP) e no Método de Acesso Indexado (MAI) na criação ou abertura de um arquivo invertido se recebe um ponteiro, chamado identificador de arquivo (no caso do Método de Acesso Invertido, identificador de arquivo invertido), que deverá ser usado para referenciar todo o acesso subsequente ao arquivo.

Para acessar os elementos da lista (primitivas de manipulação de elementos da lista) o parâmetro passado é um registro do qual o primeiro byte contém o tamanho do elemento e o resto do registro contém o elemento da lista.

Consideramos que, para a chave de uma lista, a restrição de não conter 00H não traz grandes problemas, enquanto que a vantagem em simplicidade é grande. Quanto aos elementos da lista, a restrição de não poder ter ooH tornaria muitas aplicações impraticáveis..

VI) CONCLUSÕES

As operações primitivas especificadas e implementadas conseguem ser tanto simples quanto poderosas, permitindo sua utilização em diversas aplicações de vários graus de complexidade. Elas preveem mecanismos bastante adequados à manipulação de dados em memória secundária utilizando conceitos que se sobrepõem em níveis de abstração distintos.

O conjunto é modular, com rotinas totalmente parametrizadas e independentes uma da outra. Essa modularidade permite:

- uma boa documentação;
- modificação de uma rotina sem ter que alterar demais;
- o teste de cada rotina individualmente;
- a possibilidade de estender o sistema para outras aplicações;

Podemos definir um sistema de propósito geral (generalidade), como sendo um que pode ser utilizado para uma enorme variedade de aplicações. Este conjunto de primitivas satisfaz esta característica também. Praticamente qualquer sistema de informação pode ser implementado utilizando as primitivas do MAI e do MAV.

s rotinas foram escritas em C, o que nos garante uma boa confiabilidade contra falhas de software e também grande transportabilidade. Embora o projeto do sistema tenha sido "top-down", a implementação e os testes foram feitos "bottom-up".

Pretende-se que as rotinas sejam acessíveis por outras linguagens também, o que tornaria o MAI e o MAV bem mais interessantes comercialmente: BASIC e COBOL seriam os primeiros passos.

Uma das limitações do nosso sistema é a ausência de mecanismos contra falhas físicas (p.

ex., falta de energia). Poderíamos acrescentar técnicas de recuperação dinâmica de falhas. Porém, estas técnicas se mostram fracas diante das limitações do CP/M com respeito à integridade dos arquivos em tal situação.

Quanto à eficiência, sabemos que a linguagem C (pelo menos o nosso compilador) gera um bom código e o sistema de árvores B+ é sem dúvida um dos mais eficientes métodos de acesso existentes. Porém, o nosso sistema pode se tornar mais eficiente com algumas medidas como:

- introduzir técnicas de compactação de chaves para seu armazenamento nas páginas de índice.
- reprogramar em linguagem de montagem (assembly) os trechos do código que seriam os gargalos.

Finalmente, podemos dizer que o sistema é muito simples para o usuário ou o programador no nível de aplicação utilizar. Basta possuir a biblioteca contendo as rotinas primitivas do MAV, MAI, e MAP, e conhecer suas definições (10). Como exemplo de utilização do MAI podemos citar um sistema de informação desenvolvido na PUC para controlar os estoques de circuitos integrados do LESC (13). O MAV foi utilizado para a implementação de um Sistema de Pastas Eletrônicas no projeto de automação de escritórios em redes locais também desenvolvido na PU/RJ (11).

Concluindo ratificamos que o objetivo do trabalho foi plenamente alcançado e esperamos, que o sistema venha a ser de grande utilidade.

REFERÊNCIAS BIBLIOGRÁFICAS

1 - Date, C.J. "An Introduction to Data Base Systems". Addison-Wesley Publishing Co. Mass., 1977, Second Edition.

2 - Furtado A.L. e Santos, C.S. "Organização de Banco de Dados". Editora Campos Ltda. RJ, 1982.

3 - Melo, Rubens N. "Estruturas de Armazenamento de dados" CCE, Departamento de Informática PUC/RJ, 1981.

4 - Wedekind, H. "On the selection of acess paths in a data base system", Proc. IFIP Working Conference on Base Management, North Holland, 1974.

5 - Comer, D., "The ubiquitous B-Tree, ACM Computing Surveys, 11(2), 6/79.

6 - Kruglinski, David "Data Base Management Systems" Osborne, Mc.Graw-Hill, 1983.

7 - Stonebraker, Michael, "Restropection on a Data Base System", ACM Transactions on - Database Systems, 5(2), 6/80.

8 - Kernigham, B.W. e Ritchie, M., "The C programming Language", Prentice Hall, 1978.

9 - Bayer,R.,Unterauer,K.,"Prefix B-Trees"ACM Transactions on Data Base Systems, 2(1) 3/77.

10 - Vieira, Ewerton A.P. e Guimarães, Mario A.M., "MAV,MAI e MAP, Manual do Usuário, - V1.1", Depto de Informática PUC/RJ.

11 - Guimarães, Mario A.M., "Especificação e Implementação de um Sistema de pastas Eletrônicas" Dissertação de Mestrado, PUCRJ, 04/84.

12 - Vieira Ewerton A.P. e Guimarães, Mario A.M., "Implementação de um Método de Acesso Indexado com utilização de árvores B+", anais da X Conferência Latino Americana de Informática, Valparaiso, Chile, 4/1984.

13 - Vieira Ewerton A.P., "Sistemas de Controle de Estoque para Circuitos Integrados" , Trabalho de Final de Curso em Engenharia Elétrica/Sistemas, PUC/RJ, 1983.

14 - Menascé, D. e Landes O,, " On the design of a reliable storage component for dis - tributed database Management systems", Proc. Very Large Data Base Conference, - Montreal, 1980.

15 - Menascé, D. e Landes, O., "Dynamic crash recovery of balanced trees". IEEE Compu - ter Society Press, Symposium on reability in distributed software. Pittsburgh,7/81.