UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT

ESTUDO DE TÉCNICAS DE PAGINAÇÃO

ANTONINO MONGIOVI

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DO CENTRO DE CIÊNCIAS E TECNOLOGIA DA UNIVERSIDADE FEDERAL DA PARAÍBA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS - (M.Sc.).

APROVADO POR:

Prof. BRUNO C. DA NÓBREGA QUEIROZ
- Presidente -

COMISSÃO :

Prof. ORION DE OLIVEIRA SILVA

Prof. ADRIAN JAMES WEERHEIM

Prof. JAY C. MAJITHIA
- Orientador -

CAMPINA GRANDE

ESTADO DA PARAÍBA - BRASIL

- 1976

AOS MEUS PAIS

AGRADECIMENTOS

Ao professor JAY C. MAJITHIA, meu orientador, pelo empenho e dedicação com que me acompanhou durante a elaboração deste trabalho;

Ao professor MÁRIO T. HATTORI, pela revisão dos manuscritos desta Tese;

À CAPES pela ajuda financeira e à Universidade Federal da Paraíba pelas condições materiais, indispensáveis à realização desta obra;

Ao Centro de Prestação de Serviços Técnicos de Pernambuco - CETEPE, na pessoa do seu Presidente Dr. ROLDÃO GOMES TORRES, pelo apoio oferecido, e membros da Divisão de Apoio Técnico, pela colaboração e atenção dispensada na datilografia do texto.

R E S U M O

Este trabalho tem por objetivo a análise de técnicas de paginação, em sistema que utilizam memória virtual. Para o seu desenvolvimento, à apresentação dos conceitos de página , e de sua utilização fez-se necessária.

Para atingir a finalidade deste trabalho, as técni - cas consideradas foram discutidas de início, individualmente, quanto a sua filosofia de substituição das páginas, bem como da sua implementação, e em seguida uma comparação entre as mesmas foi realizada, baseada em modelos matemáticos, onde os resultados obtidos foram os mesmos alcançados empiricamente por vários pesquisadores.

Conclusões a respeito dos resultados, teóricos e práticos, são apresentadas.

Uma simulação destas técnicas com o intuito de se testarem aqueles resultados é também sugerida.

ABSTRACT

The subject of this paper is an analysis of paging techniques used in systems which operate in virtual memory mode. An explanation of the concepts of "page" and "paging" was necessary for the introduction of the subject.

In order to attain the desired objectives, each of the paging techniques considered is discussed, with regard to its philosophy of page replacement and its implementation, and then a comparative study is performed, on basis of mathematical models, which is shown to yield the same results as those obtained empirically by several researchers.

Conclusions about the theoretical and practical results are presented, along with suggestions on further research on the subject, by way of simulation of the paging techniques discussed in this paper.

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO E ESTRUTURA DA TESE

O objetivo principal desta tese é fazer um estudo de técnicas de paginação analisando os resultados obtidos por vários pesquisadores. A análise tem por finalidade obter a técnica que apresentou melhor desempenho.

1.1 Importância do Problema

A importância deste trabalho deve-se ao fato de vários pesquisadores terem desenvolvidos diversos Algoritmos para utilização de memória virtual, tais como: Algoritmo FIFO; Algoritmo LRU; Algoritmo Working Set(DENNING) [8]; Algoritmo PFF(CHU-OPDERBECK) [5]; e o Algoritmo CPFR(SADEH)[21] no intuito de melhorar a utilização da memória principal, sem contudo fazer uma comparação entre as mesmas.

1.2 Estrutura Geral da Tese

Esta tese organizacionalmente está dividida em 5 capítulos. A sua estrutura será melhor entendida através do resumo feito dos capítulos que a constituem.

1.2.1 CAPÍTULO 2: ORGANIZAÇÃO DA MEMÓRIA

Capítulo que faz uma apresentação geral dos esquemas existentes para gerenciamento de memória. Visando-se uma melhor sequência na apresentação e compreensão do mesmo, os esquemas são mostrados conforme seu grau de complexidade. Iniciando-se pelo gerenciamento de memória sequencial, que é o mais simples, até apresentar-se o gerenciamento de memória com demanda de página, empregado na utilização de memória virtual.

1.2.2 CAPÍTULO 3: CONCEITO DE PÁGINA E TÉCNICAS DE PAGINAÇÃO

Para utilizar-se memória virtual, existe a necessidade de dividir-se os programas em várias partes a fim de serem processados. Neste capítulo, uma definição destas partes de programas foi feita as quais foram chamadas de páginas, também mostra-se por que deve-se usar a paginação dando-se inclusive o histórico do seu aparecimento, ainda apresenta-se através de uma função, como um sistema paginado opera, e por fim a descrição das técnicas de paginação a serem estudadas neste trabalho é feita, classificando-as em duas categorias, técnicas de partição fixa e técnicas de participação variável.

1.2.3 CAPÍTULO 4: COMPARAÇÃO ENTRE AS VÁRIAS TÉCNICAS DE PAGINAÇÃO EXISTENTES

Após a apresentação das técnicas, a serem analisadas, um estudo comparativo entre as mesmas é feito neste capítulo . Este estudo baseia-se no "Custo de Memória", que define qual a técnica que ocupará menor espaço de memória em determinado intervalo de tempo, e na "eficiência" das mesmas, que são parâmetros do fator econômico.

Ainda, a análise dos resultados, obtidos quando da comparação destas técnicas é realizada e observações, com respeito a variação do tempo dentro de cada técnica, são feitas.

1.2.4 CAPÍTULO 5: CONCLUSÕES E PESQUISAS FUTURAS

Um resumo do 4º capítulo, as conclusões obtidas e um algoritmo ideal para substituição de páginas são aqui apresentados. Como complementação deste trabalho uma simulação destas técnicas é sugerida.

CAPÍTULO 2

ORGANIZAÇÃO DA MEMÓRIA

O objetivo principal dos módulos de um sistema operacional que gerenciam a memória de um sistema de computador é organizar a memória principal. Especificamente este gerenciamento terá dentre outras funções, as seguintes:

1 - Manter registrado o estado de cada posição da memória, isto é, se cada posição está alocada ou livre.

2 - Determinar um plano para a distribuição da memória, decidindo quem a ocupará, com que tamanho, quando e onde será alocado.

3 - Possuir técnicas de alocação (Allocation), uma vez decidido ocupar-se a memória, posições específicas devem ser selecionadas e as informações de alocação atualizadas.

4 - Possuir técnicas de liberação (Deallocation), um processo pode explicitamente liberar a memória previamente alocada, ou requisitar a memória baseado num esquema de desatribuição.

O esquema escolhido para o gerenciamento da memória será uma decorrência natural do desejo de possuir-se em um sis

tema operacional, módulos pequenos e simples para o gerencia - mento da memória, maior flexibilidade do usuário quanto ao uso do mesmo e melhor eficiência do sistema. Neste capítulo daremos a descrição de alguns esquemas e técnicas existentes. Algumas destas técnicas darão maior utilização de memória, outras fornecerão ao usuário maior flexibilidade, cuja implicação destas facilidades é tornar os sistemas mais complexos, aumentar o custo de Hardware, bem como a improdutividade (Overhead). Os esquemas a serem descritos são:

1 - Gerenciamento de Memória Sequencial

2 - Gerenciamento de Memória Particionada

3 - Gerenciamento de Memória Particionada com Relocação

4 - Gerenciamento de Memória Paginada

5 - Gerenciamento de Memória com Demanda de Página

Para cada técnica e esquema de gerenciamento de memória a ser estudada, faremos a sua análise dividida em três seções:

1 - Daremos uma visão geral do método e conceito empregados.

2 - Daremos uma descrição do algoritmo de Software particular e o processamento requerido.

3 - Apresentaremos uma discussão das vantagens e desvantagens para cada técnica em particular.

2.1 Alocação Sequencial

A alocação sequencial é um esquema simples de gerenciamento de memória que não requer Hardware especial. Este esquema é usualmente associado a pequenos computadores padrões cujos simples sistemas operacionais operam em Batch. Utilizando este esquema temos os computadores IBM OS/360 Primary Control Program, IBM 1130 Disk Monitor System, IBM 7094 Fortran Monitor System. Em tais sistemas não existe multiprogramação e uma correspondência unívoca é feita entre um usuário, um Job, um Job Step e um processo. A memória é atribuída para um Job como mostrado na Figura 2.1.

A memória é conceitualmente dividida em três regiões contíguas, sendo que, uma partição da memória é permanentemente alocada pelo sistema operacional. O restante da memória fica reservado para um único JOB a ser processado. Os JOB'S em geral não ocupam toda a memória, consequentemente teremos uma terceira parte não utilizada.

2.1.1 Software de Apoio

A Fig. 2.2 mostra o fluxograma para a alocação unicamente sequencial. Este algoritmo é utilizado quando a rotina - roteiro (JOB SCHEDULER) deseja organizar a memória para o processamento de um JOB, desde que não existe nenhum no momento

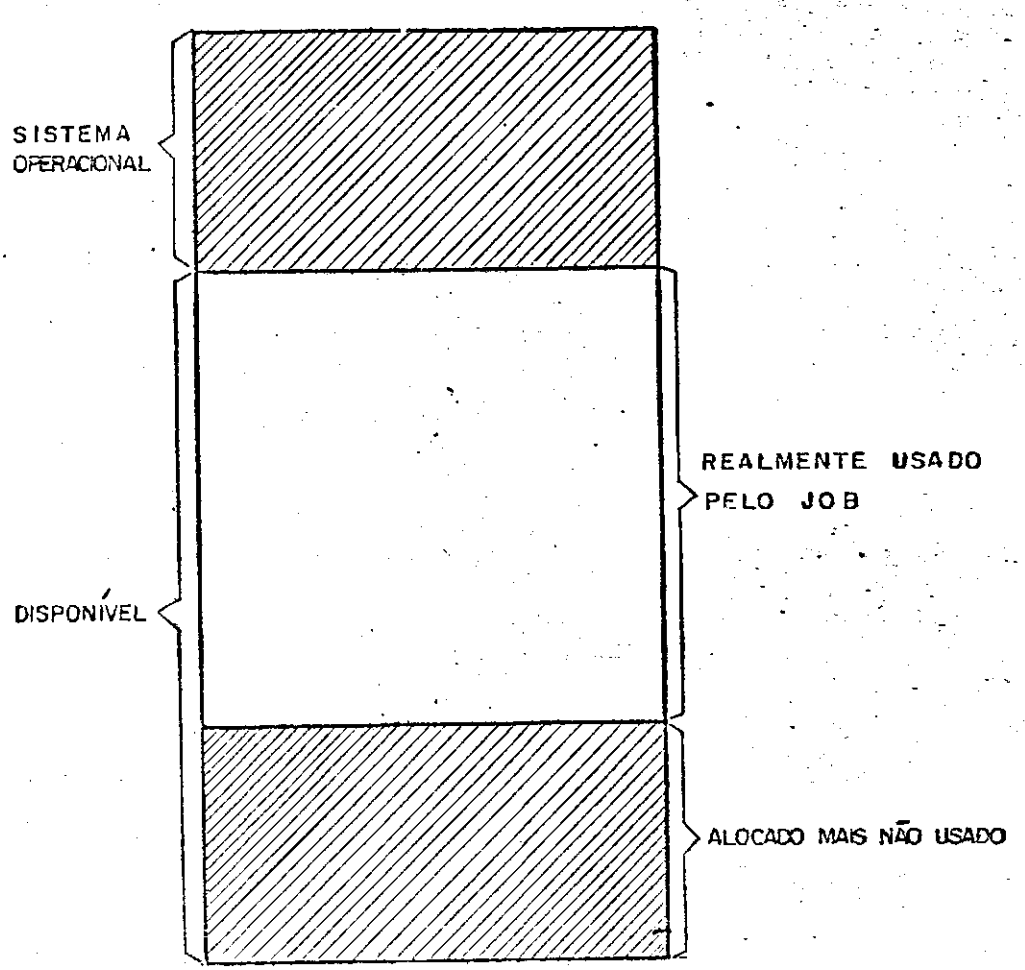

FIG. 2–1

ALOCAÇÃO SEQUENCIAL

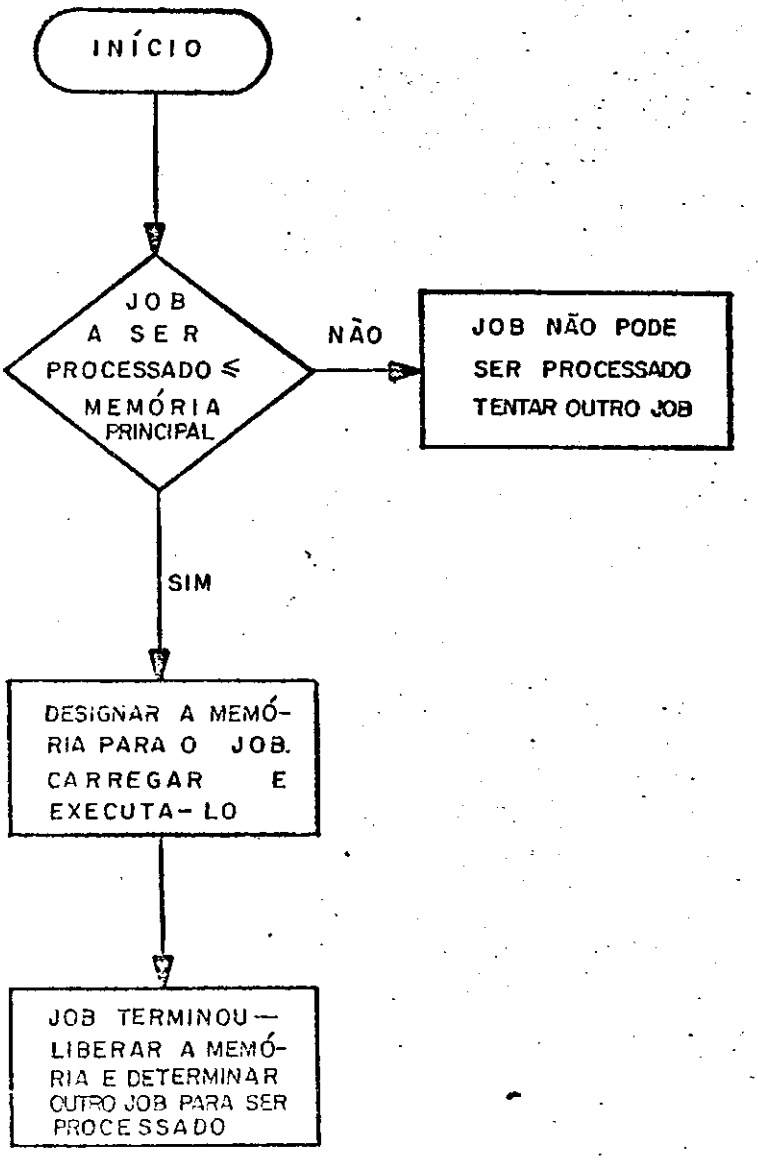

FIG. 2-2

A L O C A Ç Ã O S E Q U E N C I A L

sendo processado.

2.1.2 Vantagens

A grande vantagem deste esquema consiste na sua simplicidade. Uma outra vantagem é que não se necessita grandes conhecimentos para entender ou utilizar o sistema operacional.

2.1.3 Desvantagens

A grande desvantagem deste esquema está relacionado ao fato que a memória não é totalmente utilizada. Uma outra desvantagem é a má utilização dos processadores, devido a espera de uma operação de entrada/saída (I/O), além da limitação no tamanho dos programas a serem processados.

2.2 Alocação de Memória Particionada

A alocação Particionada, é uma das técnicas mais simples para gerenciamento de memória quando se utiliza multiprogramação. A memória é dividida em várias partes ou partições, onde cada partição contém um JOB, ver Fig. 2-3 .

2.2.1 Algoritmo do Software

Existem várias maneiras de se utilizar a alocação par

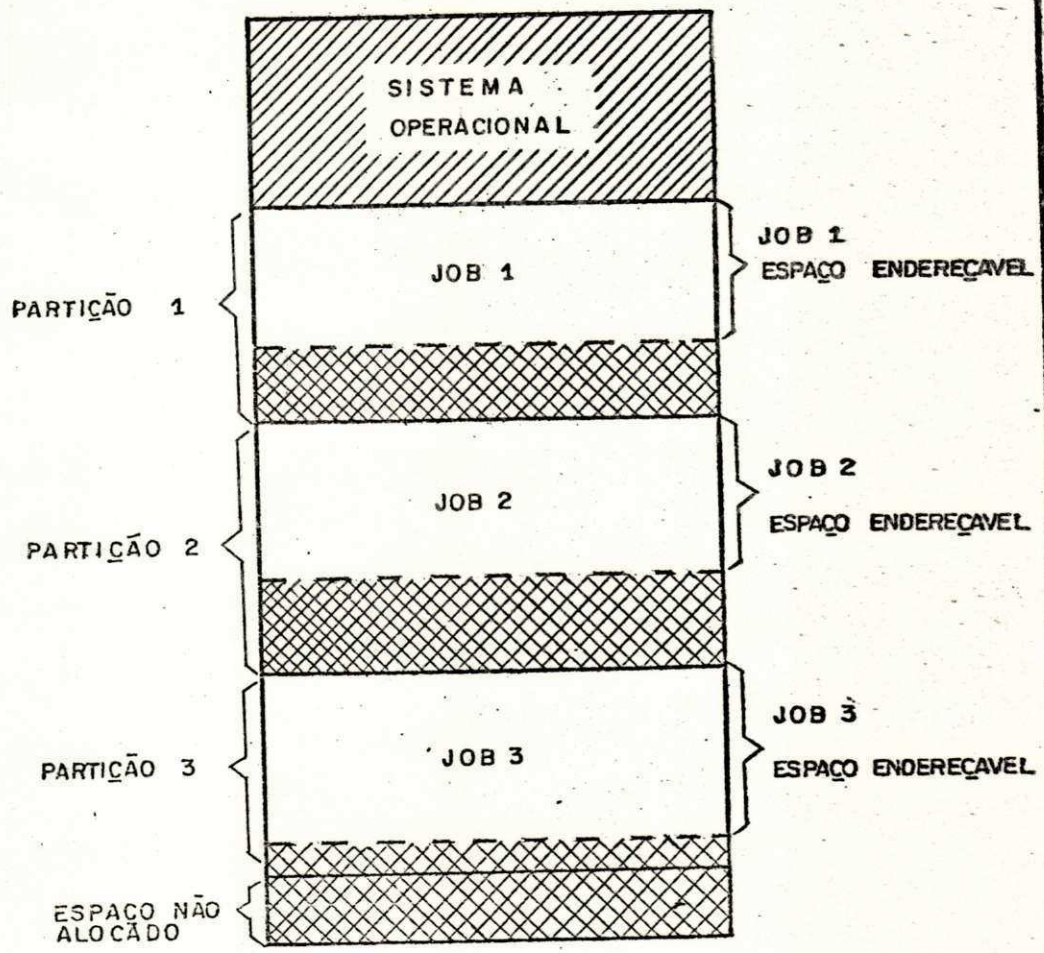

FIG. 2-3

ALOCAÇÃO DE PARTIÇÃO

ticionada. Apresentaremos aqui, as duas mais comumente usadas que são: UTILIZAÇÃO DE PARTIÇÃO ESTÁTICA (Static Partition Specification) e UTILIZAÇÃO DE PARTIÇÃO DINÂMICA (Dynamic Partition Specification).

a) Utilização de Partição Estática

Será denominada utilização estática, a divisão feita à priori na memória para o processamento de qualquer JOB. Esta técnica é semelhante a usada nos sistemas IBM'S OS/360 MFT (Multiprogramming with a fixed number of Tasks). As partições podem ser criadas tanto pelo operador do computador como podem ser construídas dentro do próprio sistema operacional. Um exemplo de uma tabela de especificações de partição estática é dado na Fig. 2.4.

Na utilização desta técnica, cada programa do usuário especifica a quantidade de memória que o mesmo necessitará, logo, a partição estática é apropriada para processar programas de tamanhos conhecidos. O sistema operacional se encarrega de alocar cada programa na partição disponível cujo tamanho for apropriado para o mesmo. O algoritmo para gerenciar a alocação e a liberação das partições é um tanto simples.

b) Utilização de Partição Dinâmica

NÚMERO DA PARTIÇÃO	TAMANHO	POSIÇÃO	ESTADO
1	8K	312 K	OCUPADA
2	32K	320 K	OCUPADA
3	32K	352 K	LIVRE
4	120K	384 K	LIVRE
5	520K	504 K	OCUPADA

FIG. 2-4

TABELA DE UTILIZAÇÃO DE PARTIÇÃO ESTÁTICA

NÚMERO DA PARTIÇÃO	TAMANHO	POSIÇÃO	ESTADO
1	8K	312 K	ALOCADO
2	32K	320K	ALOCADO
3	—	—	ENTRADA VAGA
4	120K	384K	ALOCADO
5	—	—	ENTRADA VAGA
	---	---	

TABELA DE ESTADO DAS PARTIÇÕES ALOCADAS

ÁREA LIVRE	TAMANHO	POSIÇÃO	ESTADO
1	32K	352 K	DISPONÍVEL
2	520K	504 K	DISPONÍVEL
3	—	—	ENTRADA VAGA
4	—	—	ENTRADA VAGA
	---	---	---

TABELA DE ESTADO DAS ÁREAS LIVRES

FIG. 2-5

TABELAS PARA UTILIZAÇÃO DE PARTIÇÃO DINÂMICA

Será denominada utilização dinâmica, quando as partições são criadas à medida que os JOB'S são processados e cujo tamanho das partições será correspondente a cada JOB.

A Fig. 2-5 mostra o esquema de tabelas para a utilização de partição dinâmica. Duas tabelas são usadas, uma para registrar as áreas alocadas e a outra para indicar as áreas disponíveis. A Fig. 2-6 mostra um exemplo de utilização de partição dinâmica. Na Fig. 2-6a vemos três partições alocadas na memória, cada uma contendo um JOB de tamanho correspondente. Se três JOB'S adicionais devem ser multiprogramados, então novas partições de tamanho correspondente a cada JOB devem ser criadas nas áreas livres restantes, Fig. 2-6b. Após os JOB'S serem processados, as partições serão liberadas, a Fig. 2-6c mostra o estado da memória após dois JOB'S terem sido processados. Para escolher-se qual a área livre da memória que será ocupada por um JOB, existem dois esquemas a se adotar. Uma maneira seria alocar-se o programa no primeiro espaço de memória livre que couber o mesmo e um segundo esquema seria o de alocar o programa no espaço de memória livre que melhor o encaixe, isto é, no menor espaço possível que caiba o programa, para utilização destes esquemas, dois algoritmos existem os quais serão descritos a seguir:

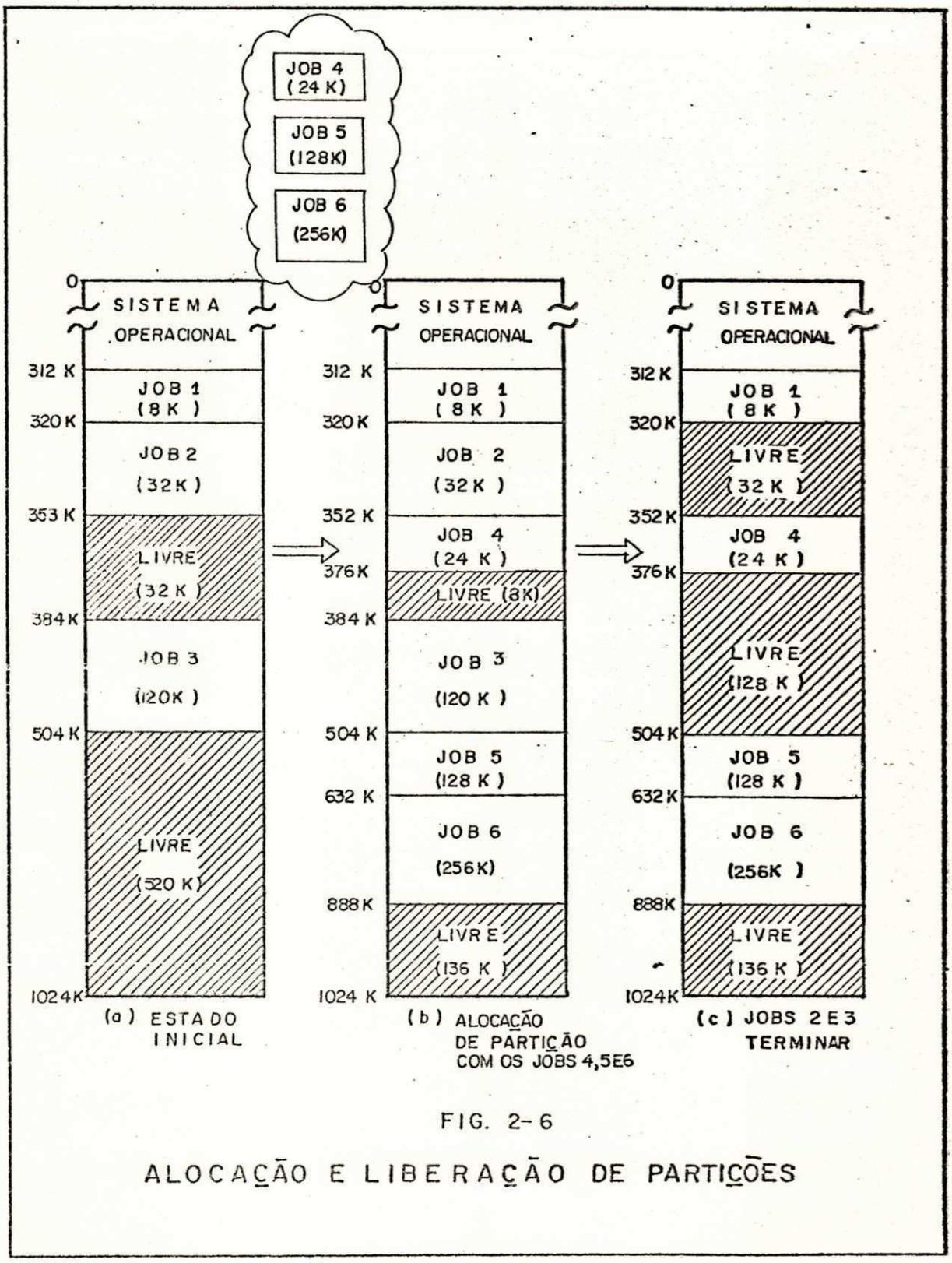

FIG. 2-6

ALOCAÇÃO E LIBERAÇÃO DE PARTIÇÕES

b.1 Algoritmo primeiro a se adaptar (First Fit Algorithm)

Neste algoritmo a tabela das áreas disponíveis é mantida classificada em ordem crescente pelas posições de memória. Quando uma partição deve ser alocada, o algoritmo inicia a pesquisa na tabela pelo endereço mais baixo e vai sempre comparando o tamanho da partição com o tamanho da área disponível, até encontrar a primeira área disponível que caiba a partição a ser alocada.

b.2 Algoritmo melhor a se adaptar (Best Fit Algorithm)

A diferença básica entre este algoritmo e o First Fit é que enquanto o First Fit classifica a tabela das áreas disponíveis pela posição de memória, a classificação das áreas disponíveis do Best Fit é feita pelo tamanho da área, isto é,o primeiro bloco da tabela será a área disponível de menor tamanho. Assim a primeira área que se encontrar cujo tamanho, se adapte a partição que se deseja alocar, será de melhor encaixe.

c) Problema da Fragmentação

Vários fatores devem ser considerados na escolha do algoritmo para utilização de memória particionada, dentre eles teríamos, rapidez de alocação e simplicidade quanto ao seu uso. Porém o fator mais importante a considerar é o problema da

fragmentação, que consiste em se possuir um grande número de áreas disponíveis separadas, isto é, a porção de memória disponível é fragmentada em pequenas partes. A fragmentação pode ser minimizada através de uma escolha adequada quanto ao algoritmo a ser utilizado, podendo ser o algoritmo para utilização de partição dinâmica, o First Fit ou o Best Fit. Para utilização de partição dinâmica este problema pode ser superado utilizando-se outras técnicas, tais como relocação e paginação que serão estudadas nas seções seguintes.

2.2.2 Vantagens

As principais vantagens ao se particionar a memória são:

a) Implementação da multiprogramação, acarretando em utilizar-se mais eficientemente os processadores e os dispositivos de entrada e saída.

b) O Hardware não terá custos extras.

c) Os algoritmos usados são simples e de fácil implementação

2.2.3 Desvantagens

Na utilização de memória particionada existem várias

desvantagens que reduzem a utilização da memória.

a) A fragmentação, que é um problema muito significativo no particionamento da memória.

b) As áreas disponíveis podem não ter tamanho suficiente para alocar a partição.

c) Este esquema requer memória maior do que na alocação sequencial, devido a multiprogramação, bem como sistema operacional mais complexo.

d) Da mesma maneira que na alocação sequencial, a memória poderá conter informações que nunca serão utilizadas.

2.3 Gerenciamento de Memória Particionada com Relocação

Uma solução para o problema de fragmentação seria, compactar periodicamente todas as áreas disponíveis, em uma única área contígua. Isto poderá ser feito, deslocando-se o conteúdo de todas as partições alocadas de forma que se tornem contíguas, como mostra a Fig. 2-7. Este processo é chamado compactação (ou recompactação já que podem ser feito várias vezes). Apesar da relocação conceitualmente ser bastante simples, ao se deslocar um JOB nada garantirá que o JOB será processado corretamente na nova posição. Isto devido alguns itens chaves nos programas, tais como: (1) Registradores Base, (2) Instruções que referenciam a memória, (3) Estrutura de Informação (Ex. Fi

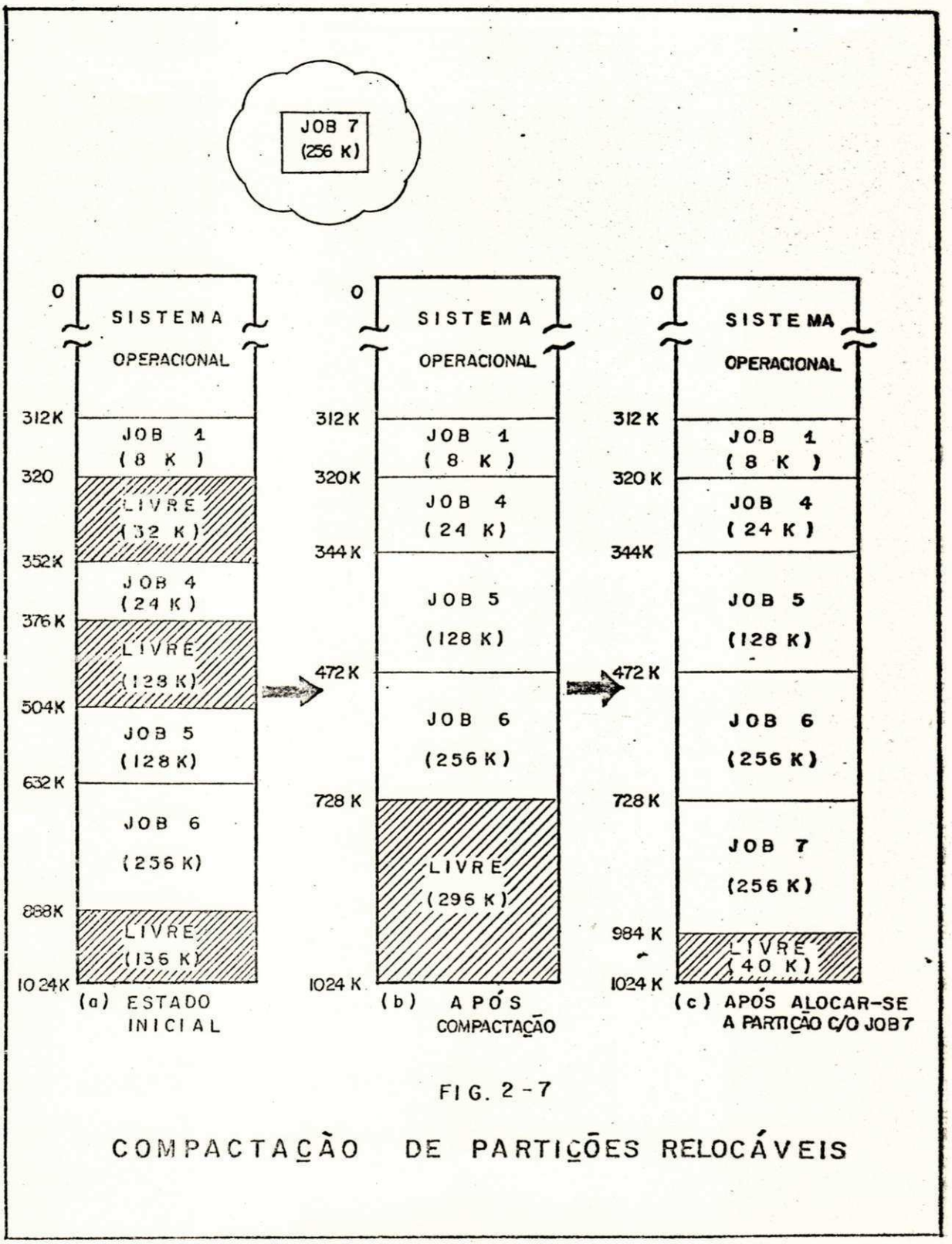

FIG. 2-7

COMPACTAÇÃO DE PARTIÇÕES RELOCÁVEIS

la, Pilha, etc) que utilizam apontadores de endereços.

Uma sólução para este problema será, deslocar-se todos os endereços dentro do JOB, a ser relocado, do mesmo valor durante o processamento. Esta relocação em alguns casos poderá ser complexa. Uma das razões seria como o Sistema Operacional determinará os apontadores de endereços que devem ser alterados.

Uma outra solução existente para o problema da relocação, será a de relocar-se todo o JOB e processá-lo novamente do início. Nesta alternativa, além de perder-se algum tempo de máquina, devido a repetição de alguns trechos de programas, este reinício poderá não ser viável, em casos onde o programa já tenha realizado algumas ações irreversíveis. Várias técnicas de Hardware foram desenvolvidas visando o problema da relocação.

2.3.1 Algoritmo do Software

Já que o início de cada espaço endereçável de um JOB (Job Address Space) não está relacionado com a posição física da partição na memória será conveniente considerá-lo na posição 0 (zero). Com isto cada JOB atuará como se estivesse carregado na posição 0 (zero) da memória. Estas considerações são feitas para facilitar os Hardwares de proteção para as partições alocadas.

O algoritmo para gerenciamento das partições básicamente é o mesmo que o descrito na seção 2.2 apenas a condição, "quando a compactação for realizada", será adicionada a este algoritmo. Duas alternativas existem para a compactação, a primeira é compactar-se às áreas disponíveis, logo após a partição ser liberada de maneira que a área disponível sempre estará contígua e não haverá fragmentação.

Uma segunda alternativa será compactar a memória sempre que uma área livre relativamente grande estiver disponível. A compactação ocorrerá com menor frequência do que o esquema anterior, porém a tabela que manterá as áreas livres será mais complexa. A Fig. 2-8 mostra o fluxograma para este algoritmo.

2.3.2 Vantagens

O esquema de partição relocável elimina a fragmentação tornando assim possível a alocação de mais partições, isto permite um alto grau de multiprogramação resultando no aumento de utilização de memória e dos processadores.

2.3.3 Desvantagens

Várias são as desvantagens a serem consideradas neste esquema:

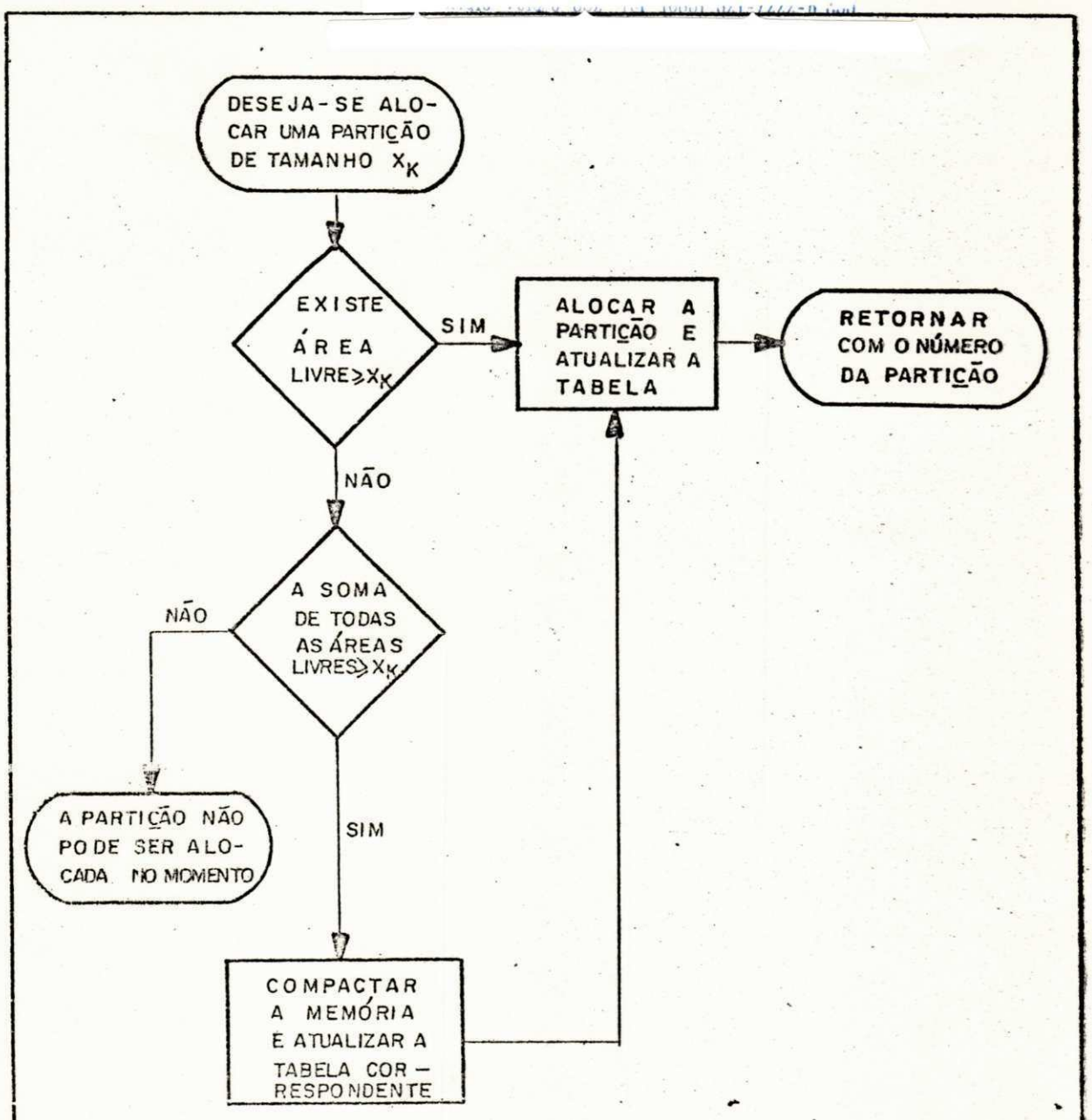

FIG. 2–8

ALOCAÇÃO DE PARTIÇÃO RELOCÁVEL

a) A relocação aumenta o custo de Hardware dos computadores.

b) O tempo gasto para a compactação das áreas livres poderá ser considerável.

c) Apesar da compactação, parte da memória poderá não ser usada, devido a quantidade de área disponível ser menor do que a partição a ser alocada.

d) Como a partição contém todas as instruções de um programa, poderá ocorrer que algumas nunca serão executadas.

2.4 Gerenciamento de Memória Paginada

O esquema anteriormente descrito, alocação de partições com relocação é uma das soluções para o problema da fragmentação. Esta relocação é feita utilizando-se dispositivos copiadores de endereços os quais fazem a junção de varias áreas livres da memória em uma única área contígua. Embora o problema de se ter várias áreas livres tenha sido resolvido através deste esquema, nem sempre a fragmentação é reduzida. Uma outra solução para a fragmentação e que não necessita-se tornar contíguas as áreas livres, é o esquema intitulado de Gerenciamento de Memória Paginada. Para o Gerenciamento de Memória Paginada, cada espaço endereçável(Address Space) ou JOB é dividido em partes iguais os quais são chamados páginas e igualmente a memória física é dividida em partes de mesmo tamanho chamados

blocos, sendo que qualquer página poderá ser alocada em qual - quer bloco disponível. Apesar dos blocos que contêm as pági - nas fisicamente não estarem contíguos, logicamente as páginas encontram-se contíguas, isto é, para o usuário, tudo se passa como se o programa estivesse alocado numa única área contígua. A cópia dos espaços endereçáveis para a memória física, realiza-se utilizando-se registros individuais para cada página, os quais são chamados de tabela copiadora de páginas, eles podem ser partes reservadas da memória ou podem ser Hardwares espe - ciais.

Visto que cada página pode ser alocada em qualquer bloco da memória e que as páginas na memória encontram-se lógi camente contíguas, não existe necessidades de se tentar alocar todas as partições de um JOB numa única área contígua. Observação deve ser feita com respeito a escolha do tamanho das páginas, pois desta escolha é que depende a eficiência do sistema.

Um exemplo simples é mostrado na Fig. 2.9 utilizan - do-se para o sistema páginas de 1.000 bytes, o JOB 2 possui es paço endereçável de 3.000 bytes, o qual foi dividido em três páginas. A tabela de páginas associada ao JOB 2 indica a posição de suas páginas. Neste caso, a página 0(Zero) está no bloco 2, página 1 no bloco 4 e a página 2 no bloco 7. A instrução LOAD 1,2108 do espaço endereçável do JOB 2 na posição 0518 (Pá

gina 0 Byte 518) na realidade ocupará a posição física 2158 (Bloco 2 Byte 0518). Igualmente, o Dado 015571 logicamente na posição 2108 será alocado no endereço de memória 7108. Este esquema é usado em várias computadores contemporâneos tais como, o XDS 940, XDS SIGMA 7, CDC 3300.

2.4.1 Algoritmo do Software

Existem três tipos de tabelas que devem ser gerenciadas pelo Software do sistema operacional: Tabela de JOB'S(TJ), Tabela de Blocos da Memória (TBM) e a Tabela Copiadora de Páginas (TCP). Cada JOB possui entrada individual na Tabela de JOB, indicando o endereço e o comprimento da sua "TCP", bem como outras informações. A tabela de blocos da memória indica as condições de cada bloco na memória isto é, alocado ou disponível. A Fig.2.10 mostra as tabelas que seriam utilizadas para o exemplo do parágrafo anterior referente a Fig. 2.9.

Um algoritmo simples para a alocação dos espaços endereçáveis é mostrado na Fig. 2.11. O único passo não explicitado neste algoritmo é, alocar tabelas copiadoras de páginas com n entradas. Uma solução é reservar determinado espaço da área do sistema operacional para as tabelas copiadoras. Estas tabelas são alocadas individualmente nesta área usando técnicas semelhantes ao esquema de memória particionada onde cada

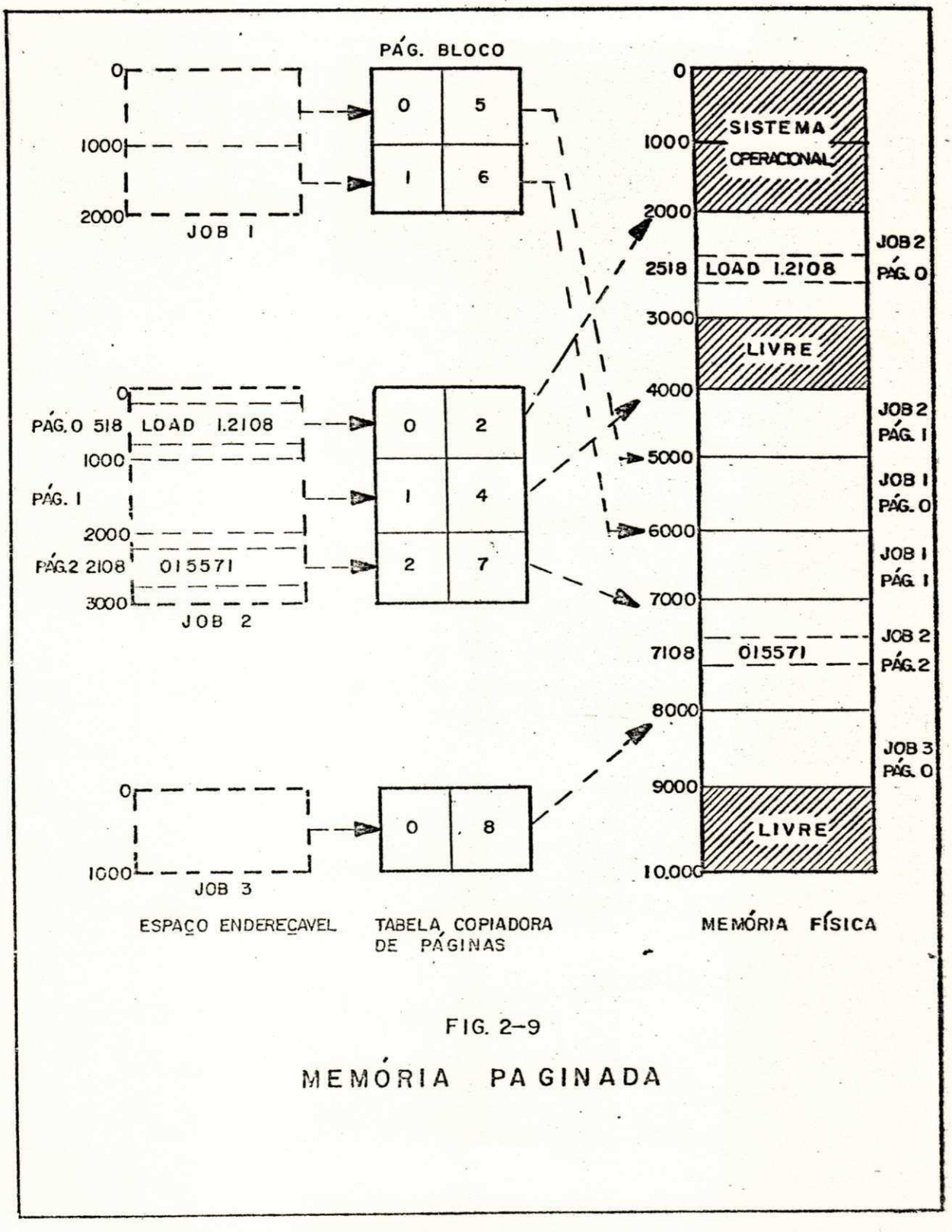

FIG. 2-9

MEMÓRIA PAGINADA

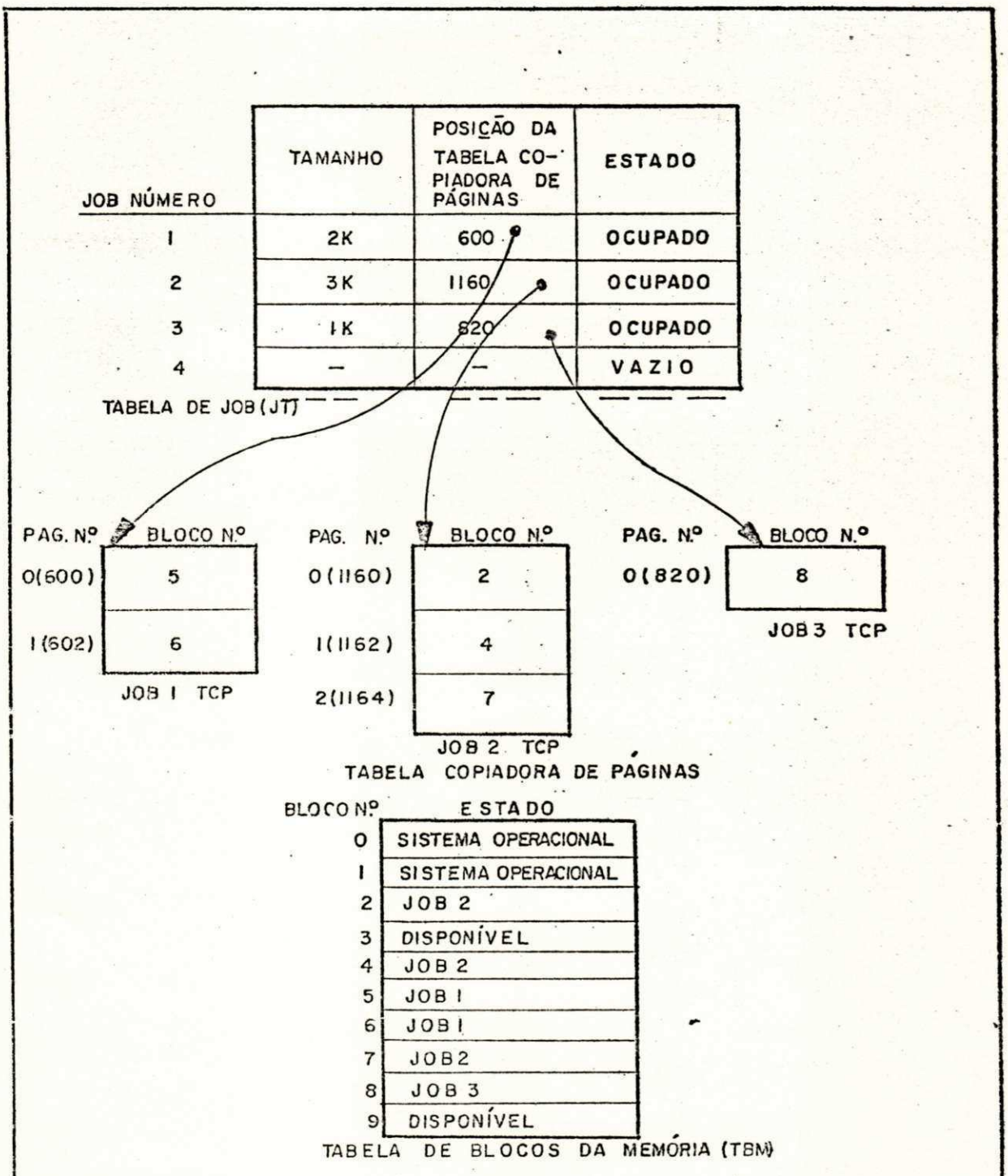

FIG. 2-10

TABELAS USADAS PARA ALOCAÇÃO DE MEMÓRIA PAGINADA

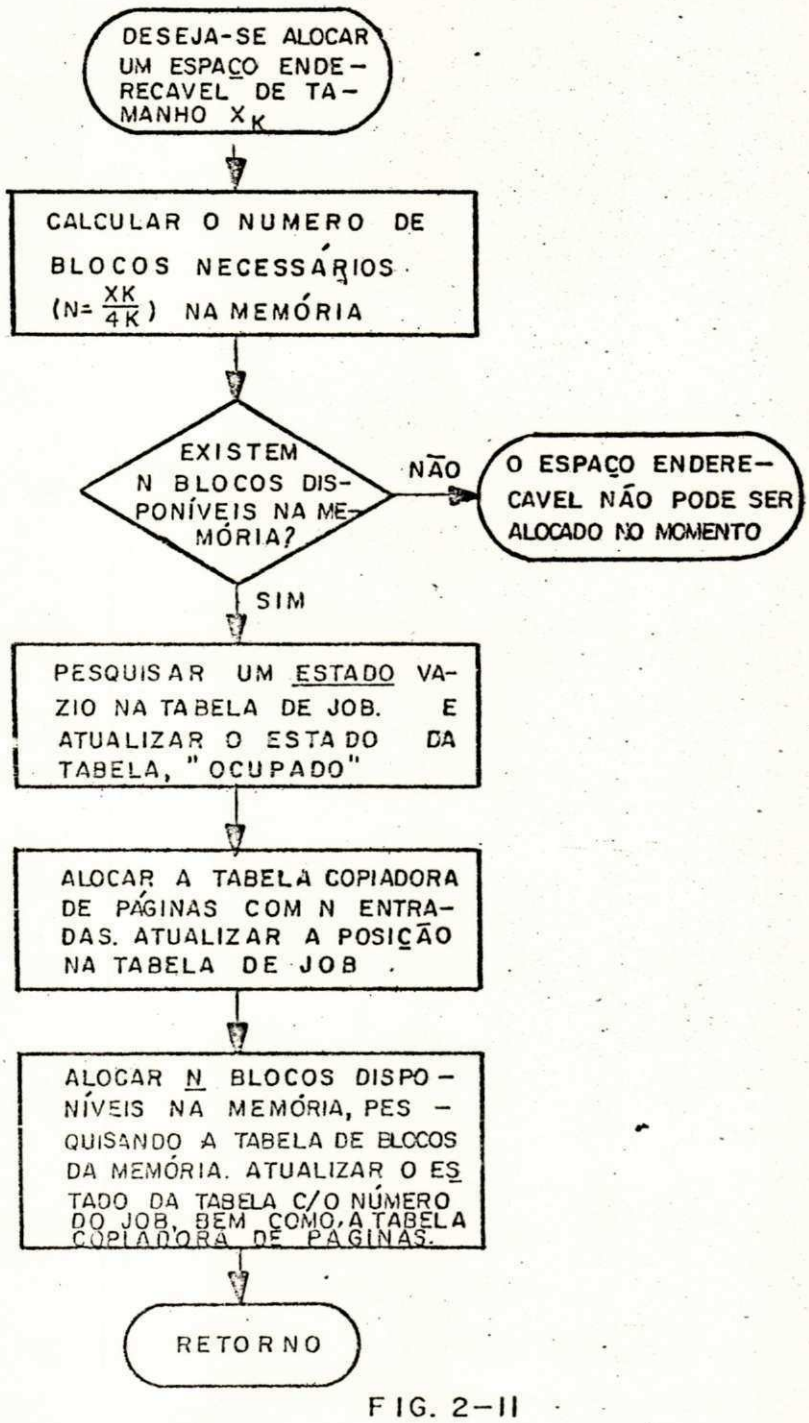

FIG. 2-11

ALGORÍTMO PARA ALOCAÇÃO DE MEMÓRIA PAGINADA

tabela é tratada como uma partição. Quando um Job é processado, sua partição correspondente a tabela criada é liberada.

2.4.2 Vantagens

O esquema de memória paginada elimina a fragmentação, o que torna possível alocar maior quantidade de JOB'S simultaneamente. Isto permite um alto grau de multiprogramação, o qual resulta num aumento de utilização da memória e dos processadores. Também elimina o problema de relocação de partições.

2.4.3 Desvantagens

Várias desvantagens devem ser consideradas:

a) O Hardware para copiar as páginas aumenta o custo do computador.

b) Parte da memória deve ser usada para guardar várias tabelas, principalmente as tabelas copiadoras.

c) Apesar da fragmentação ser eliminada, um fenômeno análogo chamado, fragmentação interna ou quebra de página, ocorre.

d) Parte da memória não será utilizada se o número de blocos disponíveis não forem suficientes para alocar o espaço endereçável do Job a ser processado.

2.5 Gerenciamento de Memória com Demanda de Página

Em todos os esquemas anteriores um JOB para ser processado teria que ser totalmente carregado na memória, exigindo que uma mesma quantidade de memória esteja disponível. Devido a esta condição vários JOB'S deixavam de ser processados apesar de existir áreas livres na memória, porém, menor que qualquer Job a ser processado. Uma solução para este problema seria construir-se computadores com memórias extremamente grandes. Solução esta que apesar de simples não é economicamente viável. Uma outra solução é utilizar-se sistemas operacionais os quais dão a "ilusão" de se possuir computadores com memória extremamente grande. Já que o grande tamanho da memória é meramente uma ilusão, chamou-se esta técnica de memória virtual (Virtual Memory) Denning [9].

O objetivo das técnicas anteriores era tentar utilizar cem por cento a memória, no entanto este esquema, no sentido lógico, pode utilizar mais do que cem por cento da memória, isto é, a soma de todos os espaços endereçáveis dos Job's a serem multiprogramados pode exceder o tamanho físico da memória. A filosofia básica para a utilização desta técnica é carregar apenas parte dos programas a serem processados em contrapartida com as outras técnicas onde os programas tem que ser carregados totalmente.

A Fig. 2.12 mostra um exemplo no qual 4 Job's serão multiprogramados; como indicado na Figura, os três primeiros Job's foram totalmente carregados, porém o quarto Job apenas duas das quatro páginas estão na memória. Entretanto este Job pode ser processado corretamente se as informações referenciadas estiverem todas contidas nestas duas páginas. (Ex. a instrução LOAD 1,1120 na posição 100 do espaço endereçável do quarto Job seria processada corretamente).

A tentativa de se processar um Job sem carregá-lo totalmente na memória é razoável ao fato de muitos programas usar apenas pequenas partes de seu espaço endereçável enquanto está sendo processado. Justificativas para tais adoções seriam:

a) As rotinas de erros, escritas pelo usuário não precisam estar na memória, pois são utilizadas apenas quando um erro nos dados ocorrer.

b) Certas partes de programas são meramente exclusivas ou não precisam ser processados constantemente. (Ex. Classificação alfabética ou numérica de folhas de pagamento).

c) Algumas rotinas são usadas meramente exclusivas durante um processamento (Ex. Rotina de entrada, rotina de cálculos e rotina de saída).

As páginas que não foram carregadas na memória são

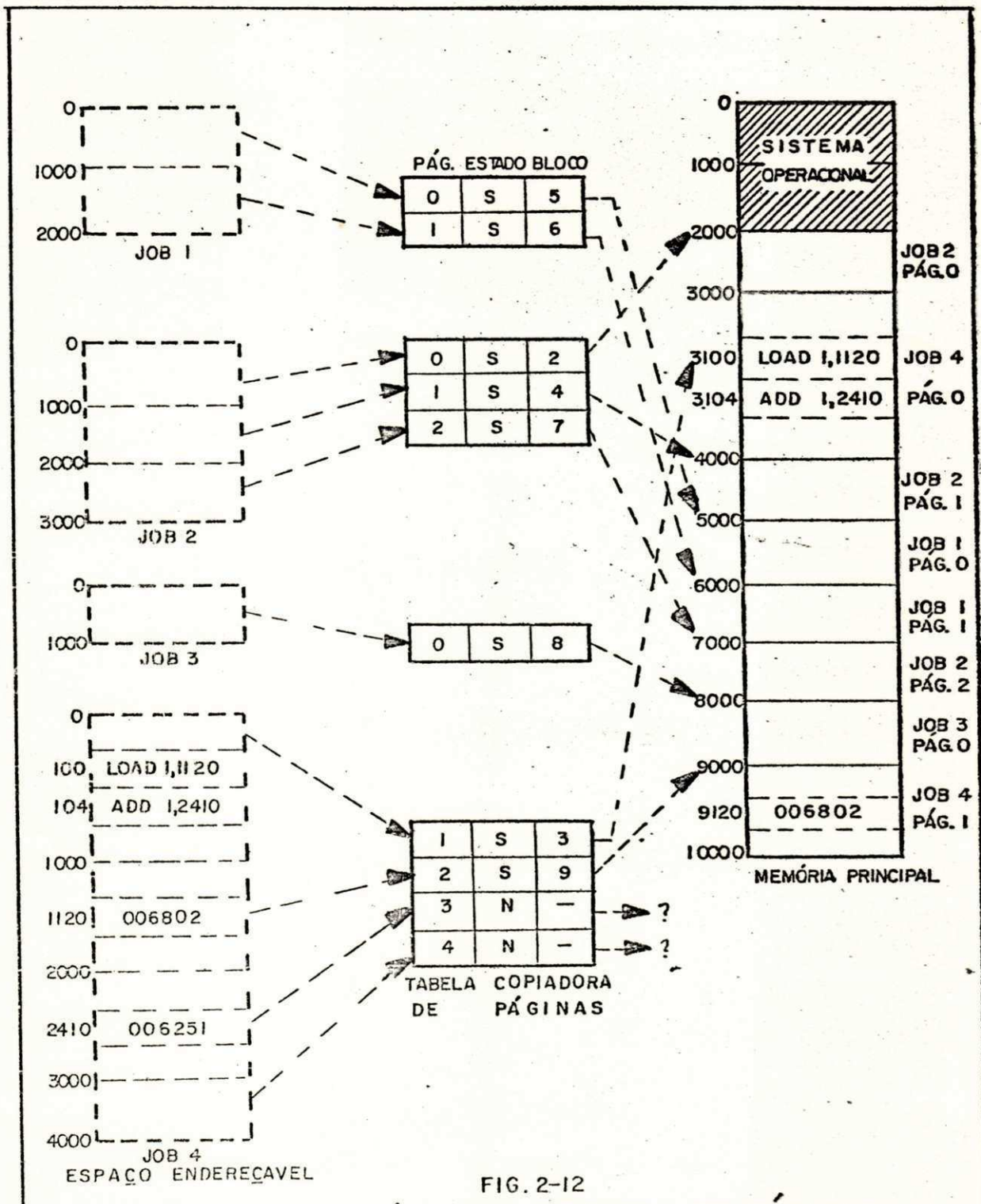

FIG. 2-12

ALOCAÇÃO DE MEMÓRIA COM DEMANDA DE PÁGINA

armazenadas na memória auxiliar (discos, tambores magnéticos). Quando uma destas páginas contém informações que forem referenciadas(ver Fig. 2.12 na qual a instrução ADD 1.2410 na posição 104 do JOB 4 referência uma página que não está na memória) uma interrupção no processamento do programa ocorrerá até que a referida página seja carregada. O reconhecimento se uma página encontra-se ou não na memória é feito com auxílio da tabela copiadora de páginas na qual inclui-se um BIT de condição indicando se a página está na memória.(BIT DE CONDIÇÃO = S, ou, não encontra-se na memória BIT DE CONDIÇÃO = N).

Desta maneira diz-se que as páginas são, carregadas por demanda. O Gerenciamento de memória com demanda de página apesar de ser atrativo, por não necessitar de memória disponível do mesmo tamanho do espaço endereçável a ser processado , requer técnicas especiais a fim de se fazer as substituições das páginas. Já que antes de se carregar a página em demanda , uma terá que ser substituída e copiada na memória auxiliar, então haverá uma troca de página entre a memória principal e a memória auxiliar. A esta técnica chamou-se de substituição de página ou remoção de página, e para decidir qual a página a ser removida da memória, algoritmos de substituição tem sido o objetivo de vários pesquisadores (Belady,[2]; Coffman[6]; Denning,[9],[10]; Gustavson,[4]; Randel[20] e Chu,[5]) como veremos nos capítulos seguintes.

O esquema com demanda de página é usado em alguns sistemas operacionais contemporâneos tais como; VS/1, VS/2 e VM/370 no sistema IBM/370, Multics no Honeywell 6180, e VMOS no UNIVAC série 70/46.

2.5.1. Algoritmo do Software

O esquema com demanda de página dá grande flexibilidade para o sistema operacional, o qual tem recebido considerável atenção. Neste parágrafo enunciaremos alguns dos vários algoritmos de substituições existentes, deixando as suas discussões e comparações para os capítulos seguintes o que será o objetivo deste trabalho. Dos vários algoritmos existentes os mais utilizados ultimamente são:

a) ALGORITMO FIFO (FIRST IN - FIRST OUT)

b) ALGORITMO LRU (LEAST RECENTLY USED)

c) ALGORITMO WS (WORKING SET)

d) ALGORITMO PFF (PAGE FAULT FREQUENCY)

2.5.2 Vantagens

O Gerenciamento com Demanda de página tem todas as vantagens discutidas com relação a memória paginada. Particularmente elimina a fragmentação e não necessita de compactação.

Outras vantagens adicionais ainda são:

1 - Utilização de memória virtual, onde o Job a ser processado não está condicionado ao tamanho físico da memória.

2 - Maior eficiência na utilização da memória, consequência do não armazenamento na memória da parte do Job que não será processado no momento.

3 - Multiprogramação ilimitada, isto é, a soma do tamanho de todos os Job's a serem multiprogramados pode ser maior do que o tamanho físico da memória.

2.5.3 Desvantagens

Afora as desvantagens mencionadas na utilização de memória paginada (tais como, aumento do custo de Hardware, Quebra de Página e ocupação da memória pelas tabelas), outras desvantagens aparece neste esquema tais como:

1 - O número de tabelas e a improdutividade dos processadores derivado das interrupções dos programas que estão sendo processados, devido a falta de página, é maior do que no simples esquema de gerenciamento de memória paginada.

2.6 Conclusões Gerais

Neste capítulo apresentaram-se alguns dos esquemas conhecidos, para utilização da memória dos computadores, bem como uma descrição sucinta sobre como funciona cada técnica. No capítulo seguinte far-se-á uma análise sobre o esquema de gerenciamento de Memória com demanda de página, na qual será feita uma apresentação mais detalhada sobre o que é página e das técnicas de paginação mais utilizadas, citadas no parágrafo 2.5.1.

CAPÍTULO 3

CONCEITO DE PÁGINA E TÉCNICAS DE PAGINAÇÃO

O objetivo deste capítulo é fazer uma exposição sobre o conceito e a utilização de página e as técnicas de paginação mais comumente usadas, abordando os seguintes itens: O que se entende por página, por que deveremos usar a Paginação , como faremos para operar um Sistema Paginado e finalizaremos dando uma descrição sucinta das técnicas de Paginação mais utilizadas.

3.1 O Que é Página

Páginas são blocos (conjunto de informações) de tamanho fixo, definidos em função dos programas a serem processados e do sistema que os comportará. Considerando que tanto os programas como o sistema (memória) utilizado, podem ser divididos em partes iguais de tamanho fixo, onde teremos uma correspondência unívoca entre as páginas dos programas e as da memória, por convenção, chamar-se-ão simplesmente PÁGINAS (espaço endereçável pelo programador "ADDRESS SPACE") os blocos que conterão os programas, e Blocos (Page Frame) ou espaço de memória (Memory Space) a divisão feita na memória principal do sistema. Num determinado sistema o número total de páginas para

cada programa a ser processado sempre será maior do que o número de blocos reservados para este programa e mesmo assim ele pode ser processado a contento, razão pela qual as páginas também são chamadas de endereços virtuais(Virtual Address), visto que parece ser impossível processarmos N páginas na memória principal que contém M blocos com N>>M.

3.2 Por que Devemos Usar a Paginação

Após o aparecimento de linguagens de alto nível, nos meados da década de 1950, os usuários dos computadores voltaram-se mais para resolver os problemas de programação surgidos, devido a ampla utilização dos computadores nos meios comerciais, do que propriamente com os detalhes de máquina, detalhes esses que constituíam o grande obstáculo para a utilização dos computadores em grande escala praticamente solucionado. Com o desenvolvimento dessas linguagens tornou-se evidente a necessidade de uma organização na memória dos computadores com o objetivo de obter-se um melhor desempenho e rendimento, surgindo consequentemente os supervisores do sistema constituídos de uma série de programas próprios de cada sistema computador, os quais organizam e distribuem os recursos disponíveis. Em decorrência de toda esta evolução e alterações realizadas nos computadores verificou-se a necessidade de construir-se máquinas com memória cada vez maior, solução essa não muito viável devido ao alto custo do sistema resultante. Buscando uma

melhor solução para esses problemas foi que em 1961 um grupo de pesquisadores propôs um novo conceito de armazenamento de informações[9], chamado hoje MEMÓRIA VIRTUAL (Virtual Memory). A idéia básica consiste em dar ao programador a ilusão de que existe uma memória muito grande à sua disposição, mesmo que na realidade o computador possua uma memória relativamente pequena. Por conveniência, ao conjunto de endereços referenciados pelos programadores chamaremos de Páginas, ou também Endereços Virtuais (Virtual Address) e o conjunto de vários Endereços Virtuais, Espaço de Endereços (ADDRESS SPACE). Enquanto que os blocos da memória também serão chamados Endereços de Memória (Memory Address) e ao conjunto destes, Espaços de Memória (Memory Space).

Existem duas técnicas para a utilização de Memória Virtual, diferindo conceitualmente na maneira de como organizar os Endereços Virtuais. Se estes endereços tiverem tamanho variável a técnica será chamada de Segmentação, enquanto que, se tiverem tamanho fixo, a técnica será chamada de Paginação ; os Endereços Virtuais serão as páginas e os Endereços de Memória, os Blocos. Vimos que passou-se a utilizar Memória Virtual para melhorar o desempenho de um sistema(devido a ilusão que se tem do sistema possuir uma memória realmente grande). Por isso passou-se a utilizar Memória Virtual em quase todos os sistemas de computadores modernos, e a Paginação é uma das técni-

cas bastante empregada .

3.3 Como Opera um Sistema Paginado

O objetivo de um sistema com paginação é dar maior flexibilidade na utilização da memória virtual reduzindo também a quantidade de informações a serem copiadas da memória auxiliar para a memória principal. A memória principal é organizada em blocos, que servem para acomodar as páginas. A identificação de cada bloco será feita através do seu Endereço na Memória (Frame Address) que consiste do endereço da primeira palavra de cada bloco. As informações a serem copiadas constituem um conjunto N= { 0,1,..., n-1 } de páginas nos M= {0,1,..., m-1} blocos em que foi dividida a memória, com N > M.

Para melhor compreensão de um sistema paginado, daremos nomes aos Endereços da Memória e às páginas. Com essa finalidade teremos a definição de uma função f:N → MU { Ø } tal que a cada instante teremos:

$$f(a) = \begin{cases} a' & \text{se o item } a \text{ está em } M \text{ no endereço } a' \\ \emptyset & \text{se o item } a \text{ não está em } M. \end{cases}$$

Esta função f é conhecida como Copiadora de Endereços (Address Map), na Fig 3.1 é mostrado como esta função co -

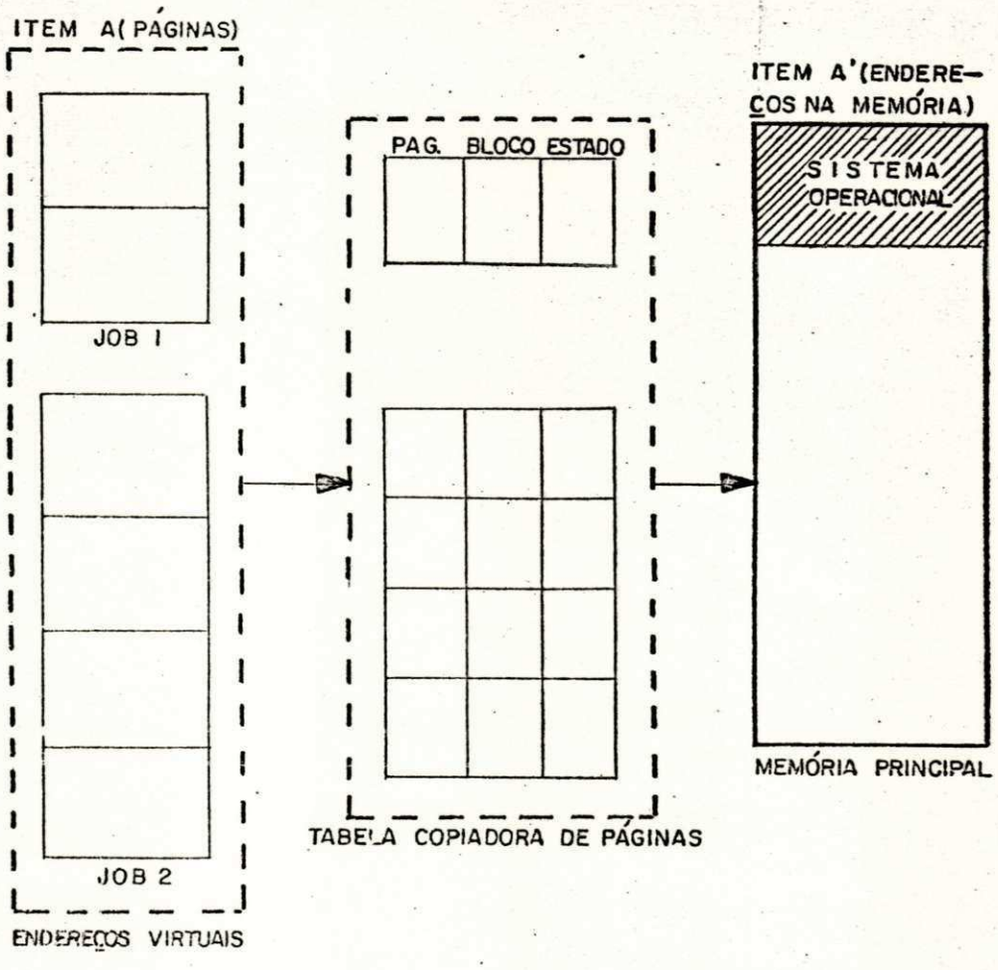

FIG. 3-1

FUNÇÃO f COPIADORA DE ENDEREÇOS

piadora atua no sistema.

Os Endereços Virtuais (Páginas) são guardados na memória auxiliar (item a) e quando referenciados serão copiados na memória principal no Endereço a' através da tabela copiadora de páginas. A coluna, Estado (paragráfo 2.4) da tabela copiadora será atualizada colocando-se um S (sim) para indicar que a página encontra-se na memória (f(a) = a') e um N (não) caso contrário (f(a) = ∅). Se ao referenciar-se uma página o seu Estado correspondente na tabela estiver com um N, o processamento do programa será interrompido até que a mesma seja copiada na memória. A seguir o Estado da tabela copiadora será atualizado colocando-se um S e o processamento continuará, se ao tentar-se copiar uma página a memória estiver totalmente ocupada, uma das páginas terá que ser removida da memória utilizando-se uma Regra de substituição (Replacement Rule) dando lugar a nova página a ser alocada.

Com a finalidade de dar melhor eficiência ao sistema, Hardwares associados à memória são empregados para auxiliar as tabelas copiadoras.

3.4 Descrição das Técnicas de Paginação

Existem duas maneiras básicas de se utilizar técni-

cas de paginação nos sistemas que utilizam memória virtual. A diferença consiste na Localidade (Locality), que é a porção de memória reservada para cada programa, podendo esta porção ser constante (partição fixa) durante todo o processamento , ou variar (partição variável) conforme a necessidade do pro - grama:

a) Partição Fixa - Para cada programa em execução é pré-fixada uma porção da memória (determinado número de blo - cos), menor do que o total das páginas em que foi particionado o programa, e este número será constante enquanto o mesmo estiver sendo processado.

b) Partição Variável - A porção de memória reservada para cada programa em execução, poderá se compactar e expan - dir dinamicamente durante o seu processamento, um determinado programa poderá ocupar espaço variável de memória (número de blocos) a cada instante.

Dentre as várias técnicas existentes para a substituição de páginas na memória, descreveremos apenas as mais utilizadas atualmente, como os Algoritmos LRU (Least Recently Used) e FIFO (First In-First Out) que utilizam partição fixa e o Algoritmo Working Set [11] utilizando partição variá - vel.

Outras técnicas utilizando também partição variável propostas recentemente, serão descritas, tais como, o Algoritmo PFF (Page Fault Frequency) proposto por CHU-OPDERBECK [5] e o recentemente desenvolvido por SADEH [21] Algoritmo CPFR (Controlled Page Fault Rate).

Os algoritmos serão apresentados na sequência: Apresentação, Filosofia de Substituição das Páginas, Procedimento na Substituição e Considerações acerca de sua implementação.

3.4.1 Algoritmo LRU

O Algoritmo LRU, abreviação de "Least Recently Used" (menos usada recentemente), seleciona para remoção as páginas que por certo período de tempo (relativamente Grande) não foram referenciadas e encontram-se na memória. A filosofia básica deste algoritmo é que se uma página foi referenciada na memória, provavelmente ela será necessária novamente em seguida e inversamente. Se uma página não foi referenciada por um determinado período de tempo, possivelmente não o será futuramente. Para determinar-se a página que foi a menos usada recentemente, o algoritmo deve manter uma lista na qual as páginas que se encontram em execução devem estar ordenadas de acordo com o seu uso. Esta lista é conhecida como PILHA LRU e deve ser atualizada após cada referência.

A Fig. 3.2 mostra uma configuração da Pilha LRU.

O topo da pilha contém o índice da página que está sendo usada no momento e a base contém o índice da página menos usada ultimamente. Supondo que em algum momento a configuração da pilha seja a mostrata na Fig. 3.2a e a próxima referência seja a página de nº K, então no processo de atualização, ao se referenciar esta página, o seu índice será deslocado para o topo da pilha e todos os outros índices que estavam acima dele serão deslocados uma posição abaixo, Fig. 3.2b.

Devido ao alto custo para a implementação do algoritmo LRU, uma aproximação foi feita e sendo usada em alguns sistemas IBM. Esta aproximação do algoritmo LRU, consiste em associar-se a cada bloco da memória um BIT DE REFERÊNCIA, e em lugar de mover-se o índice da página referenciada, apenas o BIT é atualizado, colocado para 1(um). Sempre que uma demanda de página ocorre, o algoritmo utiliza um apontador para pesquisar os blocos da memória de uma maneira cíclica. A pesquisa é interrompida quando um BIT com o valor 0 (Zero) for encontrado e todos os BIT'S pelos quais o apontador passou são feitos iguais a 0 (Zero). A página correspondente ao BIT encontrado com o valor 0(Zero) será substituída pela página em demanda, e este BIT será atualizado com o valor 1. Se uma nova demanda ocorrer o apontador iniciará a pesquisa no ponto onde tinha pa-

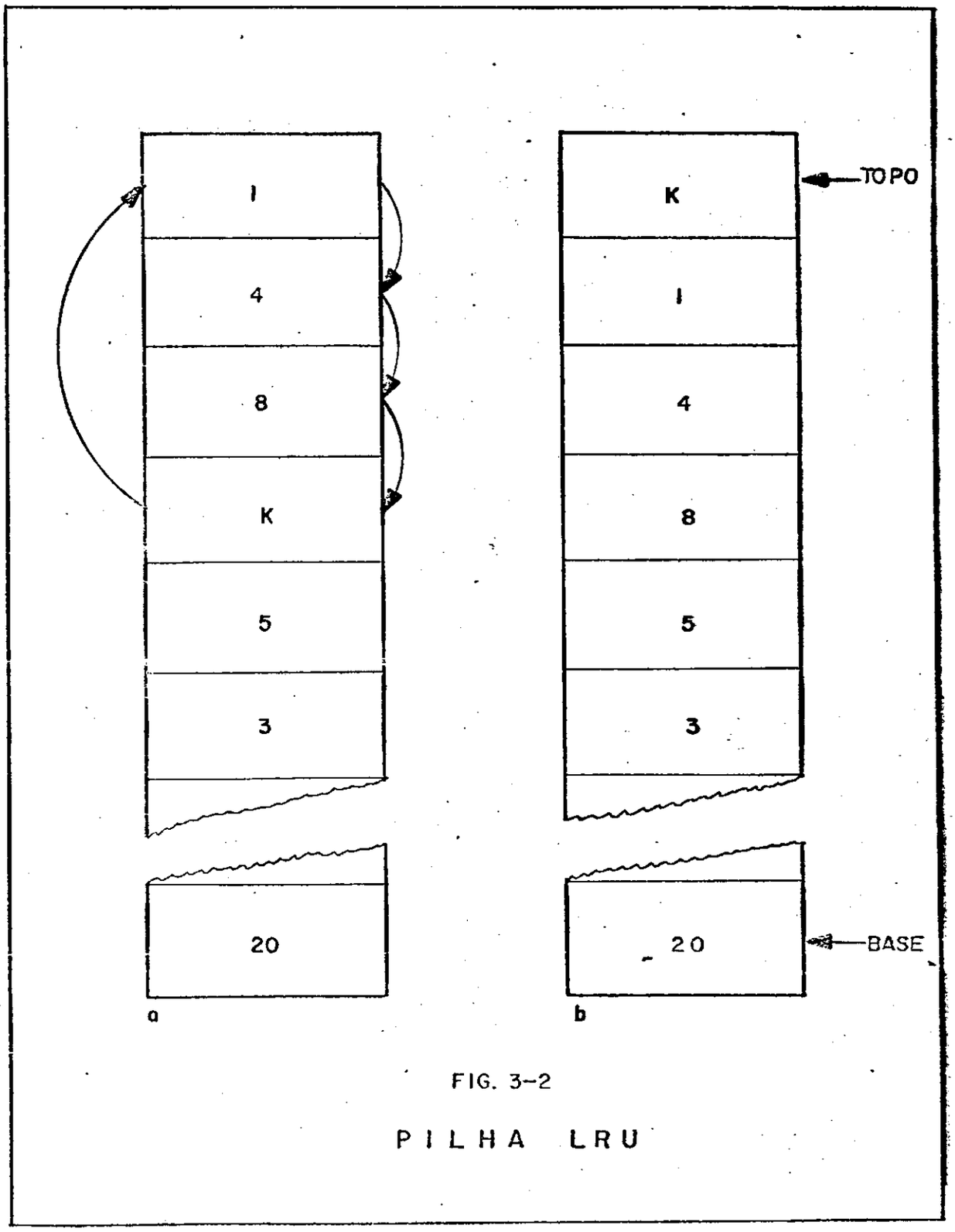

FIG. 3-2

PILHA LRU

rado.

3.4.2 Algoritmo FIFO (First In-First Out)

O algoritmo First In-First Out (Primeiro a entrar - primeiro a sair) substitui as páginas que por mais tempo estiveram na memória. Este critério supõe que os programas tendem a seguir uma sequência de instruções em seu interior. Portanto uma página que está por um longo período de tempo na memória, adotando este critério, será a que tem menor probabilidade de ser referenciada novamente, logo poderá ser substituída. A sua implementação é simples e feita da seguinte maneira: uma tabela com as páginas referenciadas é organizada, sendo sua ordenação, de maneira cíclica. E elas são incluídas nesta tabela a medida que forem referenciadas. Um apontador aponta para a página que por mais tempo está na memória, quando esta for substituída o ponteiro passará a apontar para a sua vizinha que passa a ser mais antiga, assim a ordenação cíclica é mantida.

Neste método considerações devem ser feitas a respeito da possibilidade de porções de programas não serem executadas sequencialmente como se considerou, pelo contrário, devemos considerar a presença de pequenos Loops, derivações, chamadas de sub-rotinas e estruturas semelhantes que causam uma dis

persão sobre as páginas e consequentemente uma desorganização na sequência das instruções, o que vai de encontro a filosofia considerada.

3.4.3 Algoritmo Working Set

Recentes pesquisas mostraram que utilizando-se as partições de memória dinamicamente (Partição Variável), resulta uma melhor utilização da memória.

O algoritmo Working Set é uma das técnicas para utilizar a memória dinamicamente, substitui as páginas que não foram referenciadas durante os últimos Tmseg onde T é um parâmetro do algoritmo dado em milisegundos (1 mseg = 1000 páginas referenciadas). Assim o desempenho desse algoritmo dependerá muito da escolha do parâmetro T e da característica dos programas. A sua filosofia é a de manter na memória as páginas que foram necessárias durante os últimos Tmseg. Em geral ele pode ser considerado como um algoritmo LRU com a porção de memória disponível para cada programa de tamanho variável.

Para determinar-se as páginas a serem removidas da memória, um dispositivo auxiliar é utilizado, registrando todas as páginas referenciadas, nos últimos Tmseg; essas páginas permanecem na memória e as que não foram referenciadas são re-

movidas para alocar-se a página em demanda. Na implementação real deste algoritmo uma chave é associada a cada bloco, para mostrar se o bloco está disponível ou não. Um relógio também é associado a cada bloco da memória, marcando o tempo decorrido desde a última referência feita a página que se encontra neste bloco. Se este tempo for maior do que o parâmetro T e esta página não foi referenciada, então será removida. Observação tem que ser feita a respeito da dificuldade de se determinar o tempo decorrido desde a última referência a uma determinada página deste algoritmo. Desta maneira a implementação deste algoritmo é realmente de custo altissimo.

3.4.4 Algoritmo PFF (Page Fault Frequency) [5]

O Algoritmo "Ideal" para substituição de páginas tem que depender o menos possível do conhecimento à priori acerca do procedimento do programa. Assim no lugar de fazer-se uma escolha antecipada dos parâmetros, o processo de decisão será dinâmico e baseado em informações que são fornecidas durante a execução do programa. O algoritmo PFF que é uma adaptação do Working Set é baseado nesta filosofia e tenta controlar o nível da taxa de Falta de Página usando a relação inversa entre a disponibilidade de alocação e a taxa de Falta de Páginas.

As considerações que levaram para a formulação deste algoritmo são:

a) Uma Taxa de Falta de Páginas Alta, indica que a execução do programa é ineficiente, como consequência do espaço alocado para este programa ser pequeno.

b) Uma Taxa de Falta de Páginas Baixa, indica que o espaço alocado ao programa é relativamente grande no que resulta num desperdício de memória, portanto para melhorar o desempenho do sistema um ou mais blocos da memória devem ser liberados.

Seguindo estas considerações a idéia básica do algoritmo PFF é monitorar a frequência de Faltas de Páginas durante a execução do programa. Quando esta frequência estiver acima do parâmetro K (escolhido a priori) a disponibilidade de alocação será aumentada pelo sistema levando as páginas faltosas para a memória sem remoção de outras páginas, este aumento reduzirá a taxa de falta de páginas. E se a taxa de falta de páginas estiver abaixo de K (medido em número de páginas que faltaram por mseg) as páginas não referenciadas serão liberadas e a em demanda alocada. Na implementação real o que se faz é expressar-se o parâmetro K = 1/T onde T é o tempo crítico entre as faltas de páginas escolhido a priori. Quando ocorre uma falta de página, verifica-se o tempo decorrido desde a última interrupção, se este tempo for menor ou igual a T,então a página em falta é adicionada à memória e nenhuma é retirada. Caso o tempo for maior do que T, então as páginas referencia -

das são mantidas, a página em falta substituída e o restante liberadas. Para monitorar o intervalo entre páginas ausentes, o algoritmo requer apenas um relógio para cada programa em execução. Os testes e as decisões de alocação são feitos apenas quando ocorre uma interrupção devido a uma falta de página.

3.4.5 Algoritmo CPFR (Controlled Page Fault Rate) [21]

Este algoritmo possui dois parâmetros T e Z e utiliza um relógio que marca cada referência feita às páginas na memória, sendo reposicionado no início de cada marcação. Toda vez que ocorre falta de página o conteúdo do relógio C é comparado com os parâmetros T e Z. De acordo com o resultado da comparação, o algoritmo toma as seguintes decisões, isto quando ocorre uma interrupção devido a uma falta de página.

a) Se $C < T$, a página faltosa é alocada na memória, sem substituição de qualquer página. O relógio não será reposicionado.

b) Se $T \leq C \leq Z$, temos o fim da marcação atual e o início de uma outra. A página faltosa é alocada na memória e as que não foram referenciadas no intervalo de tempo anterior, são liberadas. O relógio é reposicionado.

O parâmetro T(cuja unidade de tempo é definido como

o tempo de acesso a uma palavra da memória) é determinado à priori no sistema. O parâmetro Z é um limitador de tempo introduzido para tomar cuidados especiais caso em que o programa tenha todas as suas páginas na memória, pois para cada referência temos:

a) Se C=Z, temos o fim da marcação anterior e o início de uma outra. As páginas que não foram referenciadas durante a marcação anterior serão substituídas e o relógio reposicionado.

Então vemos que o algoritmo liberará blocos da memória sempre que este tempo limite for atingido mesmo que não ocorra interrupção devido a uma falta de página. Para reconhecer uma página não referenciada o algoritmo manterá uma tabela de páginas usando um BIT associado a cada elemento.

Estes BIT'S são reposicionados no início de cada marcação e eles tem o valor 1 (um) ou 0 (zero) conforme a sua página foi referenciada ou não. Este algoritmo utiliza apenas um relógio para cada programa em execução.

3.5 Sumário das Técnicas de Paginação

Vimos que várias são as técnicas para utilização de

memória virtual nos sistemas de computadores, onde o objetivo de cada uma é dar melhor eficiência ao sistema, com esta finalidade a filosofia de substituição das páginas e a complexidade varia para cada técnica.

No próximo capítulo faremos um estudo comparativo das técnicas aqui apresentadas na tentativa de determinar qual a melhor técnica para determinado sistema de computador.

CAPÍTULO 4

COMPARAÇÃO ENTRE AS VÁRIAS TÉCNICAS DE PAGINAÇÃO EXISTENTES

Neste capítulo faremos um estudo comparativo entre as técnicas apresentadas no capítulo anterior. Este confronto, objetivo principal deste trabalho, será baseado nos diversos resultados de pesquisas publicadas nas referências [1], [5], [7], [8], [12], [13], [15], [21], existem vários pontos para se avaliar o "desempenho" de um algoritmo de paginação. Faremos a nossa análise, primeiramente apresentando o que se necessita e quais as dificuldades na implementação destas técnicas, e posteriormente faremos um estudo simultâneo entre estas técnicas determinando, qual a que proporciona menor custo de memória (que será definido posteriormente) e melhor eficiência, obtendo-se assim a que mais se aproxima do algoritmo ótimo(definido no capítulo seguinte). Intuitivamente podemos dizer que é aquela que causar a menor quantidade de falta de página e ocupar menor espaço de memória em determinado período de tempo.

4.1 Implementação

Algoritmo FIFO: para a utilização do algoritmo FIFO necessita-se adaptar um mecanismo (Software) que faça a orde-

nação nos blocos da memória, indicando qual o bloco recentemente alocado e qual o mais antigo (próximo a ser removido). Esta ordenação é feita na tabela de blocos da memória (descrita no 2º capítulo), numa estrutura (FIFO) utilizando apontadores. Este algoritmo é de fácil implementação e elimina a possibilidade de se carregar uma página na memória e imediatamente remove-la. Por outro lado uma página será removida da memória, quando se tornar a mais antiga, mesmo que tenha sido usada frequentemente e que seja necessária futuramente. Um outro ponto a salientar é o efeito notado em alguns casos, chamado anomalia FIFO (estudado por Belady, [2]).

Algoritmo LRU: como descrito no paragráfo 3.4.1, para a sua real implementação este algoritmo utiliza Bit's de referências como um meio no processo de atualização das páginas referenciadas. Um Bit de referência é associado a cada bloco de memória, e este Bit é transformado em 1(um) pelo Hardware sempre que esta página for referenciada. O Software para remoção de páginas, periodicamente substitui os Bit's de referência por 0 (zero). A página a ser removida é a que não foi referenciada por mais tempo, desta maneira uma página que for utilizada frequentemente, encontrar-se-á na memória para referência futura. Este algoritmo é de fácil implementação quando se usam os Bit's de referência e é uma aproximação do verdadeiro algoritmo, LRU definido para utilizar pilhas.

Algoritmo Working Set: conforme vimos no paragráfo 3.4.3 para a implementação real deste algoritmo, uma chave associada a cada bloco de memória faz-se necessária para indicar que o seu correspondente bloco pode ser usado por uma página de qualquer programa. Para se determinar qual a página a remover, um relógio é associado a cada bloco de memória, o qual mede o tempo decorrido desde a última referência feita a cada página. Uma desvantagem deste algoritmo consiste na necessidade de determinar-se o tempo exato das páginas não referenciadas nos últimos Tmseg. Contudo, uma vantagem seria que durante a Execução Real, a frequência de páginas ausentes pode ser ligeiramente inferior a observada na simulação, pois pode acontecer que uma página deixe o conjunto de trabalho (Working Set) e posteriormente retorne a ele sem neste meio tempo ser removida da memória. Em geral a implementação deste algoritmo é de custo alto.

Algoritmo PFF: a implementação deste algoritmo é realmente simples. Necessita-se apenas de um relógio na CPU para medir o tempo entre as faltas de páginas de cada tarefa, onde é medido apenas o tempo de processamento(tempo virtual) de cada uma. A tabela de página pode ser usada para determinar quais as páginas estão presentes na memória. Para os sistemas que utilizam Bit de referência, esta característica pode ser usada para determinar as páginas que foram referencia-

das desde a ocorrência da última falta de página. Quando uma falta de página ocorre, os Bit's de referência são reposicionados e o supervisor determina se a tarefa em execução está operando abaixo do nível crítico de frequência de falta de página K. Se a última falta de página ocorreu com um intervalo de tempo maior do que $T = \frac{1}{K}$ mseg, os Bit's de referência serão usados para determinar quais as páginas devem ser removidas da memória.

4.2 Estudo Comparativo do "Custo" de Memória e da Eficiência

Começaremos definindo qual o custo de uma porção de memória ocupada por um programa. E qual a eficiência do programa que está sendo executado num meio paginado. A seguir apresentaremos os resultados obtidos por, KING [16], FRANASZEK - WAGNER [13], CHU-OPDERBECH [5] , comparando estas técnicas, em alguns casos particulares.

Custo de Memória: o custo de memória será um parâmetro do ponto de vista econômico para comparar-se os vários algoritmos de paginação. Um dos critérios para se avaliar este custo é determinar-se o produto espaço x tempo, que pode ser considerado como sendo proporcional ao custo de memória. BELADY e KUEHNER [3] definiram o produto C de espaço x tempo durante o

intervalo (0,t) de Tempo Real (tempo de Processamento mais tempo de espera de páginas em falta) como sendo:

$$C = \int_0^t s(x)\,dx$$

onde s(x) é o espaço de Memória ocupado por um processo no tempo x. É importante ressaltar que o tempo real para determinadas informações na memória poderá ser muito maior do que o tempo virtual (tempo de processamento). Isto devido a multiprogramação (onde um programa pode estar esperando por um processador) e ao tempo decorrido em esperar páginas faltosas.

Eficiência: a eficiência E para um programa que está sendo executado é dada por:

$$E = \frac{\text{Tempo Virtual Total}}{\text{Tempo Real Total}} \quad \text{onde,}$$

Tempo Virtual Total = tempo total de processamento menos o tempo da espera de páginas faltosas.

Tempo Real Total = tempo total de processamento mais tempo total de espera de página.

Num sistema com multiprogramação, um outro fator se-

rá adicionado ao tempo real que será o de espera por um programa até que o processador se torne disponível.

Análise dos resultados obtidos, na comparação entre as técnicas: FIFO, LRU, WORKING SET e PFF.

Começaremos apresentando os vários estudos e pesquisas feitas entre as técnicas, FIFO e LRU, por MADNICK [19], KING [16]e FRANASHEK-WAGNER [13],onde todos obtiveram aproximadamente os mesmos resultados. Devido a sua simplicidade, o algoritmo FIFO foi usado em alguns sistemas de paginação anteriores. Em recentes pesquisas, entretanto, verificou-se que o FIFO tem um efeito peculiar, que foi chamado ANOMALIA FIFO (Belady), e que em certas circunstâncias, ao adicionar-se mais memória para um determinado programa, a utilização de memória poderá resultar menos eficiente. Enquanto que o Algoritmo LRU , foi verificado é imune a peculiaridade FIFO observada por Belady, isto é, ele é uma função monótona da memória principal. Ainda em recente pesquisa feita por King [16],foi obtido o seu desempenho, que é bem mais próximo do Algoritmo Ótimo(é o Algoritmo Ideal, onde se conhecem todas as páginas que serão necessárias no Programa, à priori) do que o FIFO, ver Fig. 4.1.

FRANASHER-WAGNER [13] também provaram (teorema 5) que o Algoritmo FIFO é menos eficiente do que o Algoritmo LRU em relação

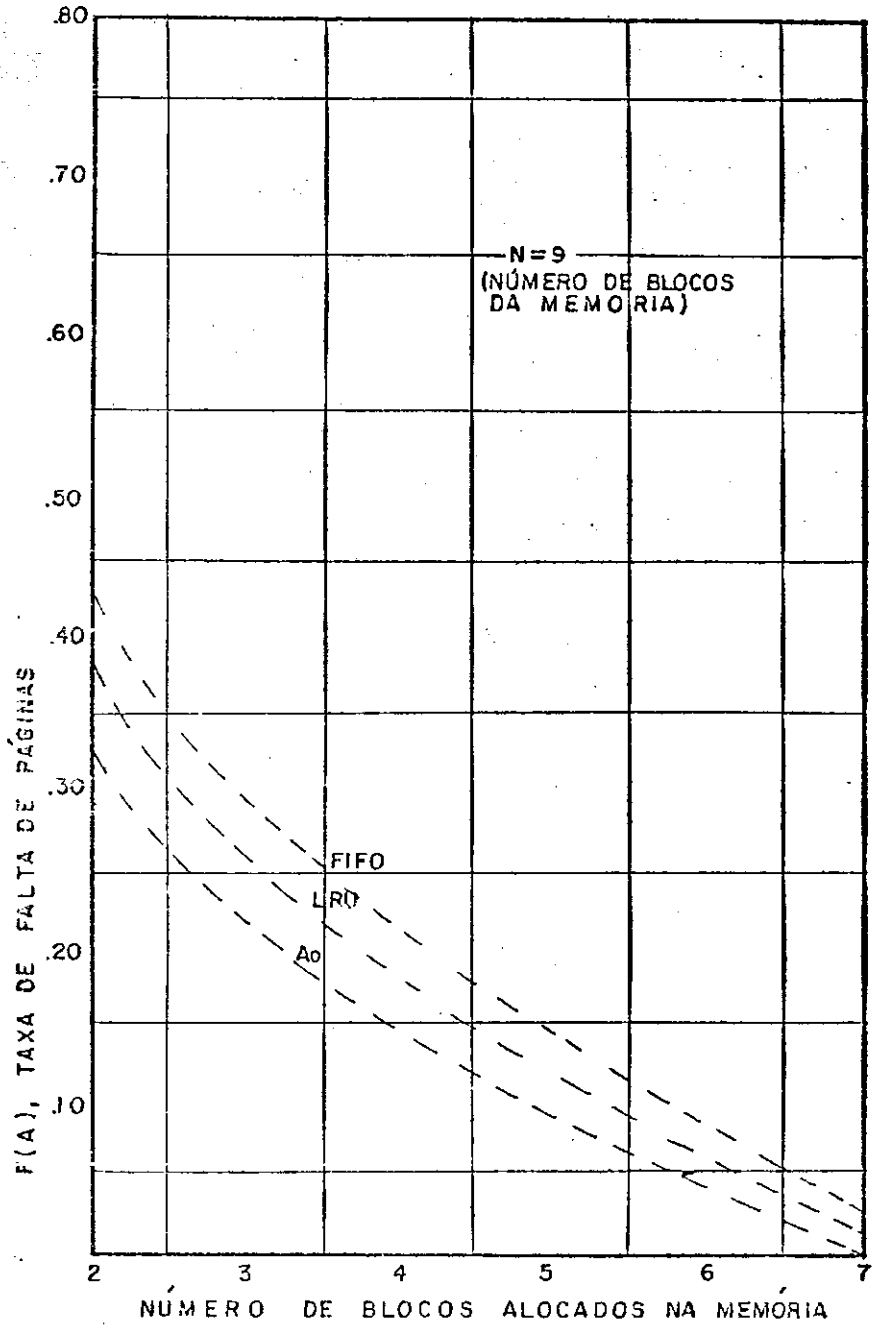

FIG. 4-1

COMPARAÇÃO DOS ALGORÍTMOS Ao(ALGORÍTMO IDEAL), LRU E FIFO

a utilização de Memória. King[16] na pesquisa que realizou, desenvolveu um modelo de um Programa em execução num meio paginado, baseado num modelo estocástico simples de um programa e num modelo para um algoritmo de Paginação usados por AHO-Denning-Ullman [1] e Denning - Chen-Shedler [12]. King demonstrou que F (FIFO) $\geq$ F (LRU), onde F (A) é a taxa esperada de falta de página para um determinado Algoritmo A. Donde concluímos que neste resultado, também a eficiência de LRU > FIFO, pois para determinado tamanho de memória, tanto melhor é a eficiência quanto menor for a taxa de falta de páginas, consequência da diminuição do número de substituições de páginas. A seguir descreveremos os resultados obtidos por CHU-OPDERBECK[5]quando da apresentação do algoritmo PFF (proposto pelos mesmos) e sua comparação com os algoritmos, Working Set e LRU. Para esta finalidade utilizaram 4 diferentes tipos de programas com características diversas, para a simulação destes Algoritmos. Um programa FORTRAN e um compilador FORTRAN (Fortcomp) foram escolhidos para representar programas com pequenas localidades. Um compilador meta 7 (traduz os Programas escritos em meta 7 para a linguagem ASSEMBLER Sigma - 7) e um compilador DCDL (Digital Control and Desing Language, escrito em meta 7) foram escolhidos como programas que utilizam grandes localidades. No confronto inicial feito entre os Algoritmos Working Set e LRU, observou-se que o Algoritmo Working Set tem o produto espaço x tempo, muito melhor e é menos sensível a variação do seu parâ-

metro do que o Algoritmo LRU. Disto conclui-se que a grande desvantagem do Algoritmo LRU é que não existe nada claro a respeito de quantas páginas devem ser alocadas para cada programa. A fim de assegurar-se uma execução eficiente sem desperdício de memória, já que este número varia durante a execução. Daí vemos que o Algoritmo Working Set constitui uma melhor solução para este problema, em vista do espaço de memória a ser alocado ser automaticamente determinada pelo número de páginas referenciadas durante os últimos Tmseg.

4.3 Comparações e Resultados

Finalmente apresentaremos as conclusões a que chegaram CHU-OPDERBECK [5]ao compararem o Algoritmo PFF com os Algoritmos Working Set e LRU. No Algoritmo PFF foi utilizado um valor específico para o parâmetro K = 1/100 páginas faltosas / mseg. A fim de compará-lo com o algoritmo LRU contendo um número variado de blocos. A Fig. 4.2 obtida na simulação das técnicas PFF e LRU mostra que o produto espaço x tempo do Algoritmo PFF para todos os 4 programas tem menor valor do que o menor ponto do produto espaço x tempo atingido pelo Algoritmo LRU , em cada programa. Isto implica que o desempenho do Algoritmo PFF é melhor do que o do LRU. O resultado acima também revela a ineficiência da alocação de memória com tamanho do espaço fixo. Na Fig. 4.3 temos o resultado obtido na simulação das téc

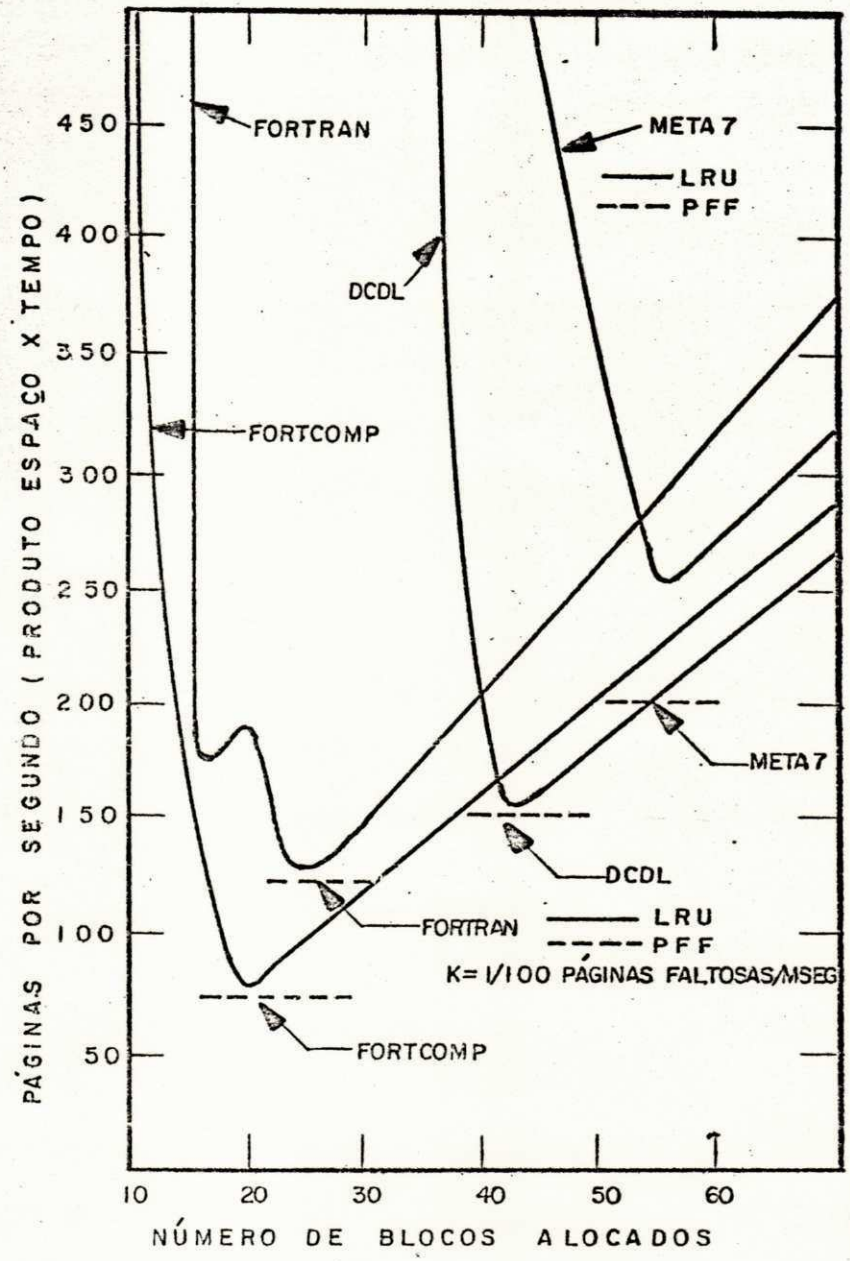

FIG. 4-2

COMPARAÇÃO ENTRE OS ALGORÍTMOS LRU E PFF

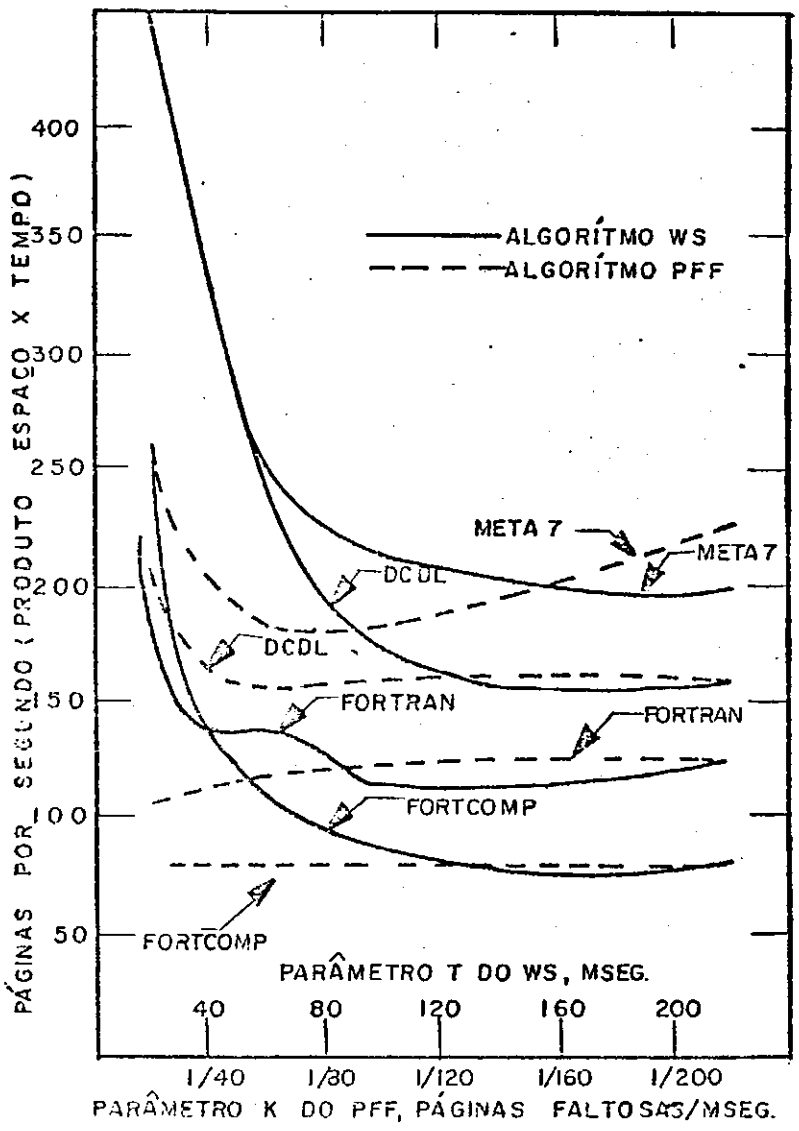

FIG. 4-3

COMPARAÇÃO ENTRE OS ALGORÍTMOS PFF E WORKING SET

nicas PFF e WS. O produto espaço x tempo foi traçado para os parâmetros, T do WS e K do PFF.

Vemos que para grandes valores de T, o produto espaço x tempo do Algoritmo Working Set é comumente inferior ao do Algoritmo PFF para valores correspondentes de K. O produto espaço x tempo do Algoritmo PFF é menos sensível a K do que o produto do Algoritmo WS o é para T. Afirmação idêntica podemos fazer com respeito a eficiência destas técnicas. Assim o menor produto espaço x tempo do algoritmo WS é comparável ao do algoritmo PFF enquanto que a eficiência do PFF é maior do que a do algoritmo WS. Observação deve ser feita que, apesar de todas as medidas obtidas terem sido realizadas com um único sistema de computador, as conclusões são igualmente aplicáveis a outros sistemas de paginação. A razão disto é que o procedimento dos programas é principalmente determinado pelas suas características, como por exemplo, Loop's, Modularidade de códigos, e Lay-out de dados. Sendo estas características relativamente independentes de comprimento de palavra ou conjunto de instruções, e comum a todos os grandes sistemas de computadores. Resultados semelhantes utilizando um outro sistema de computador, foram obtidos por Coffman e Varian's [7] .

CAPÍTULO 5

CONCLUSÕES E PESQUISAS FUTURAS

Após o estudo das várias técnicas existentes, faremos neste capítulo um resumo do que foi apresentado anteriormente e tentaremos determinar qual a técnica de melhor desempenho(apesar de sabermos que "Não Existe uma Fórmula Mágica que Solucione Todos os Problemas Existentes"), e qual a que melhor se adapta para determinados casos.

5.1 Resumo

Como vimos no capítulo 4, na comparação feita entre si das técnicas mais comumente usadas, e na comparação do algoritmo PFF (proposto recentimente por, CHU-OPDERBECK)[5] com as demais técnicas, este algoritmo apresentou um melhor desempenho, relativo ao produto espaço x tempo e a eficiência. Apesar da simulação destas técnicas terem sido feitas com casos particulares de programas e sistema de computação, vimos que estes resultados serão válidos também para outros sistemas, pois as características dos programas sempre são as mesmas em qualquer sistema. Ainda analisando o capítulo 4, vemos que de uma maneira geral, as técnicas que utilizam partição de memória variá -

vel, tem um melhor desempenho do que as que utilizam partição fixa. Afirmação esta que está de acordo com os resultados obtidos por COFFMAN-RYAN[7] em "Á STUDY OF STORAGE PARTITIONING USING A MATHEMATICAL MODEL OF LOCALITY". Para este resultado, Coffman e Ryan, utilizaram um processo estacionário de Gauss como um modelo no procedimento da execução dos programas devido a localidade. Este modelo foi aplicado diretamente para fazer a análise comparativa dos métodos de partições de memória, fixa e variável. As conclusões gerais foram que o método de partições variáveis é superior ao de partições fixas, exceto sob restritas considerações, como por exemplo quando a variação do tamanho do Working Set (Conjunto de Trabalho) é relativamente pequena.

Assim tanto das observações por nós realizadas quando da comparação das técnicas entre si no capítulo 4, como nos resultados obtidos por Coffman e Ryan utilizando um modelo matemático, conclusões idênticas foram obtidas, donde adiantaremos que as técnicas que utilizam partições variáveis deverão ser mais utilizadas nos novos sistemas a serem implantados.

5.2 Algoritmo Ideal para Substituições de Páginas

Da análise realizada nas técnicas de paginação vi-

mos que o seu desempenho era medido com base no Custo de Memória, e que um parâmetro que define bem este custo é o produto espaço x tempo e o desempenho é tanto melhor quanto menor for este produto. Então com o intuito de determinar-se uma filosofia para um algoritmo ideal, devemos ter como objetivo a minimização deste produto. Um critério a adotar para determinar este algoritmo, seria selecionar para remoção, a página que levará maior tempo sem ser referênciada. A razão disto é que, ao selecionar-se a página que passar maior tempo sem ser referênciada, estamos maximizando o tempo entre as páginas em falta, e com isto minimizando a taxa de falta de página e consequentemente o custo. Baseado nesta filosofia e considerando conhecidas a priori todas as referências a serem feitas no programa, resulta o ALGARITMO Bo que foi provado por COFFMAN-DENNING [6] ser o algoritmo ideal. O ALGORITMO Ao também adota essa filosofia e foi mostrado por AHO-DENNING-ULLMAN [1] ser um algoritmo ideal.

5.3 Pesquisas Futuras - Simulação

Dando continuidade e complementação a este trabalho, uma simulação das técnicas aqui analisadas poderá ser feita com o objetivo de obter-se empiricamente os resultados aqui aceitos por outros pesquisadores. Uma sugestão para esta simulação seria a de traçar-se o gráfico do algoritmo ideal e com-

parar os gráficos das demais técnicas com este algoritmo.

BIBLIOGRAFIA CONSULTADA

1 AHO, Alfred V.; DENNING, Peter J.; ULLMAN, E. Jeffrey - Principles of optimal page replacement. JOURNAL OF ACM, 18(1), Jan. 1971.

2 BELADY, L.A. - A Study of replacement algorithms for a virtual storage compute. IBM SYSTEMS JOURNAL, 5(2), 1966.

3 BELADY, L.A.& KUEHNER, C.J. - Dynamic space-sharing in computer systems. COMMUNICATIONS OF THE ACM, 12(5), May 1969.

4 BRAWN, Barbara S.& GUSTAUSON, Frances G.- Program behavior in a paging environment. IBM Watson Research Center, 1968. (Fall Joint Computer Conference)

5 CHU, Wesley W. & OPDERBECK, Holder - The Page fault frequency replacement algorithm. U.S. Office of Naval Research, Mathematical and Information Sciences Division, 1972 (Fall Joint Computer Conference).

6 COFFMAN Jr., Edward G. & DENNING, Peter J.- Operating systems theory.

7 COFFMAN Jr., Edward G. & RYAN Jr., Thomas A.- A Study of storage partioning using a matheratical model of locality. COMMUNICATIONS OF THE ACM, 15(3), Mar. 1972.

8 DENNING, Peter J. - The Working set model for program behavior. COMMUNICATIONS OF THE ACM, 11(5), May 1968.

9 DENNING, Peter J.- Virtual memory. COMPUTING SURVEY, 2(3), Sep. 1970.

10 DENNING, Peter J.- On modeling program behavior. 1972. (Spring Joint Computer Conference).

11 DENNING, Peter J. & SCHWARTZ, Stuart C. - Properties of the working - set model. COMMUNICATIONS OF THE ACM, 15(3), Mar. 1975.

12 DENNING, Peter J.; CHEN, Y.C.; SHEDLER, G.S. - A Model for program behavior under demand paging. IBM RESEARCH CENTER, Dec. 1968.

13 FRANASZEK, P.A. & WAGNER, T.J. - Some distribution-free aspects of pagnig algorithm perfomace . JOURNAL OF THE ACM, 21(1), Jan. 1974.

14 HANSEN, Per Brinch - Operating system principles.

15 INGARGIOLA, Giorgio & KORSH, James F. - Finding optimal demand paging algorithms. JOURNAL OF THE ACM, 21(1), Jan. 1974.

16 KING, W.F. - Analysis of paging algorithms. IBM RESEARCH CENTER, Mar. 1971.

17 KUEHNER, C.J. & RANDELL, B. - Demand paging in perspective. 1968. (Fall Joint Computer Conference).

18 MADNICK, Stuart Elliot - Storage hierarchy systems. 1972. (Thesis of Doctor of Fhilosophy).

19 MADNICK, Stuart E. & DONAVAN, John J. - Operating systems.

20 RANDELL B. & KUEHNER, C.J. - Dynamic storage allocation systems. COMMUNICATIONS OF THE ACM , 11(5), May 1968.

21 SADEH, Eitan - Analysis of the page fault frequency replacement algorithm. 1975 (Thesis of Doctor of Philosophy).

22 SHAW, Alan C. - The Logical design of operating systems.

23 TSICHRITZIS, Dionysios C. & BERNSTEIN, Philip A.- Operating systems.