UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

Ambiente Gráfico para Visualização de Imagens Tomográficas

Agma Juci Machado Traina
Jan Franz Willem Slaets

Nº 12

RELATÓRIOS TÉCNICOS

ICMC USP SÃO CARLOS

São Carlos – SP

Instituto de Ciências Matemáticas de São Carlos

ISSN-0109-2569

Ambiente Gráfico para Visualização de Imagens Tomográficas

AGMA JUCI MACHADO TRAINA

JAN FRANZ WILLEM SLAETS

Nº 12

RELATÓRIOS TÉCNICOS DO ICMSC

SÃO CARLOS

Nov./ 1993

Universidade de São Paulo
Instituto de Ciências Matemáticas de São Carlos
Departamento de Ciências de Computação e Estatística

Ambiente Gráfico para Visualização de Imagens Tomográficas

Agma Juci Machado Traina
Jan Franz Willem Slaets

Relatórios Técnicos do ICMSC
Nº 12

1 - Introdução.

O objetivo deste trabalho é a apresentação de um sistema dedicado, capaz de visualizar e manipular as imagens obtidas pelo tomógrafo por Ressonância Magnética, em desenvolvimento no Departamento de Física e Ciência dos Materiais do IFQSC [PAN_87] [TAN_87].

A construção da arquitetura dedicada baseada no processador TMS34010 [PAI_90], no Laboratório de Instrumentação Eletrônica (LIE), viabilizou o desenvolvimento de um ambiente gráfico, que possibilitasse ao usuário a utilização dos recursos oferecidos pelo "hardware".

Tendo em vista a particularidade das aplicações e das características do TMS34010, optou-se pelo desenvolvimento próprio de um ambiente gráfico. Dessa forma, o "software" resultante aproveitaria melhor as características do "hardware"; e seria também compatível com as necessidades do sistema de tomografia. Os sistemas mais utilizados atualmente, como XWindow e NeWS, são muito grandes e complexos para a arquitetura dedicada, não sendo adequados ao sistema de visualização em desenvolvimento.

As rotinas que constituem o ambiente gráfico foram desenvolvidas em linguagem de programação "C" na sua grande maioria, contudo as rotinas que acessam diretamente os recursos do GSP foram escritas em linguagem de montagem do processador TMS34010 [TEX_87].

Nas seções a seguir será detalhado o ambiente gráfico desenvolvido.

2 - O Ambiente Gráfico Desenvolvido.

Neste trabalho **Ambiente Gráfico** será definido informalmente como um sistema de "software" que, para uma dada arquitetura, fornece primitivas gráficas básicas, as quais podem ser facilmente acessadas por um aplicativo escrito em linguagem de alto nível. Além disso, um Ambiente gráfico deve fornecer ferramentas para a construção de aplicativos cuja interface suporte o conceito de "Amigabilidade com o Usuário". Exemplos típicos são: construção de janelas, "menus", tratamento de imagens, bem como todo o gerenciamento entre essas estruturas. Essas premissas foram levadas em conta para o projeto do ambiente gráfico desenvolvido.

Quanto ao ambiente gráfico optou-se por desenvolvê-lo de forma hierárquica e modular em três níveis: Nível Básico, Nível de Apoio e Nível de "Toolkit". Assim os níveis

superiores são funcionalmente dependentes dos níveis inferiores, e além disso, somente o nível mais baixo é dependente de máquina. Essa estrutura é apresentada na figura 1.

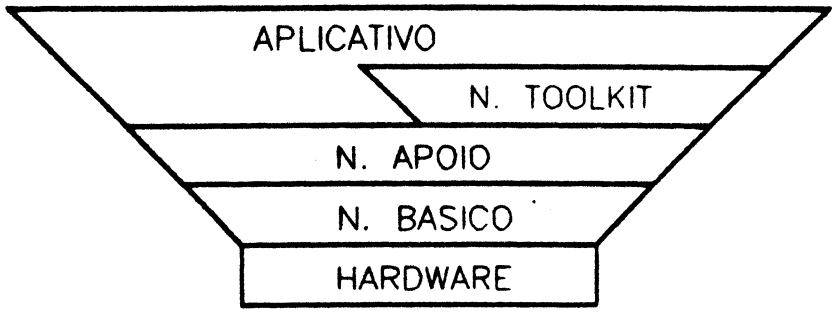


Figura 1 - Hierarquia do Ambiente Gráfico.

A seguir é descrito cada nível da hierarquia do ambiente gráfico:

Nível Básico: Neste nível estão as primitivas que solicitam as operações que acessam diretamente as primitivas gráficas que o "hardware" suporta. Tais primitivas são em grande parte escritas em linguagem montadora do TMS34010. Como exemplo, pode-se tomar as primitivas para desenhar pontos, linhas, retângulos, inicializar registradores do TMS34010, etc. As próprias estruturas de dados que gerenciam todo o ambiente gráfico (janelas, "menus", etc) encontram-se neste nível, fora do alcance do projetista de "software".

Nível de Apoio: Neste nível estão as primitivas que, utilizando aquelas do nível básico preparam o "hardware" para as operações gráficas do nível de "toolkit". Também encontram-se aqui as primitivas de comunicação com o "hospedeiro", as que solicitam-lhe recursos, tais como acesso a disco, teclado, transferência de imagens, etc., bem como as primitivas de gerenciamento do ambiente gráfico. Todas estas primitivas foram implementadas como rotinas escritas em linguagem "C".

Nível de "Toolkit": Este constitui-se no nível onde estão as primitivas usualmente utilizadas para a construção dos aplicativos. Incluem-se aqui todas

as primitivas de manipulação de "menus", janelas, imagens, tratamento de eventos, etc.

Tal como pode-se notar na figura 1, os aplicativos construídos utilizando este ambiente, fazem-no através de chamadas às primitivas do nível de "toolkit". Contudo aplicativos que demandem uma maior dependência do ambiente ou do "hardware" podem efetuar chamadas também às primitivas do nível de apoio. As primitivas do nível de apoio suprem as necessidades que são normalmente supridas por um ambiente de programação em linguagem "C". Isso deve-se ao fato de o acesso à dispositivos de entrada/saída a partir do TMS34010 poder ser feito tanto diretamente aos seus dispositivos, quanto àqueles conectados ao "hospedeiro". Assim, este ambiente provê recursos que viabilizam e flexibilizam qualquer tipo de acesso.

3 - Elementos que Compõem o Ambiente Gráfico.

O ambiente gráfico é composto por diversos elementos que podem ser utilizados por um aplicativo para utilizar os recursos da arquitetura. Tais recursos podem ser agrupados como: Gerenciador de Janelas; Gerenciador de "menus"; Subsistema de Comunicação com o Hospedeiro; Subsistema de Visualização de Imagens. Os recursos disponíveis no ambiente gráfico são descritos nas seções seguintes, e os módulos de "software" que os implementam são apresentados nos apêndices deste trabalho. Na seção 5 é apresentado um exemplo de um aplicativo que utiliza o ambiente gráfico.

3.1 - Gerenciador de Janelas.

A principal estrutura do ambiente gráfico desenvolvido é a de **Janela**. Uma ***Janela*** é uma região retangular da tela (ver figura 2), a qual é definida pelas coordenadas de seus quatro cantos. A indicação dessas coordenadas é padronizada em todo o ambiente como sendo a indicação de seu canto superior esquerdo (na figura: (x_1,y_1)) e o canto inferior direito (na figura: (x_2,y_2)).

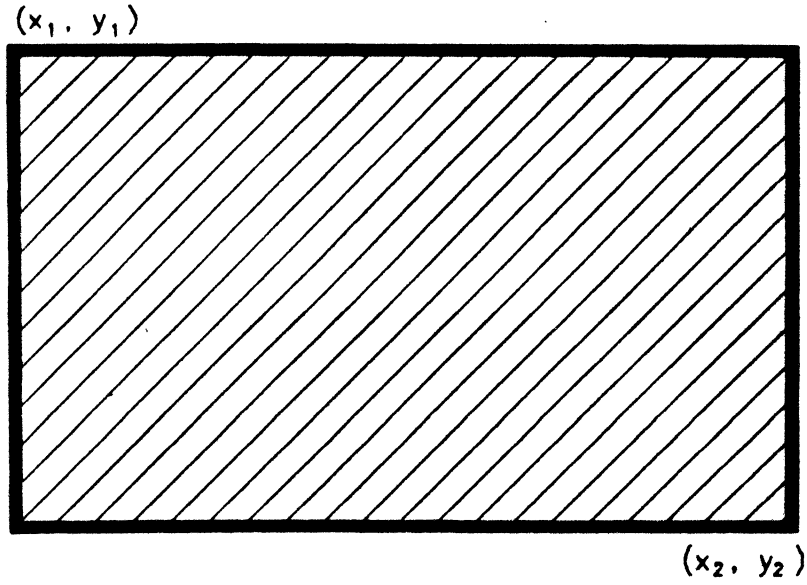


Figura 2 - Especificação de uma janela.

As janelas se associam de forma hierárquica, onde a janela principal (raiz da hierarquia de janelas), é a própria tela do monitor do sistema de visualização. A figura 3 mostra um exemplo de como poderia ficar a visualização de um sistema constituído por 9 janelas. Nesse exemplo, a tela é a janela raiz da hierarquia (R), e possui três janelas filhas: A, B e C. Por sua vez, a janela A possui três janelas filhas: A1, A2 e A3; a janela B nao possui filhas; e a janela C possui duas filhas: C1 e C2.

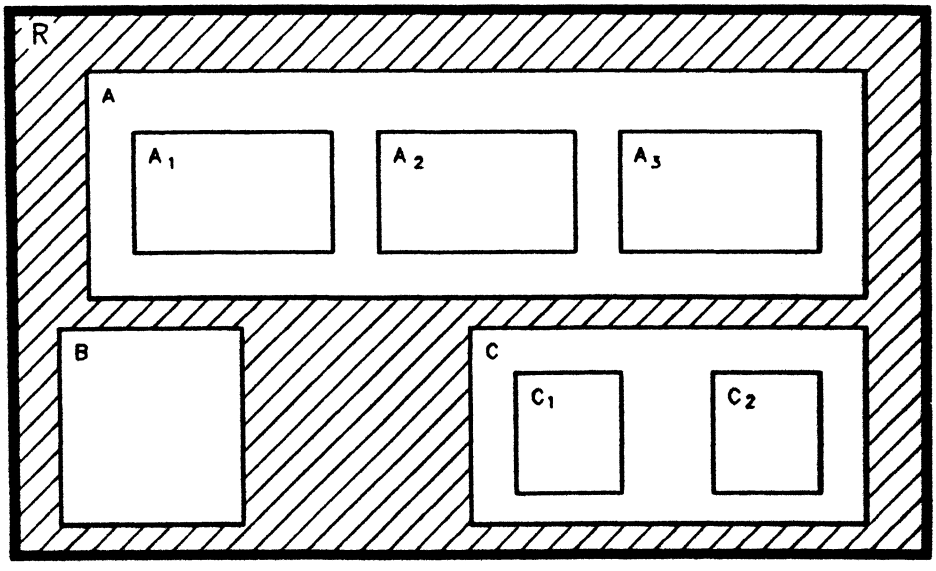


Figura 3 - Exemplo de uma tela com janelas.

A maneira como as janelas podem se associar deve respeitar algumas restrições:

- Uma janela não pode ultrapassar os limites da janela que é seu pai, ou seja, as janelas A1, A2 e A3 são inscritas na janela pai (A), assim como as janelas A, B e C são inscritas na janela raiz (tela).
- Os atributos de uma janela pai são transmitidos à filha, a menos que sejam explicitamente definidos na hora da criação. Esses atributos podem ser: tamanho e posição da janela, eventos permitidos, quais os planos de memória utilizados, cor de frente, cor de fundo, posição do cursor, arquivo de caracteres fonte.
- O sistema impõe que sempre uma janela deve ser considerada como a **janela corrente**.
- Existem operações que somente são realizadas sobre a janela corrente. Um exemplo é a criação de uma outra janela: uma nova janela somente pode ser criada como filha da janela corrente.

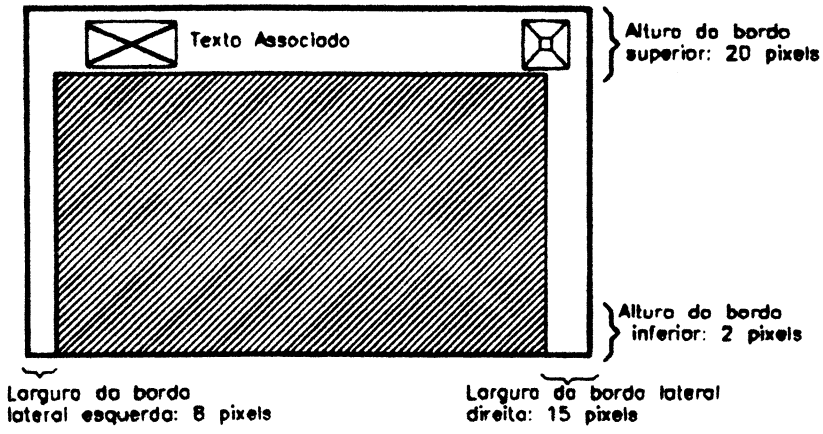
O sistema fornece dois tipos básicos de janelas: Janela com borda e Janela sem borda.

- ***Janela com Borda:*** é a janela que possui uma moldura. Esta moldura pode ser de três tipos:

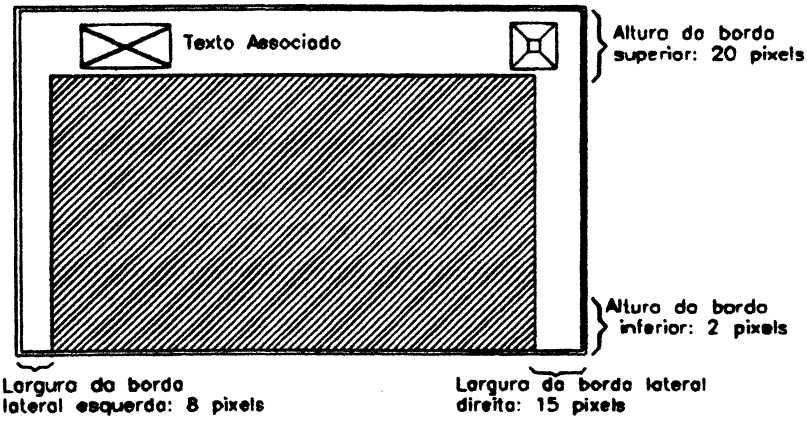
  Traço-duplo: a borda é realçada por linhas mais grossas, e admite cabeçalho e elevador de rolamento vertical.

  Traço-simples: a borda tem espessura normal e também admite cabeçalho e elevador de rolamento vertical.

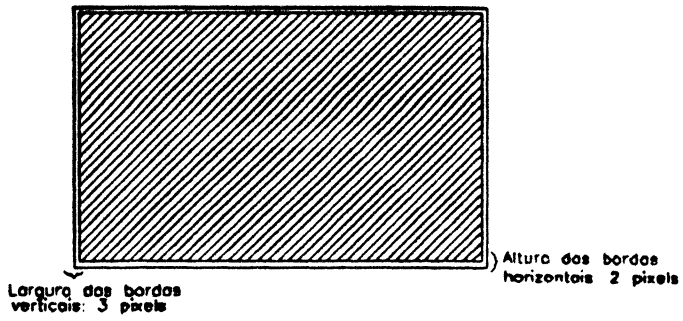
  Borda-singela: a borda tem espessura normal mas não possui cabeçalho nem elevador de rolamento. É a borda que utiliza-se em janelas que estão comportando "menus" (ver seção 6.3.2).



a) Janela com borda do tipo Traço-duplo.



b) Janela com borda do tipo Traço-simples.



c) Janela com borda do tipo Borda-singela.

Figura 4 - Apresentação dos tipos e respectiva largura de bordas de janelas.

A figura 36 mostra como é visualizada cada um dos três tipos de janela com borda, e também a largura da moldura (em pixels).

Quando o usuário solicita que seja criada uma janela com borda, na realidade o sistema cria duas janelas: a janela que contém a moldura, e uma janela filha que ocupa a região de trabalho que o usuário solicitou. Dessa forma, se um usuário solicita que seja criada uma janela com borda de traço-duplo, e que a definição da janela seja o canto superior esquerdo em (50,50) e o canto inferior direito em (100,100), na realidade a janela de trabalho será a região entre (58,70) e (85,98) por causa da reserva de espaco feita para conter a borda da janela. O controle sobre o tamanho da área efetiva de trabalho da janela deve ser feito pelo usuário, levando-se em conta a largura de cada tipo de borda. Deve-se notar que a associação entre a borda da janela, e a janela de trabalho propriamente dita no sistema de gerenciamento de janelas é que a janela de trabalho é filha da janela que contém a borda, porém esta última não é normalmente acessível ao usuário. Existe uma única primitiva que gera um janela com borda, qualquer que seja sua borda, a qual é chamada **Cria_jan_borda**.

- ***Janela sem Borda:*** este tipo de janela delimita uma região retangular da tela, a qual será utilizada como uma área de trabalho pelo aplicativo.

Janelas sem borda criam apenas a janela solicitada, utilizando toda a área solicitada pela definição da janela. Existe uma única primitiva que gera uma janela sem borda, a qual é chamada **Cria_janela**.

Como ao criar-se uma janela o Gerenciador de janelas transmite os atributos de uma janela pai à janela filha, alterando-se apenas os atributos explicitamente referenciados, as primitivas que efetuam a criação de janelas - **Cria_janela** e **Cria_jan_borda** - recebem como parâmetros pares de atributo-valor, os quais serão atribuídos à janela criada. No Manual de Referência do Ambiente Gráfico, apêndice B, são apresentados exemplos para a utilização de cada rotina.

3.2 - Gerenciador de "menus".

A utilização de "menus" para a interação com o usuário, obtendo-se dados que sejam de interesse para o software aplicativo, permitindo corrigir ou atualizar esses dados, confirmar opções e dados, ou mostrar os dados correntes numa estrutura igual à usada para a coleta de dados, a qual o usuário já está acostumado, é uma das formas de se tornar o software "amigável".

Basicamente os "menus" podem ser classificados em dois tipos [FOLEY_90]:

- "menus" estáticos: que são permanentemente visíveis. Estes "menus" são colocados uma vez na tela quando o aplicativo começa a ser executado e só são retirados quando o aplicativo termina.
- "menus" dinâmicos: tornam-se visíveis apenas quando são requisitados. Os tipos mais conhecidos, como já visto anteriormente, são os chamados "menus" tipo "pull-down" e "pop-up".

O ambiente gráfico desenvolvido conta com "menus" do tipo "pull-down", os quais podem ser colocados de forma hierárquica. As estruturas de dados, e mesmo o sistema de software já foram projetados para também suportarem "menus" do tipo "pop-up". A manutenção ou não de um "menu" na tela é opção do aplicativo. A figura 5 mostra a evolução de um processo de escolha nesse tipo de menu.

A figura 6 apresenta um "menu" do tipo "pop-up", da forma como o ambiente gráfico pode suportar.

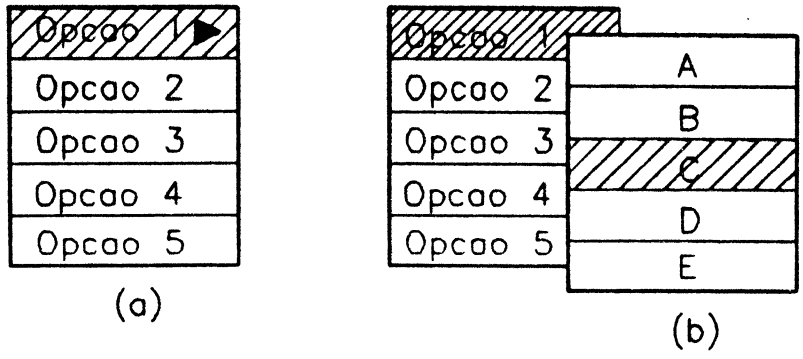


Figura 5 - "menu" "pull-down" hierárquico. (a) O primeiro "menu" mostra onde o cursor está posicionado. O cursor pode ser movido para cima ou para baixo para selecionar-se o item. (b) No "menu" hierárquico, o item que possui um sub-"menu" associado apresenta uma seta lateral, e ao andar-se com o cursor para a direita aciona-se o sub-menu.

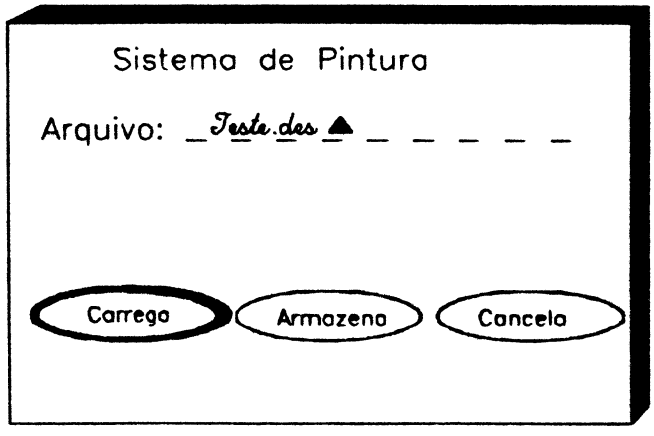


Figura 6 - "menu" "pop-up", onde há uma entrada textual e três entradas tipo botão.

O "menu" "pull-down" foi desenvolvido segundo a filosofia do ambiente gráfico **Environ V** da Intergraph [INT_88a], ampliando-se apenas a forma como as opções podem ser mostradas. No Environ V, todas as opções de um "menu" são mostradas simultâneamente, e o controle das dimensões do "menu" é feito pelo usuário. No ambiente gráfico desenvolvido, o "menu" "pull_down" não precisa mostrar simultâneamente todos os ítens, permitindo especificar na criação do "menu" quantos ítens serão mostrados ao

mesmo tempo, e os demais ítens serão mostrados conforme o usuário solicitar, através do rolamento do menu.

Quando um "menu" é criado, o ambiente gráfico coloca-o sobre uma janela (janela com borda, do tipo borda-singela), e o Gerenciador de Janelas passa também a ter controle sobre o menu. Isso é feito para simplificar a forma de gerenciar "menus", utilizando-se os recursos já descritos para o gerenciamento de janelas.

3.3 - Subsistema de Comunicação com o Hospedeiro.

A arquitetura dedicada baseada no processador gráfico TMS34010 (GSP), utiliza o microcomputador IBM-PC como hospedeiro. A capacidade de transferir grandes blocos de dados para o GSP torna a interface com o hospedeiro bastante ágil. Por exemplo, o hospedeiro pode transferir blocos de comandos para o GSP, pode parar o GSP temporariamente para carregar um novo programa para o GSP executar, ou mesmo pode ler blocos de dados gráficos gerados pelo GSP.

A interface de comunicação com o hospedeiro, que pode ser acessada tanto pelo hospedeiro quanto pelo GSP, ocupa os seguintes registradores que são mapeados ou em 4 regiões consecutivas de 16 bits no espaço de endereço de memória do GSP, ou em registradores de E/S [TEXAS_86].

HSTCTL - Este registrador controla funções tais como: a transferência de pedidos de interrupção e o código de estado em 3 bits entre o hospedeiro e o TMS34010. Estas solicitações são tipicamente utilizadas pelo software para coordenar a transferência de grandes blocos de dados através da memória local do GSP. Outras funções do HSTCTL são: permitir que o hospedeiro imponha a descarga da "cache" de instrução do GSP, parar a execução do GSP, e transmitir solicitação de interrupção não mascarável ao GSP.

HSTADRH e **HSTADRL** - Estes registradores formam um único registrador, que contém um endereço de 32 bits o qual aponta para uma palavra de memória corrente a ser acessada.

HSTDATA - Este registrador armazena o dado transferido de/para a memória local do GSP sob o controle do processador hospedeiro. A interface com o hospedeiro pode ser programada para automaticamente incrementar o endereço após cada transferência, e dessa forma o hospedeiro pode obter um

acesso rápido a um bloco de dados armazenado em posições consecutivas de memória.

A figura 7 ilustra o procedimento para a leitura de um bloco de palavras (de 16 bits) na memória local (do GSP), cujo endereço inicial é *N*. Deve-se assumir que está ativado o autoincremento do endereço indicado após o acesso ao dado. Em (a), o hospedeiro armazena os 32 bits do endereço *N* nos registradores HSTADRH (16 bits) e HSTADRL (16 bits). A operação de se carregar a segunda parte do endereço em HSTADRH, faz com que a interface do GSP com o hospedeiro automaticamente inicie um ciclo de leitura à memória local (do GSP). O ciclo de leitura, mostrado em (b), transfere o conteúdo do endereço de memória *N* para o registrador HSTDATA. Em (c), o processador hospedeiro lê o registrador HSTDATA, buscando o dado previamente lido do endereço *N*. A leitura do HSTDATA pelo processador hospedeiro, faz com que o GSP incremente de forma automática o conteúdo do par HSTADRL e HSTADRH de 16, tal como é mostrado em (d). O conteúdo do novo endereço é lido em HSTDATA, como é mostrado em (e). Este dado estará disponível em HSTDATA na próxima leitura feita pelo processador hospedeiro. O processo mostrado nos ítens (c), (d), e (e), repete-se para cada palavra lida da memória local do GSP. Um procedimento análogo é feito para o procedimento de escrita, onde o bit INCW do HSTCTL é colocado em 1.

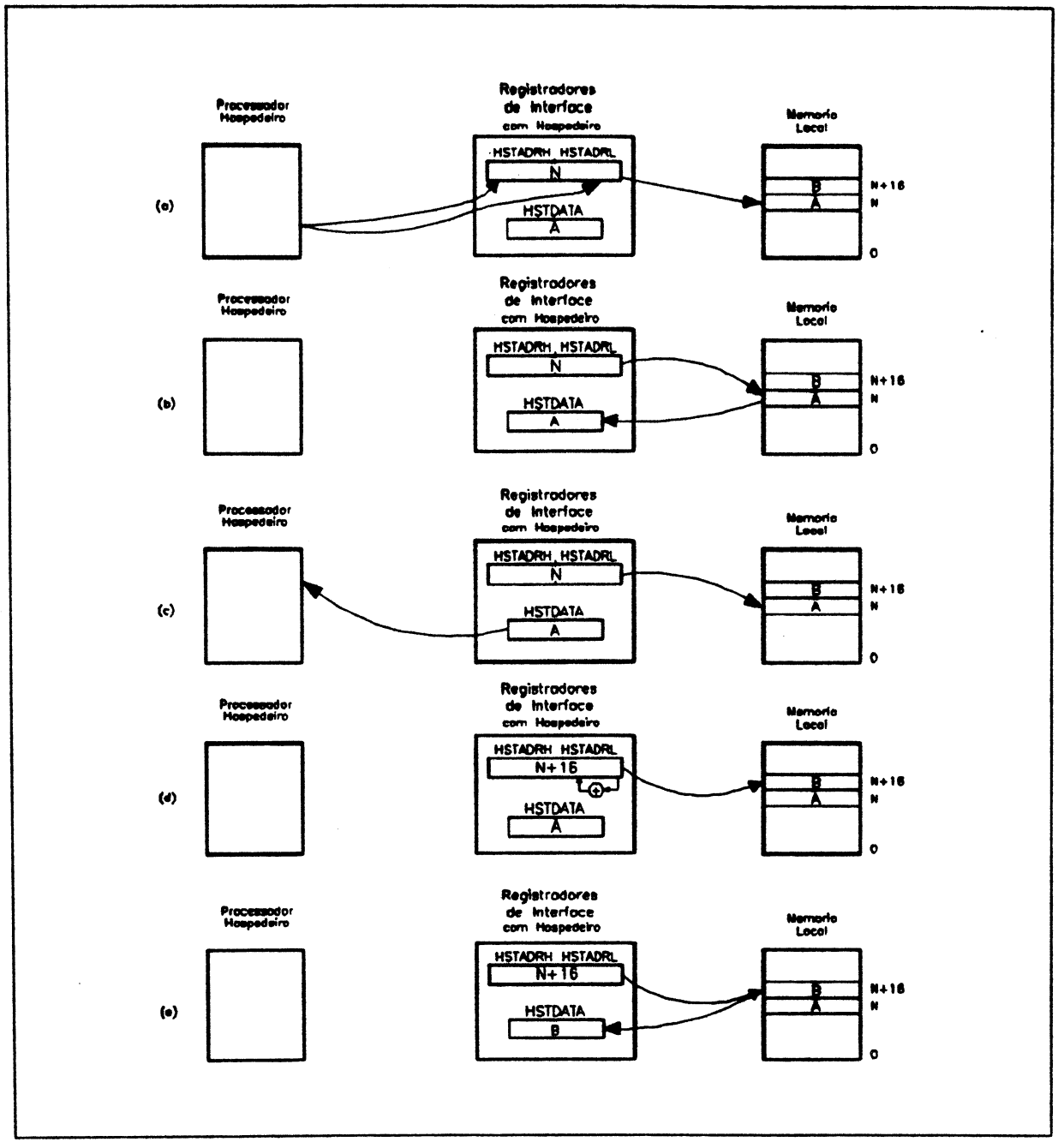


Figura 7 - Leitura indireta da memória local do GSP pelo processador hospedeiro.

O TMS34010 possui entre as ferramentas para desenvolvimento de software um carregador (loader) chamado **SDBL**, o qual verifica se o GSP está pronto, e em caso positivo carrega um programa executável na memória local do GSP. Isso é feito através dos registradores da interface com o hospedeiro vistos acima.

Através do SDBL, carrega-se um programa no GSP e não se tem mais qualquer comunicação com o hospedeiro. Ou seja, os recursos disponíveis apenas no hospedeiro, tais como: teclado e disco, não ficam disponíveis ao programa de aplicação. Contudo, um

programa aplicativo com alto grau de interatividade e que além do mais trate de imagens (estas armazenadas em disco) precisa utilizar tais recursos. É necessário também que as operações a serem realizadas o façam com a maior eficiência possível. Assim, foi utilizada a técnica de Acesso Direto à Memória (DMA) do GSP sempre que possível, como é o caso das transferências de blocos de dados entre o GSP e o disco do hospedeiro. Por outro lado, quando os dados devem ser transferidos byte a byte, tal como é o caso de transferências de caracteres digitados no teclado, ou mostrados no vídeo do hospedeiro, a técnica utilizada foi a de "hand-shake", levando-se em conta que o processamento em paralelo da transferência e do aplicativo em execução no GSP fica inevitavelmente comprometido devido à necessidade do aplicativo depender dos dados transferidos ou da confirmação da transferência.

Dessa forma foi feita uma extensão ao SDBL, criando-se uma nova ferramenta a que chamou-se ***SDBLR*** (SDBL com suporte de comunicação Residente), a qual após carregar o programa objeto na memória local do GSP, permanece residente no hospedeiro, monitorando um "buffer" de comunicação colocado entre o GSP e o hospedeiro. Este "buffer" possui 20 bytes, pois dessa forma acomoda perfeitamente a estrutura de dados LIE_comunica (ver seção 4.5), a qual contém os campos de controle para os dispositivos do hospedeiro. Na figura 8 é ilustrado o esquema de comunicação entre o GSP e o hospedeiro através do "buffer" de comunicação. A figura 9 apresenta o diagrama de blocos da ferramenta SDBLR.

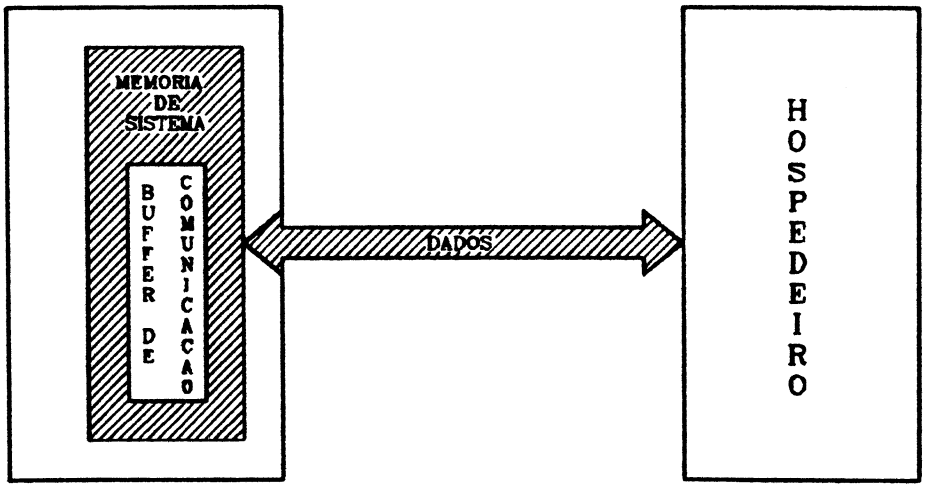


Figura 8 - Esquema de comunicação entre o GSP e o hospedeiro.

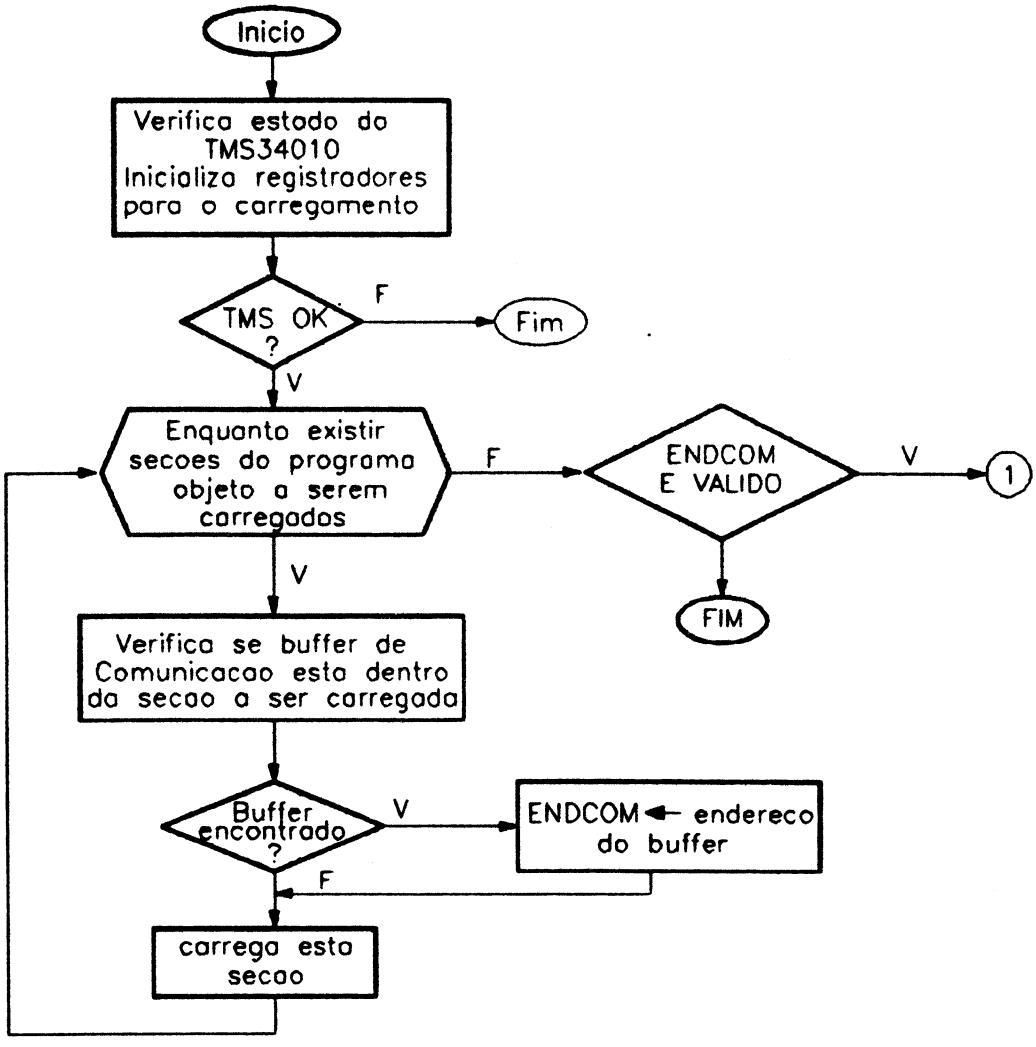


Figura 9 - Diagrama de blocos do carregador SDBLR.

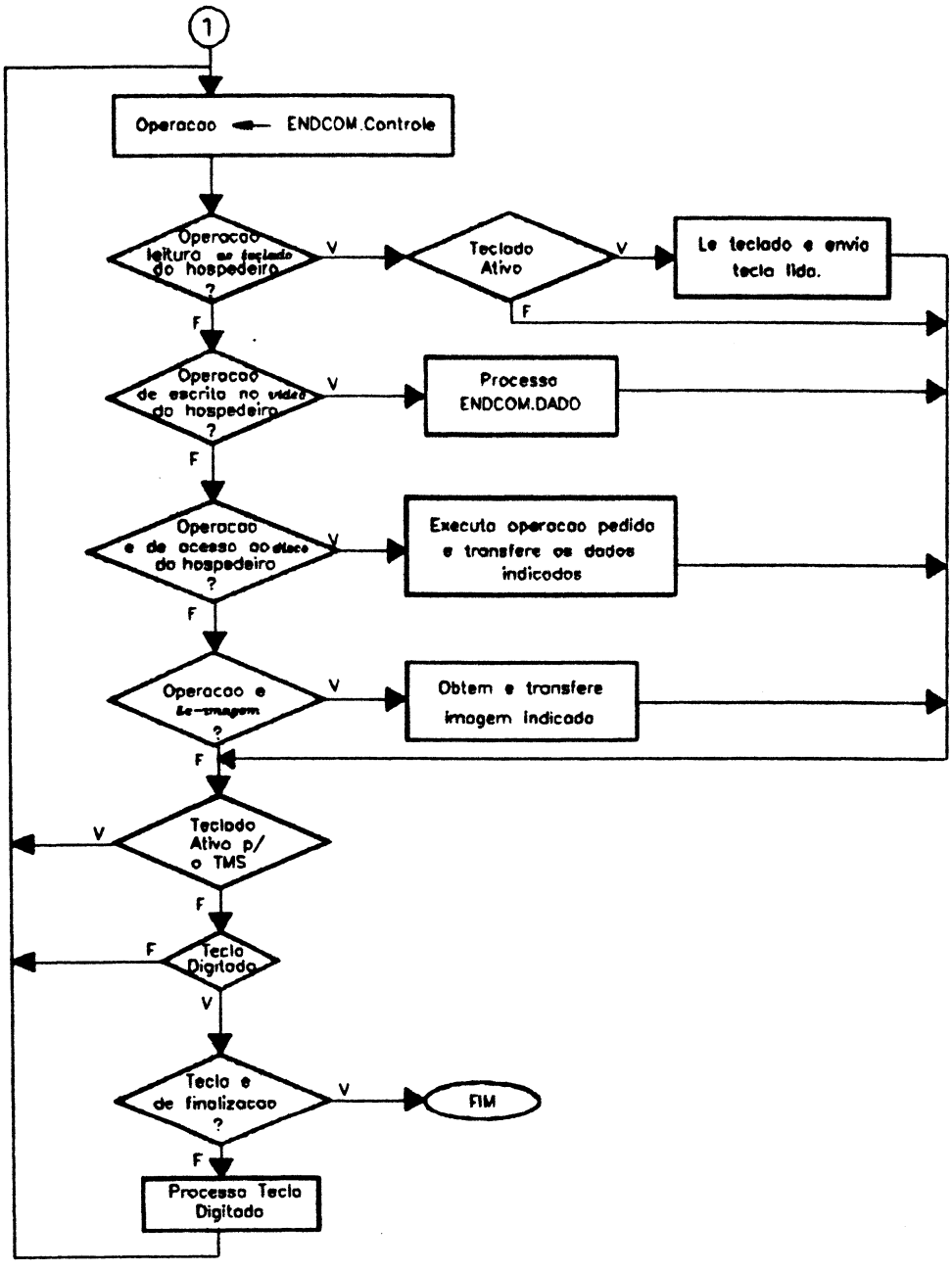


Figura 9 - Diagrama de blocos do carregador SDBLR - Continuação.

Além disso foi necessário desenvolver um subsistema de comunicação com o hospedeiro, o qual gerenciaria todas as solicitações de operações a disco e E/S executadas nos periféricos do hospedeiro, transmitindo e recebendo os dados envolvidos.

Os recursos do hospedeiro gerenciados são:

- teclado,

- vídeo,
- disco.

A transferência dos dados entre a arquitetura gráfica baseada no TMS34010 e o Hospedeiro é feita utilizando-se de 20 bytes, ao qual é associada uma instância da estrutura LIE_comunica (seção 4.5), que possibilita o controle pelo SDBLR de qual recurso do hospedeiro o aplicativo em execução no GSP está solicitando, e também da própria transferência dos dados. A forma de transferência dos dados é através de acesso direto a memória (DMA) do GSP.

O ambiente de interação com o GSP implementado no hospedeiro IBM-PC permite ao usuário controlar e monitorar o andamento da execução de um aplicativo no GSP que utilize recursos do hospedeiro. Para a criação do ambiente no IBM-PC utilizou-se o núcleo gráfico desenvolvido no Departamento de Computação e Estatística - ICMSC - USP, que faz parte do Sistema GEO [TRAINA_90].

Utilizando-se esse núcleo, a tela do hospedeiro foi dividida em três janelas: uma janela na parte superior que recebe os dados enviados pelo TMS para serem mostrados na tela do hospedeiro; uma janela na parte inferior que mostra os dados que o hospedeiro está transmitindo para o TMS, e uma janela pequena no centro que indica se o teclado está transmitindo para o TMS (ON), ou se está a serviço do próprio hospedeiro (LOCAL). A figura 10 mostra como a tela do hospedeiro é organizada pelo subsistema de Comunicação.

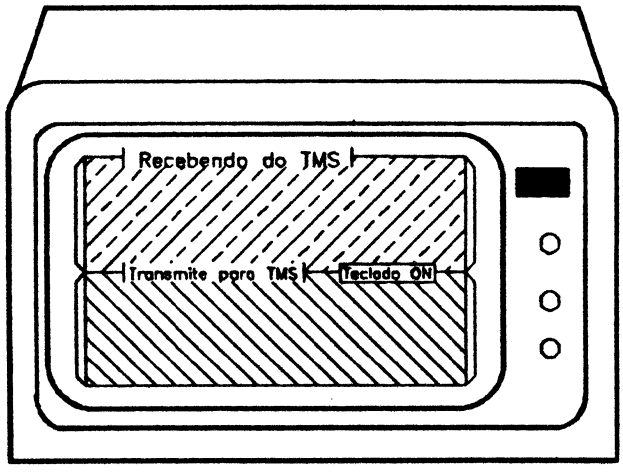


Figura 10 - Organização da tela do monitor do hospedeiro, quando o SDBLR está em execução.

Como foi visto na seção 2, no nível de Apoio do ambiente gráfico, estão as primitivas que compõem o subsistema de Comunicação com o hospedeiro. Tais primitivas estão implementadas como rotinas que possuem nomes parecidos com as da biblioteca padrão da linguagem "C", que efetuam a mesma função. Por exemplo: as funções **i_printk** que efetua uma saída formatada na tela do hospedeiro, e **v_printk** que o faz na tela do TMS, de forma equivalente à **printf** da linguagem "C". Da mesma forma foram construídas outras funções equivalentes as da linguagem "C". A tabela 8 apresenta a correspondência entre as rotinas do nível de Apoio, e as da biblioteca da linguagem "C" que possuem a mesma função.

Função Realizada	Biblioteca Padrão "C"	No GSP	No Hospedeiro
lê caracter no teclado	getch()	-	i_getc()
escreve caracter	putch (caracter)	v_putc (caracter)	i_putc (caracter)
realiza saída formatada	printf (fmt,...)	v_printk (fmt, ...)	i_printk (fmt, ...)
lê cadeia de caracteres	gets (*string)	-	i_getk (*cadeia)
escreve cadeia de caracteres	puts (string)	v_puts (string) v_putk (cadeia)	i_puts (string) i_putk (cadeia)
Abre arquivo	open (arquivo, modo)	-	i_open (arquivo, modo)
Fecha arquivo	close (Narquivo)	-	i_close (Narquivo)
Lê dado de arquivo	read (Narq,buffer,tam)	-	i_read (Narq, buffer, tam)
Escreve dados no arquivo	write (Narq, buffer, tam)	-	i_write (Narq,buffer,tam)
Posiciona no arquivo	seek (Narq, offset, como)	-	i_seek (Narq, offset, como)

Tabela 8 - Rotinas para E/S pertencentes ao nível de Apoio do ambiente gráfico, e sua corresponência às da Linguagem "C".

O subsistema de Comunicação pode ser dividido em duas partes:

1. Um conjunto de rotinas que são executadas no hospedeiro, e realizam as funções de E/S normalmente realizadas pela biblioteca da linguagem "C";
2. Um outro conjunto de rotinas que executam sincronizadas com aquelas em execução no hospedeiro, e complementam as operações necessárias à comunicação.

A tabela 9 ilustra como os pares de funções do subsistema de comunicação operam para a transferência de dados. Mostra a equivalência das funções do GSP com as do hospedeiro.

Função Realizada	No GSP	No Hospedeiro
Obtém um caracter do hospedeiro	i_getc	tms_sendc
Envia um caracter para o hospedeiro	i_putc	tms_receivec
Verifica existência de caracter transmitido pelo hospedeiro	i_pollc	tms_send_poll
Verifica existência de caracter transmitido pelo GSP	-	tms_receive_poll
Inicializa busca de arquivos, posiciona no primeiro	i_arqinicio	tms_disco_ARQINICIO
Prossegue a busca de arquivos, posiciona no seguinte	i_arqproximo	tms_disco_ARQPROXIMO
Abre arquivo indicado no hospedeiro	i_open	tms_disco_ABREARQ
Fecha arquivo indicado no hospedeiro	i_close	tms_disco_FECHARQ
Lê dados indicados do arquivo do hospedeiro	i_read	tms_disco_LEARQ
Escreve dados indicados no arquivo do hospedeiro	i_write	tms_disco_ESCARQ
Posiciona ponteiro no arquivo do hospodeiro	i_seek	tms_disco_POSICIONARQ

Tabela 9 - Equivalência das funções de E/S (GSP ↔ Hospedeiro).

3.4 - Subsistema de Transferência de Imagens Tomográficas de RM.

O objetivo principal da construção do ambiente gráfico sobre o GSP foi para possibilitar a transferência de imagens coletadas pelo sistema de tomografia do IFQSC para a arquitetura dedicada a visualização de imagens, permitindo ao usuário efetuar manipulações sobre a imagem.

A operação de transferência das imagens foi colocada em um subsistema a parte, apesar de necessitar de comunicação entre o hospedeiro e o GSP, foi necessário fazer-se um tratamento diferenciado sobre os dados que compõem a arquivo de imagem.

O arquivo de imagens gerado pelo sistema de tomografia contém os dados formatados, pelo fato de facilmente compatibilizá-lo com todos os módulos que constituem o sistema.

A primitiva que efetua a operação de leitura de uma imagem no disco do hospedeiro, e a transfere para o GSP foi chamada de **i_le_imagem**. Esta primitiva abre o arquivo de imagem no hospedeiro e o transfere para o GSP. O GSP cria uma janela com borda que comporte exatamente a imagem, e coloca a imagem na janela para que o usuário possa manipulá-la. Na seção 5 o exemplo apresentado ilustra como a primitiva **i_le_imagem** opera.

4 - Descrição das Estruturas de Dados Utilizadas.

Nessa seção é feita uma descrição das estruturas de dados utilizadas na implementação do ambiente gráfico. O apêndice D lista os arquivos de "headers" do sistema, onde as estruturas estão definidas.

4.1 - Estrutura de Dados para o Gerenciamento de Janelas - LIE_janela.

Esta estrutura define o nó básico da estrutura de dados que é disposta hierarquicamente em forma de árvore, indicando como as janelas do ambiente gráfico estão relacionadas. A estrutura LIE_janela contém as informações principais sobre uma janela, e nela também estão dispostas outras estruturas de dados (esquematizadas como listas), que armazenam informações tidas como secundárias. Cada nó da estrutura é alocado dinamicamente na memória de sistema do GSP. A figura 11 apresenta a estrutura de dados LIE_janela. Na figura 12 é ilustrado como ficaria a hierarquia de janelas para a figura 3.

O sistema mantém em uma variável global ***lie_jan_corrente*** qual é a janela corrente que o sistema está tratando. Isto é feito porque as operações de janelas são na sua maioria definidas sobre a janela corrente.

A estrutura LIE_janela faz parte do Nível Básico do ambiente gráfico, bem como as rotinas que manipulam a hierarquia em forma de árvore.

O significado de cada um dos campos da estrutura LIE_janela é colocado a seguir, devendo-se notar que os primeiros quatro campos da estrutura armazenam os ponteiros que constrõem a árvore de janelas.

PAI - (32 bits) armazena o ponteiro para o nó LIE_janela que armazena as informações da janela pai desta janela.

PRIM_FILHO - (32 bits) armazena o ponteiro para o nó LIE_janela que armazena as informações da janela que é o primeiro filho desta janela. A partir deste ponteiro pode-se recuperar a lista de filhos de uma dada janela. Sempre que é criado um filho de uma janela, o nó que especifica este filho é colocado na cabeça da lista.

PROX_IRMAO - (32 bits) armazena o ponteiro para o nó LIE_janela que armazena as informações da janela que é o próximo irmão desta janela na lista.

IRMÃO_ANTERIOR - (32 bits) armazena o ponteiro para o nó LIE_janela que armazena as informações da janela que é o irmão anterior desta janela na lista.

TIPO - (32 bits) armazena o ponteiro para a estrutura LIE_tipo_janela que armazena as informações sobre a aparência dessa janela.

SENTINELA - (32 bits) campo utilizado para consistência da estrutura de dados. Na criação de um nó dessa estrutura é atribuído o valor 1234 a esse campo, e sempre que se recuperar um nó, este campo é testado.

XLO - (32 bits) armazena a posição no eixo X do canto superior esquerdo da janela.

YLO - (32 bits) armazena a posição do eixo Y do canto superior esquerdo da janela.

XHI - (32 bits) armazena a posição no eixo X do canto inferior direito da janela.

YHI - (32 bits) armazena a posição no eixo Y do canto inferior direito da janela.

PLANE_MASK - (16 bits) máscara que especifica quais os planos que estão ativos na memória. Para o sistema atual que conta com 8 planos de memória, se todos os planos estiverem ativos, em PLANE_MASK deve estar com o valor 0000H.

EVENT_MASK - (16 bits) máscara que especifica quais os eventos permitidos para essa janela.

COR_FRENTE - (32 bits) armazena qual é a cor de frente para esta janela.

COR_FUNDO - (32 bits) armazena qual é a cor de fundo para esta janela.

X_CURSOR - (32 bits) armazena a posição corrente do mouse no eixo X.

Y_CURSOR - (32 bits) armazena a posição corrente do mouse no eixo Y.

ARQ_FONTE - (32 bits) armazena o ponteiro para o arquivo de fontes que essa janela pode acessar.

BASE - (32 bits) armazena o ponteiro para a região da memória do sistema que armazena a região da tela onde a imagem foi mapeada. É usado para a operação de refrescamento da tela do GSP.

RESERVADO 1 - (32 bits) campo reservado para futuras ampliações.

RESERVADO 2 - (32 Bits) campo reservado para futuras ampliações.

ESTADO - (16 bits) especifica se a janela está mapeada na tela (bit 0 é 1), ou se a janela presentemente não está mapeada (bit 0 é 0).

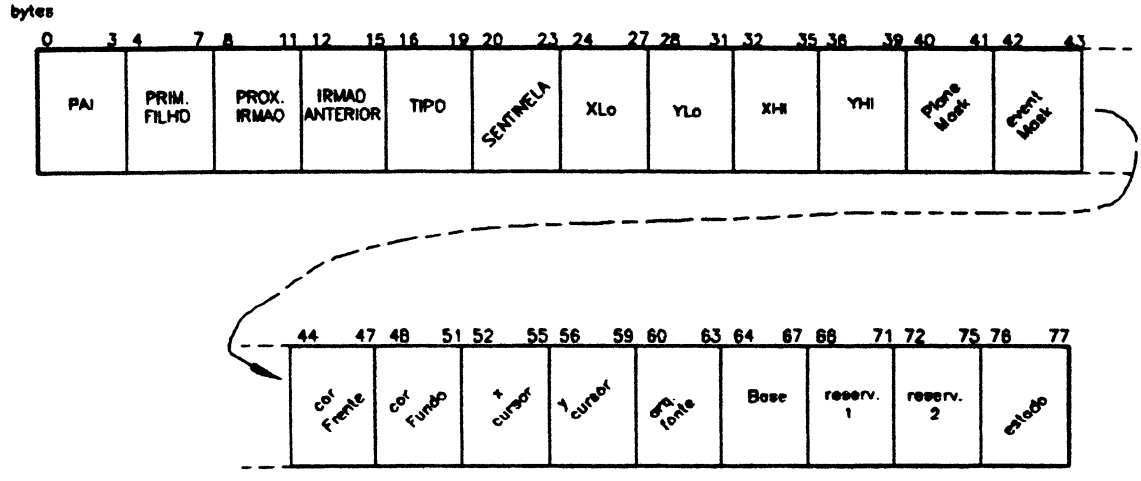


Figura 11 - Estrutura de dados LIE_janela.

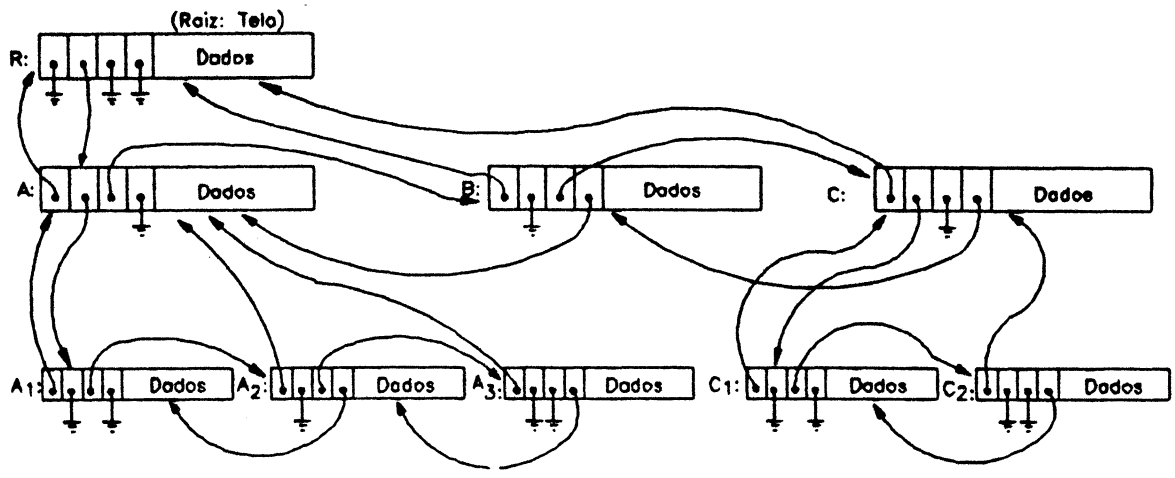


Figura 12 - Visão da hierarquia de janelas da figura 3.

4.2 - Estrutura de Dados LIE_tipo_janela.

Nesta estrutura são armazenadas as especificações de aparência de uma janela. Os nós dessa estrutura também são alocados dinamicamente na memória de sistema do GSP, e acoplados à estrutura principal LIE_janela. A figura 13 abaixo mostra a forma dessa estrutura, e a explicação de seus campos é colocada a seguir.

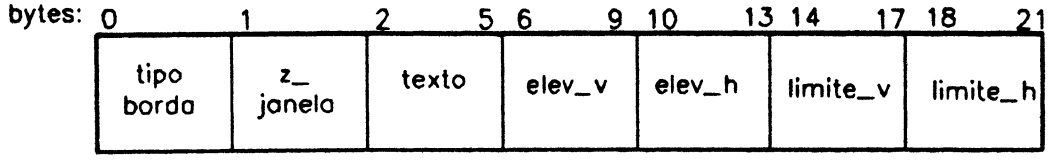


Figura 13 - Estrutura de dados LIE_tipo_janela.

tipo_borda - (8 bits) especifica qual o tipo da borda:

bit 0 : 0 → borda sem traço.
1 → borda com traço.

bit 1 : 0 → borda com 1 traço.
1 → borda com 2 traços.

bit 2 : 0 → sem evelador horizontal.
1 → com elevador horizontal.

bit 3 : 0 → sem elevador vertical.
1 → com elevador vertical.

bit 4 : 0 → janela aberta.

1 → janela fechada.

bit 5 : 0 → janela com borda ampla.

1 → janela com borda singela (usada em "menus")

bit 6-8: Reservados para futuras ampliações.

z_janela - (8 bits) índice do vetor **lie_prof_janela** que indica qual é a profundidade da janela atual com relação às demais janelas. Será utilizado para tratamento de eventos de sobreposição de janelas.

texto - (32 bits) ponteiro para a cadeia de caracteres (nome) associada a janela.

elev_v - (32 bits) armazena a posição do elevador vertical em relação ao eixo Y.

elev_h - (32 bits) armazena a posição do elevador horizontal em relação ao eixo X.

limite_v - (32 bits) armazena a posição máxima que o elevador vertical pode assumir na janela.

limite_h - (32 bits) armazena a posição máxima que o elevador horizontal pode assumir na janela.

4.3 - Estrutura de Dados LIE_fonte.

Nessa estrutura de dados é armazenada a especificação dos fontes de caracteres que o ambiente gráfico suporta. O ambiente gráfico trabalha com uma variável global, ***lie_fonte_corrente***, que especifica qual é o fonte correntemente utilizado pela janela.

A figura 14 apresenta a estrutura LIE_fonte.

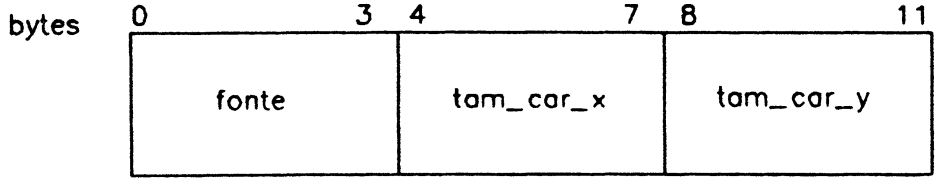


Figura 14 - Estrutura de Dados LIE_fonte.

Os campos da estrutura LIE_fonte significam:

fonte - (32 bits) armazena o endereço do arquivo que contém o mapa de bits para cada caracter.

tam_car_x - (32 bits) armazena qual a largura do caracter em pixels de tela.

tam_car_y - (32 bits) armazena qual a altura do caracter em pixels de tela.

4.4 - Estrutura de Dados lie_prof_janela.

A estrutura lie_prof_janela é um vetor de MAXJANELAS (número máximo de janelas que o ambiente gráfico permite por aplicativo) elementos, onde cada elemento é um ponteiro (32 bits) para uma lista de janelas (lie_janela). O índice do vetor indica qual a lista de janelas que sobrepõem outra. Assim no índice 0 (zero) do vetor encontra-se a lista de janelas que sobrepõem todas as outras (as que estão em cima), e no índice MAXJANELAS-1 as janelas que todas sobrepõem (as estão embaixo).

A figura 15 apresenta o vetor lie_prof_janela.

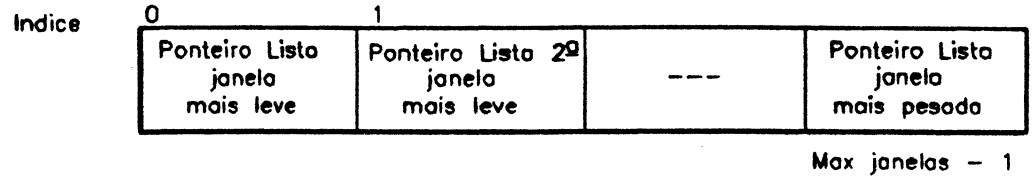


Figura 15 - Estrutura de dados lie_prof_janela.

4.5 - Estrutura de Dados LIE_comunica.

Essa estrutura de dados é utilizada pelo gerenciador de comunicação. Os seus campos, que serão explicados abaixo, são acoplados ao "buffer" de comunicação que é mapeado na memória de sistema do GSP. A figura 16 ilustra esta estrutura.

controle_teclado - (16 bits) especifica o controle do teclado do hospedeiro. Se estiver com 0 (DADO_ACEITO) o hospedeiro não enviou caracter, ou o caracter enviado já foi aceito. Se estiver com 1 (DADO_PRONTO), o GSP já pode tratar o dado enviado pelo hospedeiro.

dado_teclado - (16 bits) armazena o dado enviado pelo hospedeiro.

controle_vídeo - (16 bits) especifica o controle do vídeo do hospedeiro. Se estiver com 0 (DADO_ACEITO) o hospedeiro já escreveu o caracter, ou nenhum caracter foi enviado. Se estiver com 1 (DADO_PRONTO), o GSP já enviou um caracter, mas o hospedeiro ainda não o processou.

dado_vídeo - (16 bits) armazena o dado enviado pelo GSP.

controle_disco - (16 bits) especifica qual a operação em disco solicitada pelo GSP. Pode conter os seguintes valores:

DADO_ACEITO (0) : Não existe nenhuma operação pendente.

ARQINICIO (1) : buscar a primeira entrada no diretório do disco do hospedeiro que aceita a especificação fornecida.

ARQPROXIMO (2) : buscar a próxima entrada no diretório do disco do hospedeiro que aceita a especificação fornecida anteriormente por uma chamada de ARQINICIO.

LEARQ (3) : ler arquivo especificado.

ESCARQ (4) : escrever no arquivo especificado.

ABREARQ (5) : abrir o arquivo especificado.

FECHARQ (6) : fechar o arquivo especificado.

POSICIONARQ (7) : posicionar em um endereço dentro do arquivo especificado.

LEIMAGEM (8) : abre um arquivo de imagens no hospedeiro e o transfere para o GSP. É feito o processamento de conversão dos dados e a imagem é mostrada em uma janela no GSP.

CONTINUA (9) : prepara o controle para a próxima operação em disco.

dado_disco - (16 bits) passa um dos parâmetros de uma operação em disco cujo significado depende da operação em andamento.

endereço_disco - (32 bits) passa um dos parâmetros de uma operação em disco cujo significado depende da operação em andamento.

auxiliar_disco - (32 bits) passa um dos parâmetros de uma operação em disco cujo significado depende da operação em andamento.

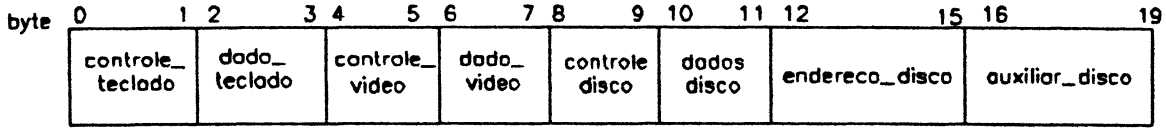


Figura 16 - Estrutura de dados LIE_comunica.

4.6 - Estrutura de Dados ArqSpec.

Esta estrutura de dados é utilizada para manipulações com arquivos. Armazena as informações do cabeçalho de um arquivo que são de utilidade para o usuário. Essas informações estão colocadas no campos descritos a seguir. A figura 17 mostra a estrutura ArqSpec.

fattr - (16 bits) indica o estado ou o tipo do arquivo.

bit 0: Arquivo R/O.

bit 1: Arquivo Oculto (Hidden File).

bit 2: Arquivo do Sistema.

bit 3: Nome do disco.

bit 4: Arquivo de subdiretório.

ftime - (16 bits) indica a hora de criação ou última modificação do arquivo. Está codificada segundo o padrão de hora do sistema DOS.

fdate - (16 bits) indica a data de criação ou última modificação do arquivo. Está codificada segundo o padrão de data do sistema DOS.

fsize - (32 bits) indica o tamanho do arquivo em bytes.

fname - (16 bytes) armazena o nome e extensão do arquivo.

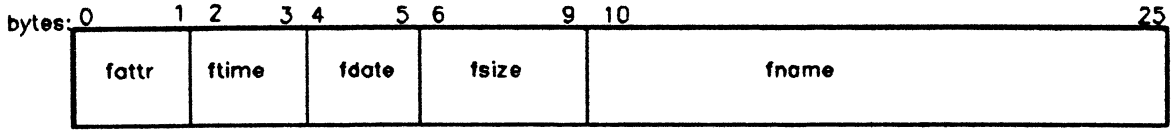


Figura 17 - Estrutura de dados ArqSpec.

4.7 - Estruturas de Dados pdmenu e pdmdef.

Esta estrutura de dados armazena as informações necessárias para a construção de "menus". Foi construída para ser utilizada tanto por "menus" do tipo "pull-down" quanto "pop-up".

A figura 18 apresenta a estrutura pdmenu, a qual é utilizada como item básico para a construção do vetor **pdmdef** que armazena as informações para todos os "menus" utilizados por um aplicativo.

Uma estrutura pdmenu armazena os ponteiros para os vetores que contém as informações. Os vetores são passados pelos aplicativos, mas não são copiados em estruturas internas, para economia de memória. Isso significa que uma vez criados os "menus", os aplicativos devem evitar alterações nesses vetores, uma vez que são usados sempre que um "menu" é ativado. A seguir é feita a descrição dos campos da estrutura pdmenu:

cor - (4 x 32 bits) vetor de 4 posições que armazena as cores com que devem ser mostrados os seguintes elementos do menu:

cor[0] = armazena cor de fundo;

cor[1] = armazena cor da borda;

cor[2] = armazena cor dos textos regulares;

cor[3] = armazena cor do texto em destaque.

nro_entries - (32 bits) armazena o número máximo de entradas que o "menu" possui.

scr_size - (32 bits) armazena o número de entradas que são mostradas simultaneamente na tela (pois o "menu" pode sofrer rolamento ("scrool")).

entry - (32 bits) armazena o ponteiro para um vetor de cadeias. Cada cadeia deve conter o texto de uma entrada do menu.

cad - (32 bits) armazena o ponteiro para um vetor de cadeias alternativas. É usado apenas para "menus" "pop-up".

command - (32 bits) armazena o ponteiro para um vetor de inteiros, o qual indica para cada entrada do "menu" qual é o valor de retorno associado, que deve retornar se aquela entrada for a escolhida.

link - (32 bits) armazena o ponteiro para um vetor de inteiros que indica para cada entrada do menu, se existe um sub"menu" associado, e se existir qual é ele.

retorno - (32 bits) armazena o ponteiro para um vetor de inteiros que indica o valor de retorno associado a alguma entrada. É usado apenas para "menus" "pop-up".

poslin - (32 bits) armazena o ponteiro para um vetor de inteiros. É usado apenas em "menus" "pop-up" para, juntamente com o vetor **poscol** definir a posição de cada peça no "lay-out" do menu.

poscol - (32 bits) armazena o ponteiro para um vetor de inteiros. É usado apenas em "menus" "pop-up" para, juntamente com o vetor **poslin** definir a posição de cada peça no "lay-out" do menu.

cursor - (32 bits) indica qual a entrada que o cursor estava quando o "menu" foi encerrado na última vez em que foi chamado. Não pode ser usado pelos aplicativos.

inipos - (32 bits) indica qual a entrada que ocupava o topo do "menu" quando este foi encerrado pela última vez em que foi chamado. Não pode ser usado pelos aplicativos.

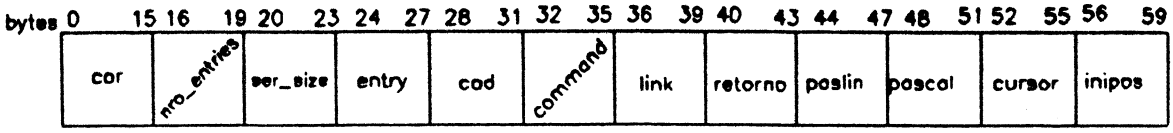


Figura 18 - Estrutura de dados pdmenu.

5 - Um Exemplo.

Nessa seção é apresentado um exemplo de um aplicativo construído utilizando esse ambiente. Nesse exemplo apresenta-se um "menu" "pull-down" principal com 5 opções:

LE IMAGEM - Caso essa opção seja a escolhida, é montado um sub"menu" "pull-down" com rolamento, em cujas entradas são listados os arquivos contidos no diretório corrente que possuem extensão PIC. Esses arquivos, por convenção, armazenam imagens. Ao usuário escolher uma imagem, o arquivo correspondente é aberto no hospedeiro e transmitido ao GSP. No GSP a imagem é colocada dentro de uma janela com borda, e o "menu" de opções volta a estar ativo.

MAXIMIZA HISTOGRAMA - Ao escolher essa opção, é efetuada uma operação de "alargamento da imagem". Ou seja é realizada uma redistribuição dos níveis de cinza da imagem permitindo-se uma melhor visualização. Este tipo de operação é comumente feito através de manipulação interativa do histograma da imagem. Devido a restrições de espaço de memória dessa versão, foi implementado apenas a operação dessa forma, sem interação com o usuário. A saturação é a máxima permitida, levando-se o nível mais alto da imagem ao nível mais alto permitido (nessa arquitetura é de 255), e o nível mais baixo em zero. Todos os demais níveis da imagem acompanham o deslocamento imposto pelos pontos limites.

APAGA - Apaga a janela com a cor de fundo especificada para essa janela.

SAI DE CIMA - Retira o "menu" da tela para poder-se visualizar inteiramente a tela. Ao teclar-se qualquer tecla o "menu" retorna a tela.

FIM - Finaliza o programa

Listagem do Programa Exemplo

```
/*****************************************************************************c
 * Programa para testar a comunicacao entre o PC (host) e o TMS
 *
 * E' usado a tecnica de "hand-shake".
 *
 *  Testa a visualizacao de uma imagem a partir de um arquivo no host .
 *
 *****************************************************************************/

#include "data_str.h"
#include "lie_sgj.h"
#include "i_disco.h"

#define MAXARQIMAG  20   /* Este programa pode tratar ate 20 arq de imagem */
#define TAMNP       5    /* Tamanho do menu principal   */

/*------------  Variaveis Globais ------------------------*/
extern lie_janela *lie_jan_corrente, raiz;
extern struct LIE_comunica comunica;

main()
{
  cadeia teste, kcvt();
  static imag_xlo, imag_ylo, min, max;

  lie_janela *jan1;
  int i, j, aux, aux2, nivel, erro;
  static int menu_id1, menu_idp, nlin=0, ncol=0;
  static int com1, comp;
  static int red[4]  ={0,0x3,0x3,0x0},
             green[4]={0,0x4,0x7,0x3},
             blue[4] ={0,0x1,0x7,0x3};
  static cadeia text_font = (cadeia)"-.";  /* Inicializa a cadeia */

  static cadeia entry1[MAXARQIMAG];
  static int command1[MAXARQIMAG];
  static int link1[MAXARQIMAG];
  ArqSpec arquivo[MAXARQIMAG];

  /* Definicao do menu principal */
  static cadeia entryp[]={(cadeia)"-.Le Imagem     ",
                          (cadeia)"-.Maximiza Hist.",
                          (cadeia)"-.Apaga         ",
                          (cadeia)"-.Sai de Cima   ",
                          (cadeia)"-.Fim           " };

  static int commandp[] = {1, 5, 2, 3, 4};
  static int linkp[]={0,0,0,0,0};

  Inicializa();  /* Inicializa o TMS e apaga a tela */
  Limpa_jan_corrente();


  /* Obter todos os arquivos com extencao .PIC, preparando o
     menu com os nomes dos arquivos */
/*  teste = kcvt (teste);  */
  teste[0]=6; teste[1]=5; teste[2]='*'; teste[3]='.';
  teste[4]='P'; teste[5]='I'; teste[6]='C';
```

```
i=i_arqinicio (teste, 23, &arquivo[0]);

if (i == 0) { /* Se tem ao menos um arquivo com extencao .PIC... */
    max=0;
    for (j=0; j<MAXARQIMAG - 1; j++) {
        entry1[j]=arquivo[j].fname;
        /* Limpar as cadeias onde vao os nomes */
        for (i=entry1[j][1]; i < entry1[j][0]; i++)
            entry[j][i+2]=' ';
        if (entry1[j][1]>max) max=entry1[j][1];
        command1[j]=TAMNP+1+j;  /* O menu pai tem TAMNP entradas */
        link1[j]=0;
        i = i_arqproximo(&arquivo[j+1]);
        if (i != 0) break;
        }

    /* Criar os menus  */
    j++;
    for (i=0;i<j;i++) entry1[i][1]=max;

    Mn_PDcria (red, green, blue, text_font, j, j<8 ?j:8, entry1,command1,
               link1, &menu_id1);

    /* Estabelecer uma hierarquia */
    linkp[0]=menu_id1;
    Mn_PDcria (red, green, blue, text_font, TAMNP, TAMNP, entryp,
                   commandp, linkp, &menu_idp);


    while (1) {
        i=Mn_PDprocessa (menu_idp, 0, 10, 1, &comp);
        if (i != 0)  { /* Deu erro */
           i_printk (" Erro =%d \n", i);
           break;
           }

        if (comp==4)     /* FIM */
            break;

        if (comp == 2) {  /* Apaga a janela */
           Limpa_jan_corrente ();
           continue;
           }

        if (comp == 3 )  { /* Tira o menu de Cima */
           i_getc();
           continue;
           }

        if (comp > TAMNP)  { /* Ler uma imagem */
           j=comp-(TAMNP+1);
           i=i_open(arquivo[j].fname, 0);

           /* A jan. da imagem sera aberta em (50,40)*/
           Jan_corrente (&raiz);
           imag_xlo=50; imag_ylo=40;
           erro=i_le_imagem (i,imag_xlo,imag_ylo,arquivo[j].fname,
                             &nlin, &ncol);
           i_close (i);
           continue;
           }
```

```
            if (comp == 5) { /* Maximiza a Imagem (Aumenta contraste) */
               if (nlin==0) /* Nao foi lida nenhuma imagem */
                  continue;
               /* Calcular minimo e maximo da Imagem */
               min=1000;  max=0;
               for (i=0; i< nlin; i++ )
                   for (j=0; j<ncol; j++) {
                       aux=get_pixel(imag_xlo+j, imag_ylo+i);
                       if (aux > max) max=aux;
                       if (aux < min) min=aux;
                       }
               aux2=max-min;
               for (i=0; i< nlin; i++ )
                   for (j=0; j<ncol; j++) {
                       aux=get_pixel(imag_xlo+j, imag_ylo+i);
                       nivel= (255 * (aux - min))/aux2;
                       put_pixel(nivel, imag_xlo+j, imag_ylo+i);
                       }
               continue;
               }
            }   /* Fim do while (1)  */
         }  /* Fim do if <existe arquivo> */
       else {  /* Nenhum arquivo com extencao .PIC */
         v_printk (" Nenhum arquivo .PIC\n");
         }

     i_printk ("\n\n          Processamento Encerrado.          ");
     exit (0);
}
```

6 - Conclusões.

Neste capítulo foi apresentado o ambiente gráfico desenvolvido para a arquitetura dedicada à visualização de imagens construída no LIE.

Este ambiente se destina a apoiar a construção de programas aplicativos que devem ser executados na arquitetura dedicada, provendo um conjunto de primitivas de software, as quais colocam à disposição do programador de aplicativos todos os recursos do hardware de uma maneira unificada e facilmente acessível através de primitivas de alto nível. Essas primitivas por sua vez foram implementadas de maneira a obter o máximo desempenho da arquitetura, tal como a utilização de Acesso Direto à Memória sempre que possível.

As primitivas foram construídas no sentido de estender a biblioteca padrão da linguagem "C", de maneira a facilitar seu aprendizado e utilização por parte de pessoas treinadas na utilização dessa linguagem. Assim, buscou-se também uma completeza na definição do conjunto de primitivas, buscando atender à necessidade de se dar apoio aos aplicativos que devam utilizar tal arquitetura para a visualização de imagens de uma forma genérica.

Embora tal arquitetura tenha a princípio capacidade computacional para suportar também o processamento (ao menos parcial) das imagens manipuladas, as restrições de memória e o tempo de processamento envolvidos tornam desvantajosa tal aplicação da arquitetura atual. Assim, não foram implementadas primitivas para o suporte a essas operações, pois considerou-se que esse processamento seria melhor efetuado no ambiente de processamento de imagens descrito no Capítulo 3. No entanto, é viável a ampliação da arquitetura de forma a tornar eficiente esse tipo de processamento (o que já encontra-se em fase de projeto no LIE), e nesse caso as primitivas já desenvolvidas para o ambiente Intergraph poderiam ser transportadas para esta arquitetura e integradas ao ambiente gráfico.

Bibliografia

[FOL_90] Foley, J.D.; van Dam, A.; Feiner, S.K.; Hughes, J.F.: ***Computer Graphics - Principles and Practice***, Second Edition, Addison-Wesley Publishing Company, 1990.

[HIX_89] Hix, D.: "***User Interfaces: Opening a Window on the Computer***", IEEE Software, pp. 8-10, janeiro de 1989.

[INT_88a] Intergraph Corp.: ***Environ V - C Programmer's Reference Manual***, Huntsville, Al, 1988.

[LEE_89] Lee, Ed: ***User-interface Development Tools***", IEEE Software, pp. 31-36, maio de 1989.

[MYE_88] Myers, B.D.: "***A Taxonomy of Window Manager User Interfaces***", IEEE Computer Graphics & Applications, Vol. 8, No. 5. pp. 65-84, setembro de 1988.

[MYE_89] Myers, B.D.: "***User-Interface Tools: Introduction and Survey***", IEEE Software, pp. 15-23, janeiro de 1989.

[PAI_90] Paiva, M.S.V.: "***Projeto de Uma Arquitetura de Hardware para Visualização de Imagens Digitais***", tese de doutoramento apresentada ao Instituto de Física e Química de São Carlos - USP, 20/11/90.

[PAN_85] Panepucci, H.; Donoso, J.P.; Tannús, A.; Beckamann, N.; Bonagamba, T.: "***Novas Imagens do Corpo***", Ciência Hoje, Vol 20, No. 4, pp. 46-56, setembro/outubro de 1985.

[PAN_87] Panepucci, H.; Beckamann, N.; Tannús, A.; Bonagamba, T. J: "***NMR Imaging In: Recent Advances in Organic NMR Spectroscopy***", editores J. B. Lambert e Roberto Rittner, Norell Press (USA) 1987, cap. 13, pp. 171-181.

[SUN_89a] Sun Microsystems: ***OPEN LOOK Graphical User Interface Functional Specification***, Addison-Wesley Publishing Company, 1989.

[SUN_89b] Sun Microsystems: ***OpenWindows Developer's Guide 1.1 - Installation Manual***, Sun Microsystems, Mountain View, CA, 1990.

[SUN_90a] Sun Microsystems: ***Sun Software Configuration Guide***, Sun Microsystems, Mountain View, CA, 1990.

[SUN_90b] Sun Microsystems: ***OPEN LOOK Graphical User Interface Aplication Style Guidelines***, Addison-Wesley Publishing Company, 1990.

[SUN_90c] Sun Microsystems: ***X11/NeWS Version 2 Server Guide***, Sun Microsystems, Mountain View, CA, 1990.

[TAN_87] Tannús, A. : "***Desenvolvimento de Tecnologia de Tomografia por Ressonância Magnética Nuclear***", tese de doutoramento apresentada ao IFQSC-USP, agosto de 1987.

[TEX_87] Texas Instruments: ***TMS34010 Assembly Language Tools User's Guide***, 1987.

Instituto de Ciências Matemáticas de São Carlos

ISSN-0109-2569

Primitivas do Ambiente Gráfico LIE-SJ
- Manual de Referência -

AGMA JUCI MACHADO TRAINA
JAN FRANZ WILLEM SLAETS

Nº 14

RELATÓRIOS TÉCNICOS DO ICMSC

SÃO CARLOS
Nov./ 1993

Universidade de São Paulo
Instituto de Ciências Matemáticas de São Carlos
Departamento de Ciências de Computação e Estatística

Primitivas do Ambiente Gráfico LIE-SJ
- Manual de Referência -

Agma Juci Machado Traina

Jan Franz Willem Slaets

Introdução

No Instituto de Física e Química de São Carlos - USP foi desenvolvido um Ambiente Gráfico denominado LIE-SJ, que suporta a construção de aplicativos "amigáveis" para visualização de imagens.

Este documento agrupa as rotinas do ambiente gráfico em três categorias:

- *Rotinas Básicas - LIEGSP:* são as rotinas que dependem diretamente da arquitetura, ou que fazendo parte do sistema, não é desejável que o usuário comum tenha acesso a elas. Em sua grande maioria são escritas em linguagem assembly do TMS34010.
- *Rotinas de Apoio - LIESJ:* são as rotinas que preparam os dispositivos periféricos para serem utilizados pelo nível de rotinas de mais alto nível, ou realizam operações sobre as estruturas de dados que armazenam toda a informação sobre o ambiente.
- *Rotinas que compõem o Gerenciador de Janelas - LIEGJ:* são as rotinas que o programador utiliza para a criação e manutenção de janelas. Elas são responsáveis pela "aparência" das mesmas.

Cada uma das rotinas apresentadas a seguir fazem parte de um dos módulos do LIE-SJ. Referências adicionais podem ser encontradas em:

[TRAINA_91] Agma Juci M. Traina, **"Estudo e Implementação de Software Dedicado para um Sistema de Visualização de Imagens"**, Tese de Doutoramento, IFQSC-USP, julho de 1991.

[TRAINA_93] Agma Juci Machado Traina e Jan Franz Willem Slaets, **"Um Ambiente Gráfico para Visualização de Imagens Tomográficas"**, Relatórios Técnicos do ICMSC, Nº 12, outubro de 1993.

NOME

Cria_arvore - Inicializa a estrutura de Janelas.

SINOPSE

```
void Cria_arvore (maxlin, maxcol)
int maxlin, maxcol;
```

DESCRIÇÃO

A rotina *Cria_arvore* inicializa a estrutura de janelas, e o primeiro nó dessa estrutura, que corresponde à raiz da arvore, e à tela inteira do vídeo, que poderá ser utilizada. As variáveis *maxlin* e *maxcol* definem o tamanho máximo da tela em pixels. A variável *raiz* é definida estaticamente, e pode passar a ser acessada a partir de outros módulos. A raiz passa a ser definida para o fonte corrente, o qual deve ser estabelecido antes da chamada desta rotina.

NOTA

- O valor de *maxlin* e *mascol* indicados definem a area do vídeo que poderá ser utilizada posteriormente.
- O fonte corrente será tomado como fonte da raiz, e deve ser definido antes da chamada desta rotina.

VEJA TAMBÉM

Cria_filho(3LIESJ), Retira_filho(3LIESJ), Ret_subarvore(3LIESJ), Inicializa(3LIEGJ).

EXEMPLO

```
#include <data_str.h>
#include <sysconst.h>
extern byte charset[96][16];
lie_fonte lie_fonte_corrente = {
            (byte *)charset,
            8, 16};
 . . .
Cria_arvore(YMAX, XMAX);
```

NOME

Cria_filho - cria uma estrutura de janela.

SINOPSE

```
#include <data_str.h>

int Cria_filho(ret_filho, corrente)
lie_janela **retfilho, *corrente;
```

DESCRIÇÃO

Esta rotina cria uma janela subordinada (filho) da janela indicada como *corrente*, recebendo por berança os parâmetros de definição dessa janela. Imediatamente seguindo à chamada desta rotina, deve-se proceder à definição dos parâmetros reais que essa janeja deverá ter.

RETORNO

Valor de Retorno da Rotina (tipo int):

Indica se houve erro (vide ERROS:).

ret_filho:

descritor da janela criada.

RESTRIÇÕES

- *corrente* deve ser uma janela válida já existente.

NOTA

- Todos os parâmetros da janela *corrente* indicada são duplicados para a janela *ret_filho* criada.

ERROS

0 - Tudo OK. Janela criada

1 - Janela *corrente* indicada não é uma janela válida.

2 - Memória insuficiente para esta operação.

VEJA TAMBÉM

Cria_arvore(3LIESJ), Retira_filho(3LIESJ), Ret_subarvore(3LIESJ).

EXEMPLO

```
#include <data_str.h>
extern lie_janela *raiz;
lie_janela *novajan;
 . . .
Cria_filho (&novajan, raiz); /* Cria uma janela
      diretamente na tela */
```


NOME

Cria_jan_borda - Cria uma janela com borda.

SINOPSE

```
#include <data_str.h>

int Cria_jan_borda (wid, parval)
int parval;
lie_janela **wid;
```

DESCRIÇÃO

Esta rotina cria uma janela com borda, como descendente da janela corrente. Os parametros dessa janela são assumidos como tendo os mesmos valores da janela corrente, a menos daqueles que são explicitamente alterados através de um par parâmetro-valor especificado (*parval*). O identificador da nova janela é retornado como *wid*. A janela apenas e' criada, porém não se torna corrente, e nem é mapeada na tela. Para que uma janela se torne corrente, usa-se a rotina *Jan_corrente*(3LIEGJ). Para mapear a janela, usa-se a a rotina *Mostra_Janela*(3LIEGJ).

RETORNO

Valor de Retorno da Rotina (tipo int):

Indica se houve erro (vide ERROS:).

wid:

identificador da janela criada.

RESTRIÇÕES

- Os limites (x,y) da janela devem ser especificados em coordenadas relativas à janela corrente.
- O cursor é especificado em coordenadas relativas à janela criada.
- Os limites (x,y) da janela criada devem estar inteiramente contidos dentro da janela corrente, caso contrário a janela será truncada ("clipping") para esses limites.

PARAMETROS TRATADOS

XLO YLO	XHI YHI
PLANE_MASK	EVENT_MASK
COR_FRENTE	COR_FUNDO
X_CURSOR	Y_CURSOR
COR_FRENTE_BORDA	
COR_FUNDO_BORDA	
TITULO	
SEM_BORDA	COM_BORDA
ELEV_VERTICAL	
ELEV_HORIZONTAL	
TRACO_SIMPLES	
TRACO_DUPLO	
SOBREPOE	
BORDA_SINGELA	

NOTA

Deve-se atentar para a exatidão da correspondência entre atributo e valor. Caso exista a falta de algum parâmetro, os argumentos serão tomados em seqüência, levando a resultados imprevisíveis.

ERROS

0 - Tudo OK. Janela criada corretamente.
1 - Estrutura da janela indicada danificada.
2 - Memória insuficiente para esta operação.
10 - Margens fora de limite. Janela criada mas truncada.
11 - Parâmetro Inválido.

VEJA TAMBÉM

Jan_corrente(3LIEGJ), Mostra_janela(3LIEGJ), Cria_janela(3LIEGJ).

EXEMPLO

```
      /* Criar uma janela descendente diretamente da tela
inteira:
      */
      extern lie_janela raiz;
      lie_janela *cria;
       . . .
      Jan_corrente (&raiz);
      /* torna a raiz janela
       corrente */
      Cria_jan_borda (&cria,
            XLO, 20,   YLO, 10,
            XHI, 110, YHI, 200,
            COR_FRENTE, 0xFFFF,
            COR_FUNDO, 0,
            COR_FRENTE_BORDA,
               0xAAAAAAAA,
            COR_FUNDO_BORDA, 0x44444444;
            FIM);
```

NOME

Cria_janela - Cria uma janela básica sem borda.

SINOPSE

```
#include <data_str.h>

int Cria_janela (wid, parval)
int parval;
lie_janela **wid;
```

DESCRIÇÃO

Esta rotina cria uma janela básica, como descendente da janela corrente. Os parâmetros dessa janela são assumidos como tendo os mesmos valores da janela corrente, a menos daqueles que são explicitamente alterados atraves de um par parametro-valor (*parval*) especificado. O identificador da nova janela é retornado como *wid*. A janela apenas e' criada, porém não se torna corrente, e nem é mapeada na tela. Para que uma janela se torne corrente usa-se a rotina *Jan_corrente*(3LIEGJ). Para mapear a janela, usa-se a rotina *Mostra_Janela*(3LIEGJ).

RETORNO

Valor de Retorno da Rotina (tipo int):

Indica se houve erro (vide ERROS:).

wid:

identificador da janela criada.

RESTRIÇÕES

- Os limites (x,y) da janela devem ser especificados em coordenadas relativas à janela corrente.
- O cursor é especificado em coordenadas relativas à janela criada.
- Os limites (x,y) da janela criada devem estar inteiramente contidos dentro da janela corrente, caso contrário a janela será truncada ("clipping") para esses limites.

PARAMETROS TRATADOS

XLO YLO XHI YHI
PLANE_MASK EVENT_MASK
COR_FRENTE COR_FUNDO
X_CURSOR Y_CURSOR
SOBREPOE

NOTA

Deve-se atentar para a exatidão da correspondência entre atributo e valor. Caso exista a falta de algum parâmetro, os argumentos serão tomados em seqüência, levando a resultados imprevisíveis.

ERROS

0 - Tudo OK. Janela criada corretamente.
1 - Estrutura da janela indicada danificada.
2 - Memória insuficiente para esta operação.
10 - Margens fora de limite. Janela criada mas truncada.
11 - Parametro Inválido.

VEJA TAMBÉM

Jan_corrente(3LIEGJ), Mostra_janela(3LIEGJ), Cria_jan_borda(3LIEGJ).

EXEMPLO

```
/* Criar uma janela sem borda descendente
diretamente da tela inteira: */
extern lie_janela raiz;
lie_janela *cria;
 . . .
Jan_corrente (&raiz); /* torna a raiz a janela
corrente */
Cria_janela (&cria,
      XLO, 20,   YLO, 10,
      XHI, 110,  YHI, 200,
      FIM);
```

NOME

Esconde-janela - Retira uma janela da tela.

SINOPSE

```
#include <data_str.h>

int Esconde_janela (wid)
lie_janela *wid;
```

DESCRIÇÃO

A rotina *Esconde-janela* retira da tela a janela indicada, mantendo a definição da mesma na estrutura de janelas, de maneira que esta possa ser mostrada posteriormente através de uma chamada da rotina *Mostra_janela*(3LIEGJ). A janela pai da janela retirada torna-se a janela corrente. Podem ser escondidas janelas com ou sem borda.

RETORNO

Valor de Retorno da Rotina (tipo int):
Indica se houve erro (vide ERROS:).

RESTRIÇÕES

- A janela indicada deve ser uma janela válida.
- Não pode ser escondida a janela raiz (própria tela).

ERROS

0 - Tudo OK. Janela foi escondida.
1 - Estrutura de janela indicada danificada, ou não existe janela corrente.

VEJA TAMBÉM

Mostra_janela(3LIEGJ).

NOME

Fixa_jan_fisica - Estabelece o ambiente de Hardware para a janela.

SINOPSE

```
#include <data_str.h>

int Fixa_jan_fisica (wid);
lie_janela *wid;
```

DESCRIÇÃO

Esta rotina estabelece os parâmetros físicos do hardware que permitem realizar as operações seguintes sobre a area de vídeo correspondente à janela *wid* indicada. São fixados os registradores "PMASK", "WSTART", "WEND", CONTROL, COLOR0 e COLOR1.

RETORNO

Valor de Retorno da Rotina (tipo int):
Indica se houve erro (vide ERROS:).

ERROS

0 - Tudo OK.
1 - Estrutura da janela indicada está inconsistente.

VEJA TAMBÉM

JanCorrente(3LIEGJ).

NOME

Inicializa - Prepara o Sistema para iniciar as operações.

SINOPSE

```
Inicializa()
```

DESCRIÇÃO

Esta rotina inicializa o ambiente de execução do Sistema de Gerenciamento de Janelas para aceitar solicitações de operações. Esta deve ser a primeira rotina do sistema a ser chamada, pois caso contrário as operações não serão adequadamente atendidas. A tela de vídeo é apagada, e o TMS34010 é inicializado.

RETORNO

Valor de Retorno da Rotina (tipo int):

Indica se houve erro (vide ERROS:).

RESTRIÇÕES

- Esta deve ser a primeira rotina do SGJ a ser chamada.

ERROS

0 - Tudo OK.

1 - Erro na inicialização. Sistema não está em operação.

EXEMPLO

```
main()
{
/* declarações de variáveis */
 . . .
/* Inicio do procedimento */
Inicializa(); /* Iniciliza TMS e apaga a
                 tela */
 . . .
```


NOME

Jan_corrente - Torna corrente a janela indicada.

SINOPSE

```
#include <data_str.h>

int Jan_corrente(wid)
lie_janela *wid;
```

DESCRIÇÃO

Esta rotina torna "corrente" a janela indicada. Janela corrente é aquela subentendida para realizar diversas operações, entre elas as operações de saída de dados. Assim, sempre que é solicitada a saída de dados, esta sempre tem efeito na janela corrente.

RETORNO

Valor de Retorno da Rotina (tipo int):

Indica se houve erro (vide ERROS:).

ERROS

0 - Tudo OK.

1 - Estrutura da janela indicada está inconsistente.

EXEMPLO

```
/* Cria uma janela na tela e a torna corrente */
#include <data_str.h>
extern lie_janela raiz;
lie_janela *jan1;
Jan_corrente (&raiz); /* torna a raiz a janela
corrente */
Cria_janela (&jan1,
      XLO, 20,   YLO, 10,
      XHI, 110,  YHI, 200,  FIM);
Jan_corrente (jan1); /* torna a janela criada
corrente */
```

NOME

Limpa_jan_corrente - Limpa a Janela Corrente.

SINOPSE

```
void Limpa_jan_corrente();
```

DESCRIÇÃO

Esta rotina preenche toda a janela corrente com o pixel de cor de fundo definido para a janela.

RESTRIÇÕES

• Deve existir uma janela corrente consistente, e que esteja mapeada no vídeo.

VEJA TAMBÉM

Jan_corrente(3LIEGJ), Mostra_janela(3LIEGJ), Cria_jan_borda(3LIEGJ), Cria_janela(3LIEGJ).

EXEMPLO

```
lie_janela *jan1;
 . . .
/* Criar uma janela subordinada 'a
      corrente */
Cria_janela (&jan1, XLO, 20, YLO, 10,
            XHI, 110,  YHI, 200,
            COR_FUNDO, 0,
            COR_FRENTE, 0xFFF,
            FIM);
Jan_corrente(jan1);      /*  Tornar  corrente  e
                         limpar  a  janela  cujo
                         identificador é jan1 */
Mostra_janela();
Limpa_jan_corrente();
```

NOME

Lista_arvore - Lista a estrutura de janelas.

SINOPSE

```
#include <data_str.h>

void Lista_arvore (wid)
lie_janela *wid;
```

DESCRIÇÃO

Esta rotina lista na interface de vídeo do hospedeiro a sub-estrutura de janelas atual a partir do nó correspondente à janela *wid* indicada. O propósito desta rotina é o de rastrear a estrutura, para a depuração de aplicativos.

VEJA TAMBÉM

Cria_arvore(3LIESJ), Cria_filho(3LIESJ), Retira_filho(3LIESJ), Ret_subarvore(3LIESJ).

EXEMPLO

```
#include <data_str.h>
extern lie_janela *raiz;
 . . .
Lista_arvore (raiz); /* Lista toda a
            estrutura de janelas */
```

NOME

Mn_PDcria - Definir um Menu Pull-Down.

SINOPSE

```
#include <data_str.h>

int Mn_PDcria (red, green, blue,
               text_font, nro_entries,
               scr_size, entry, command,
               link, menu_id_ptr)

unsigned int red[4], green[4], blue[4];
/* Intensidade da cor, no indice:
         0 - cor do fundo
         1 - cor da borda
         2 - cor dos textos regulares
         3 - cor da entrada em highlight */
cadeia text_font; /* Nome do arquivo de fontes
                     para os caracteres */
int nro_entries;  /* Número maximo de entradas no
                     menu */
int scr_size;          /* número de entradas
                          simultaneas na janela */
cadeia entry[];   /* Nomes das entradas */
int command[];    /* Número que deve retornar se
                     essa entrada for escolhida */
int link[];       /* identidade do menu que pode
                     dar prosseguimento 'a essa
                     entrada, se ela for escolhida */
register int *menu_id_ptr;     /* Identidade desse
                                  menu,    atribuido
                                  pela rotina */
```

DESCRIÇÃO

Esta rotina cria um Menu Pull-Down segundo a definição dada pelo usuário. As entradas do menu são definidas pelos vários vetores *entry*, *command*, e *link*, sendo que a cada índice do vetor corresponde uma entrada. O número máximo de entradas *nro_entries* determina a quantidade de entradas que o menu pode comportar. Do total de entradas do menu, apenas a quantidade indicada em *scr_size* é mostrada simultaneamente na tela, sendo que as demais são mostradas através de operações de "scroll".

Os vetores indicadores de cores *red*, *green*, e *blue* determinam a intensidade com que cada cor básica participa da formação da cor na parte correspondente do menu. Para isso, a cor é composta utilizando-se os bits de mais baixa ordem de cada entrada. A quantidade de bits utilizada depende do número de planos de cor que está sendo utilizado pelo hardware.

A entrada *text_font* destina-se a ser usada para indicar o fonte de letra a ser utilizado para escrever-se as entradas do menu, quando existir mais de um fonte definido no sistema. Caso deva ser utilizado o fonte corrente, deve ser indicado o valor <NULO>.

A variável *menu_id_ptr* no retorno da rotina conterá o identificador do menu criado, o qual deve ser utilizado para se indicar que este menu deve ser ativado, ou para se estabelecer a hierarquia de menus através do *link* de cada entrada. O *link*, se for indicado como zero indica que a entrada correspondente é folha, e se escolhida corresponde à uma escolha final, terminando o processamento do menu. Caso esse valor não seja zero, deve indicar um identificador de menu válido, o qual corresponde ao sub-menu ativado quando a entrada correspondente for escolhida.

Uma vez criado um menu, este pode ser ativado através da rotina *Mn_PDprocessa*(3LIESJ). Sub-menus escolhidos durante o processamento de um menu são por sua vez ativados automaticamente quando escolhidos.

RETORNO

Valor de Retorno da Rotina (tipo int):

Indica se houve erro (vide ERROS:).

menu_id_ptr - identificador do menu criado.

RESTRIÇÕES

• Deve ser indicado um número de entradas *nro_entries* de no mínimo duas entradas.

• Não pode ser excedido o número máximo de menus criados definido como MAXMENU no arquivo *<sysconst.h>*.

ERROS

0 - Não houve erro.

12 - Número máximo de definições excedido (máximo MAXMENU).

VEJA TAMBÉM

Mn_PDprocessa(3LIESJ).

EXEMPLO

```
main()
{
  /* Definições do menu */
  int menu_princ;
  /* Definição das cores basicas */
  static int red[4]   = {0,7,1,0},
      green[4] = {2,7,1,2},
      blue[4]  = {0,7,0,2};
  static char *text_font = 0;
  static char *entry[]={
        "\x0f\x0f Opcao 1      ",
      "\x0f\x0f Opcao 2      ",
      "\x0f\x0f Opcao tres   ",
      "\x0f\x0f Opcao quatro ",
      "\x0F\x0F Parametros   ",
      "\x0f\x0f Fim          "};
  static int command[]   = {1,2,3,4,50,500};
  static int link[]      = {0,0,0,0, 0,  0};
  static int i; /* Retorno do processa menu */

  /* Cria menu */
  Mn_PDcria (red, green, blue, text_font, 6, 4,
            entry, command, link, &menu_princ);
  i = Mn_PDprocessa
              (menu_princ,0,5,55,&comp);
```

NOME

Mn_PDprocessa - Ativa o processamento do Menu indicado.

SINOPSE

```
#include <data_str.h>

int Mn_PDprocessa (menu_id, screen_nro, lin_org,
      col_org, command_ptr)

int menu_id;       /* Número de identidade do menu
                   que deve ser ativado, devolvido
                   pela rotina Mn_PDcria(). */
int screen_nro;    /* Número da tela de vídeo que
                   deve ser usada. */
int lin_org, col_org;   /* Posição do canto
                        superior esquerdo onde o
                        menu deve ser aberto, em
                        coordenadas do vídeo em
                        termos de linha e coluna.
                        */
int *command_ptr; /* Número da entrada escolhida.
                  Corresponde ao número do vetor
                  command definido para esta
                  entrada para este menu. */
```

DESCRIÇÃO

Esta rotina ativa o processamento do menu indicado em *menu_id*, mostrando-o na tela, e permitindo que o usuário movimente o cursor segundo sua vontade. O menu a ser mostrado deve ter sido previamente definido pela rotina *Mn_PDcria*. Quando uma entrada é escolhida, retorna-se o valor correspondente à entrada atualmente em highlight na variável *command_ptr*. A janela correspondente ao menu é aberta na tela tendo como coordenadas de seu canto superior esquerdo aquelas indicadas em *lin_org* e *col_org*. Caso a janela não possa ser totalmente mostrada na janela a partir dessas coordenadas, estas são reduzidas em direção ao canto superior esquerdo da tela, até que a janela possa ser inteiramente mapeada. As dimensões da janela são calculadas automaticamente através das entradas definidas para o menu.

A variável *screen_nro* não está sendo utilizada, sendo mantida na definição por motivo de compatibilidade com o ambiente "Environ V".

RETORNO

Valor de Retorno da Rotina (tipo int):

Indica se houve erro (vide ERROS:).

command_ptr - código de comando associado à entrada escolhida. Caso nenhuma entrada tenha sido escolhida, retorna o valor -1.

- O menu indicado deve ter sido definido como um menu "pull-down"
- As dimensões do menu devem caber na tela.

ERROS

0 - Não houve erro. resultado válido
13 - menu_id indicado não válido
14 - menu_id indicado não definido

VEJA TAMBÉM

Mn_PDcria(3LIEGJ).

EXEMPLO

```
/* Utilizando-se do menu definido no exemplo da
      rotina Mn_PDcria: */
i = Mn_PDprocessa (menu_princ,0,5,55,&comp);
if (comp == 7) break;
switch(comp){
      case 1 : /* Opção 1 */
        . . .
        break;
      case 2 : /* Opção 2 */
        . . .
        break;
      case 3 : /* Opção tres */
        . . .
        break;
      case 4 : /* Opção quatro */
        . . .
        break;
      case 5 : /* Trata parametros */
        . . .
        break;
      case 6 : /* Finaliza */
        . . .
        break;
}
```

NOME

Mostra_janela - Mapeia a janela corrente no vídeo.

SINOPSE

```
int Mostra_janela()
```

DESCRIÇÃO

A rotina *Mostra_janela* mapeia e mostra no vídeo a janela corrente. Caso a janela corrente seja uma janela com borda, a borda é mostrada também, juntamente com os atributos correntes para essa janela e borda.

RETORNO

Valor de Retorno da Rotina (tipo int):
Indica se houve erro (vide ERROS:).

RESTRIÇÕES

- Deve haver uma janela corrente consistente.

NOTA

Esta rotina efetua todo o tratamento para permitir a manipulação de janelas com ou sem borda.

ERROS

0 - Tudo OK.
1 - Estrutura da janela corrente danificada, ou não existe janela corrente.

VEJA TAMBÉM

Esconde_janela(3LIEGJ).

NOME

Move_cursor - Posiciona o cursor da janela corrente.

SINOPSE

```
Move_cursor (x, y);
int x, y;
```

DESCRIÇÃO

Move o cursor da janela corrente para a posição absoluta (em relação à tela) indicada.

RETORNO

Valor de Retorno da Rotina (tipo int):
Indica se houve erro (vide ERROS:).

RESTRIÇÕES

- Deve existir uma janela corrente válida.

ERROS

0 - Tudo OK.
1 - Estrutura da janela corrente danificada, ou não existe janela corrente.

VEJA TAMBÉM

Cria_janela(3LIEGJ), Posiciona(3LIEGJ).

EXEMPLO

```
Move_cursor (10, 20);    /* Move o cursor para a
                            coordenada absoluta do
                            vídeo (10,20). */
```

NOME

Ret_subarvore - Retira uma subarvore da estrutura de janelas.

SINOPSE

```
#include <data_str.h>

int Ret_subarvore(wid)
lie_janela *wid;
```

DESCRIÇÃO

Esta rotina elimina da estrutura de janelas a janela *wid* indicada, bem como todas as janelas suas subordinadas. Note-se que a rotina não verifica se a janela está mostrada no vídeo, o que implica a necessidade de se retirar a janela (e suas subordinadas) do vídeo, através da rotina *Esconde_janela* antes da chamada desta rotina.

RETORNO

Valor de Retorno da Rotina (tipo int):

Indica se houve erro (vide ERROS:).

RESTRIÇÕES

- A janela *raiz* não pode ser retirada.

ERROS

0 - Tudo OK.
1 - Janela indicada inconsistente.
4 - A janela raiz não pode ser retirada.

VEJA TAMBÉM

Cria_arvore(3LIESJ), Cria_filho(3LIESJ), Retira_filho(3LIESJ), Esconde_janela(3LIEGJ), Retira_janela(3LIEGJ).

NOME

Retira_filho - Retira um nó da estrutura de janelas.

SINOPSE

```
#include <data_str.h>

int Retira _filho (wid)
lie_janela *wid;
```

DESCRIÇÃO

Esta rotina retira a janela *wid* da estrutura de janelas. Note-se que a rotina não verifica se a janela está mostrada no vídeo, o que implica a necessidade de se retirar a janela do vídeo, através da rotina *Esconde_janela* antes da chamada desta rotina. Para que esta rotina possa retirar a janela, é necessário que esta não tenha nenhuma outra janela subordinada.

RETORNO

Valor de Retorno da Rotina (tipo int):

Indica se houve erro (vide ERROS:).

RESTRIÇÕES

- A janela a ser retirada não deve estar mapeada no vídeo, e nem possuir janelas subordinadas.

ERROS

0 - Tudo OK.
1 - Janela indicada inconsistente.
3 - A janela possui filhos, não pode ser retirada.
4 - A janela raiz não pode ser retirada.

VEJA TAMBÉM

Cria_arvore(3LIESJ), Cria_filho(3LIESJ), Ret_subarvore(3LIESJ), Esconde_janela(3LIEGJ), Retira_janela(3LIEGJ).

NOME

Retira_janela - Retira a janela especificada da estrutura de janelas.

SINOPSE

```
#include <data_str.h>

int Retira_janela (wid)
lie_janela *wid;
```

DESCRIÇÃO

Retira a janela especificada como *wid* da hierarquia de Janelas. Caso a janela tenha sido criada com a opção SOBREPOE (vide *Cria_janela*(3LIEGJ)), e esteja presentemente mapeada na tela, então a janela também será desmapeada da tela.

RETORNO

Valor de Retorno da Rotina (tipo int):

Indica se houve erro (vide ERROS:).

RESTRIÇÕES

- Se a janela especificada não foi criada com a opção de sobreposição, ela será apenas retirada da estrutura hierárquica, mas não será desmapeada da tela.
- Não se pode retirar uma janela que possua filhos, ou a janela raiz.

ERROS

0 - Tudo OK.
1 - Estrutura da janela indicada está danificada.
3 - Janela especificada possui filhos.
4 - Janela especificada é a raiz.

VEJA TAMBÉM

Limpa_janela(3LIEGJ), Cria_janela(3LIEGJ).

EXEMPLO

```
      /* Criar uma janela descendente direta da tela
inteira: */
      extern lie_janela raiz;
      lie_janela *cria;
      Jan_corrente (&raiz); /* torna a raiz janela
corrente */

      /* Neste caso a janela não sera desmapeada da
            tela, apenas da estrutura:*/
      Cria_janela (&cria,
            XLO, 20,    YLO, 10,
            XHI, 110,   YHI, 200,
            FIM);
      Ret_janela (cria);
       . . .
      /* Neste caso a janela sera desmapeada da tela e da
      estrutura:*/
      Cria_janela (&cria,
            XLO, 20,    YLO, 10,
            XHI, 110,   YHI, 200,
            SOBREPOE, SIM,
            FIM);
      Ret_janela (cria);
       . . .
```

NOME

draw_car - escreve um caráter no vídeo.

SINOPSE

```
int draw_car (pc, x, y)
char *pc;
int x, y;
```

DESCRIÇÃO

Esta rotina escreve um caráter no vídeo, no retângulo cujo vértice esquerdo superior é definido como estando na posição indicada por *x* e *y*. O caráter é indicado pelo ponteiro *pc*, e é impresso segundo o fonte presentemente definido como corrente.

RETORNO

Valor de Retorno da Rotina (tipo int):

Constante zero.

RESTRIÇÕES

- A rotina não verifica se a posição indicada corresponde à uma posição válida do vídeo, portanto é necessário que as coordenadas *x* e *y* estejam consistentes, caso contrário uma posição errônea, ou invasão de memória poderá ocorrer.

EXEMPLO

```
/* Escrever o vetor "vet" de 10 caracteres */
extern lie_fonte lie_fonte_corrente;
char vet[]="1234567890";
char aux;
int i;

aux=vet;
for (i=0; i<10; i++)
    draw_car (aux++,
      x+(i*lie_fonte_corrente.tam_car_x),
      y);
```

NOME

erase_screen - Limpa toda a tela.

SINOPSE

```
void erase_screen()
```

DESCRIÇÃO

Esta rotina coloca todos os pixels da tela com o valor 0x1515 (cinza claro). O tamanho da tela em pixels é obtido pelas constantes *XMAX* e *YMAX* definidas no arquivo *<sysconst.h>*. Note-se que esta rotina apaga a tela independentemente de existirem ou não janelas mapeadas. Se isso ocorrer, a atualização de informações em janelas posteriores à chamada desta rotina ficam truncadas. Portanto, esta rotina somente deve ser chamada quando não houverem janelas mapeadas.

RESTRIÇÕES

- Não devem existir janelas mapeadas na tela.

NOME

fill - preenche um retângulo com uma cor.

SINOPSE

```
void fill(width, height, xleft, ytop)
long width, height;  /* largura e altura do retângulo */
long xleft, ytop;  /* coord's do canto superior esquerdo do retângulo */
```

DESCRIÇÃO

Esta função preenche um retângulo da tela, cuja especificação e dada pelas variaveis *width* e *height*. As coordenadas (*xleft*,*yleft*) especificam o canto superior esquerdo do retângulo. A cor que prenchera o retângulo deve ser previamente carregada no registrador COLOR1 (B9), antes de se chamar esta função.

VEJA TAMBÉM

control_register (3LIEGSP)

EXEMPLO

```
...
/* Este trecho do programa preenche o retangulo de
largura 100 e altura 200, cujo canto superior
esquerdo e' (0,0) com a cor 0x1515. O conteúdo do
registrador CONTROL influencia como será preenchido
o retangulo */
set_color1 (0x1515);
fill(100,200,0,0);
...
```

NOME

get_control - Retorna o conteúdo do registrador CONTROL.
set_control - Estabelece o conteúdo do registrador CONTROL.

SINOPSE

```
int get_control()
int set_control(mval)

int mval;   /* mascara que sera colocada no reg.
CONTROL */
```

DESCRIÇÃO

Estas rotinas obtém/colocam o valor do registrador CONTROL do TMS34010.

RETORNO

Valor de Retorno da Rotina (qualquer) (tipo int):
Conteúdo atual do registrador CONTROL.

VEJA TAMBÉM

No manual *TMS34010: User's Guide* [TEX_86a], é apresentada uma descrição dos campos de funções do registrador CONTROL.

NOME

get_pixel - Obtém o valor do pixel na coordenada especificada.
put_pixel - Estabelece o valor do pixel na coordenada especificada.

SINOPSE

```
int get_pixel (x, y)
void put_pixel (valor, x, y)
int valor;
int x, y;
```

DESCRIÇÃO

A rotina *get_pixel* obtém o valor (cor) do pixel na coordenada especificada. O valor do pixel é indicado nos "*n*" bits menos significativos do valor de retorno da rotina, onde "*n*" é o número de planos de imagem (o tamanho do pixel) do sistema em uso. Os demais bits desse valor voltam como "0".

A rotina *put_pixel* coloca o valor (cor) indicado em *valor* no pixel indicado, usando para isso os "*n*" bits menos significativos.

A coordenada indicada por *x* e *y* é tomada como relativa ao canto superior esquerdo da tela.

RETORNO

Valor de Retorno da Rotina (tipo int):
Indica o valor do pixel na coordenada indicada.

NOME

i_arqinicio - Inicia busca de arquivos, posiciona no primeiro.
i_arqproximo - Prossegue busca de arquivos, posiciona no seguinte.

SINOPSE

```
#include <data_str.h>

erro = i_arqinicio(specname,atributos,cabecalho);
erro = i_arqproximo(cabecalho);

int erro;              Código de erro retornado
                       pelo DOS
cadeia specname;  Cadeia contendo a especificaçao
                  de procura
int atributos;         Atributos dos arquivos a
                       serem procurados
ArqSpec *cabecalho;    Cabeçalho do arquivo
                       encontrado
```

DESCRIÇÃO

A rotina *i_arqinicio* percorre o diretório corrente ou o diretório especificado pelo path em *specname* em busca da primeira entrada que atenda ao nome de arquivo indicado. Esse nome pode conter wild_cards (* e ?). Além disso, esta rotina armazena as informaçoes necessárias para que a rotina *i_arqproximo* possa encontrar a próxima entrada que atenda a mesma indicação.

A rotina *i_arqproximo* percorre as entradas de diretório restantes procurando pela próxima que atenda ao padrão de arquivo indicado como os argumentos *specname* e *atributos* da última chamada da rotina *i_arqinicio*. Esses argumentos são mantidos internamente, de maneira que esta rotina pode operar corretamente.

Cada arquivo encontrado tanto por *i_arqinicio* quanto por *i_arqproximo* retorna uma estrutura do tipo *ArqSpec*, definida no arquivo *data_str.h*.

Para que um nome de arquivo seja encontrado, o nome deve corresponder ao padrão de procura, bem como deve corresponder aos *atributos* solicitados. A especificação em *specname* pode ser qualquer nome correto, incluindo a especificação de driver, subdiretórios e wild-cards no nome do arquivo (na indicação de subdiretórios wild cards não são aceitos).

O parâmetro *atributos* indica os tipos de arquivos que devem ser considerados para a procura. O valor desse atributo é uma concatenação por OR dos seguintes valores:

0 - Arquivo normal
1 - Arquivo Read-only
2 - Arquivo Oculto (Hidden)
4 - Arquivo de Sistema
16 - Subdiretório

Dessa forma por exemplo, o valor 23 acessa todas as entradas do diretório, enquanto o valor 2 acessa todas as entradas de arquivos normais, ocultos ou não.

RETORNO

Valor de Retorno da Rotina (tipo int):
Indica se houve erro (vide ERROS:).

MÉTODO

A rotina *i_arqinicio* é usada juntamente com a rotina *i_arqproximo* para realizar uma busca em todo o diretório pelo(s) arquivo(s) que atende(m) ao padrão indicado em *specname*. A rotina *i_arqinicio* recebe a indicação do padrão a ser procurado, e dos atributos dos arquivos que devem atender a esse padrao. Essas indicações são armazenadas para serem usadas por chamadas posteriores de *i_arqproximo*, e retorna o primeiro arquivo encontrado (se houver). Chamadas posteriores de *i_arqproximo* irão retornando os arquivos seguintes que atendem ao padrão, até que seja indicado o erro 18, a partir do qual nenhum novo arquivo será achado, o que encerra a seqüência de buscas.

ERROS

0 - Tudo OK, arquivo encontrado;
2 - Path não encontrado
18 - Não existe nenhuma entrada que atenda ao padrão procurado.

EXEMPLO

```
cadeia  specname;
ArqSpec *cabecalho;

/* Escrever o nome de todos os arquivos do
diretório corrente do disco A: */
specname=kcvt ("-.a:*.*"); /* Conv. para formato
Cadeia */
erro = i_arqinicio (specname, 23, cabecalho)
while (erro == 0) {
      v_printk ("%a\n", cabecalho.fname);
      erro = i_arqproximo(cabecalho);
      }
```

NOME

i_close - Fecha um arquivo.
i_open - Abre um arquivo.
i_read - Le dados de um arquivo.
i_seek - Posiciona dentro de um arquivo.
i_write - Escreve dados em um arquivo.

SINOPSE

```
#include <data_str.h>

erro = i_close (file);
file = i_open (nome, modo);
erro = i_read(file, buffer, num);
erro = i_seek (file, offset, como);
erro = i_write (file, buffer, num);
```

`int file;`	Indice do arquivo aberto.
`cadeia nome;`	Nome do arquivo a ser aberto
`int modo;`	Modo de abertura do arquivo:
`int erro;`	Código de erro retornado pelo DOS para a operação.
`unsigned char *buffer;`	Endereco do buffer de transferência.
`int num;`	Número de bytes a serem transferidos.
`long offset;`	Número de posições (bytes) a deslocar o ponteiro.
`int como;`	Forma de deslocar o ponteiro.

DESCRIÇÃO

A rotina *i_close* fecha o arquivo indicado no hospedeiro. O arquivo é indicado através da variável *file*, obtida quando o arquivo foi aberto através de uma chamada anterior da rotina *i_open*.

A rotina *i_open* abre o arquivo indicado no hospedeiro, retornando um valor inteiro que identifica univocamente esse arquivo entre os arquivos correntemente abertos. O *modo* de abertura corresponde aos mesmos

valores de abertura das rotinas da biblioteca padrão de "C". A posição corrente do arquivo é o primeiro byte do arquivo. O *modo* de abertura indica:

0 - Abrir o arquivo para leitura;
1 - Abrir o arquivo para escrita;
2 - Abrir o arquivo para leitura e escrita.

A rotina *i_read* lê dados do arquivo no *buffer* indicado. Os dados são lidos desde a posição corrente do arquivo, até o máximo de *num* bytes. Caso essa quantidade de bytes ultrapasse o final do arquivo, apenas os bytes restantes são lidos. O número de bytes efetivamente lidos retorna como o valor de retorno da rotina. Caso esse número seja diferente de *num*, significa que o final do arquivo foi atingido. Após uma leitura, a posição corrente do arquivo passa a ser o primeiro byte seguindo ao último byte lido.

A rotina *i_seek* movimenta a posição corrente do arquivo indicado, o número de bytes indicado em *num*, de acordo com o parâmetro como:

0 - *num* bytes a partir do ínicio do arquivo;
1 - *num* (positivo ou negativo) bytes a partir da posição corrente;
2 - *num* bytes voltando a partir do final do arquivo.

A rotina *i_write* escreve dados no arquivo a partir do *buffer* indicado. Os dados são escritos da posição corrente do arquivo, até o máximo de *num* bytes. Os dados escritos sobrepõem aqueles já existentes no disco. Caso os dados ultrapassem o final anterior do arquivo, este é extendido. Caso essa quantidade de bytes esgote o espaço disponível do disco, apenas os bytes que cabem são lidos. O número de bytes efetivamente escritos retorna como o valor de retorno da rotina. Caso esse número seja diferente de *num*, significa que não foi possível escrever todo o buffer indicado. Após uma escrita a posição corrente do arquivo passa a ser o primeiro byte seguindo ao último byte escrito.

RETORNO

Valor de Retorno da Rotina *i_close* (tipo int):
ZERO: Fechou com sucesso.
-1: Não pode fechar o arquivo.

Valor de Retorno da Rotina *i_open* (tipo int):
-1: o arquivo não pode ser aberto.
valor positivo: indica o índice do arquivo aberto.

Valor de Retorno da Rotina *i_read* (tipo int):
ZERO: Fim de arquivo.
Valor positivo: número de bytes efetivamente lidos.

Valor de Retorno da Rotina *i_seek* (tipo int):
Valor positivo: posição corrente do ponteiro.
-1: pesquisa ilegal.

Valor de Retorno da Rotina *i_write* (tipo int):
Valor positivo: número de bytes efetivamente escritos.
-1: Houve erro de escrita.

NOTA:

Essas rotinas possuem utilização semelhante às rotinas de acesso a arquivos da linguagem "C".

NOME

i_getc - Obtém caráter do teclado do hospedeiro.
i_putc - Escreve um caráter no vídeo do hospedeiro.

SINOPSE

```
int carin = i_getc()
i_putc(caract)

unsigned char caract, carin;
```

DESCRIÇÃO

A rotina *i_getc* lê um caráter do teclado do hospedeiro IBM-pc. Caso não exista um caráter digitado, espera-se até que um caráter seja teclado. O valor *carin*, de retorno da rotina é o valor ASCII do caráter digitado.

A rotina *i_putc* escreve o caráter *caract* no vídeo do hospedeiro IBM-pc. Caso ainda exista um caráter anteriormente enviado ainda não recebido pelo hospedeiro, espera-se até que o caráter anterior seja escrito. O valor de retorno da rotina é o valor ASCII do caráter impresso.

RETORNO

Valor de Retorno da Rotina (tipo int):
Valor ASCII do caráter lido/escrito.

RESTRIÇÕES

• Podem ser digitados qualquer dos valores ASCII, que os mesmos serão recebidos pela rotina *i_getc*. No entanto, os valores ASCII que correspondem a movimentos do cursor serão tratados como tal pela rotina *i_putc*.

NOTA

Essas rotinas possuem utilização semelhante às rotinas tratamento de caracteres da linguagem "C".

EXEMPLO

```
/* Para ecoar todos os caracteres digitados até
      que seja teclado um caráter <^Z>:     */
int c;
while ( (c=i_getc() ) != 26) i_putc(c);
```

NOME

i_getk - Le uma cadeia de carateres do teclado do hospedeiro.

SINOPSE

```
#include <data_str.h>

int car = i_getk(cad, como)
int como;
cadeia cad;
int car;
```

DESCRIÇÃO

A rotina *i_getk* chama a rotina _vgetk() indicando que devem ser aceitos apenas os sinais de entrada do teclado, o que é realizado pela rotina i_getc().

A rotina _vgetk lê do teclado uma seqüência de caracteres, e a coloca na cadeia indicada. A cadeia pode ser editada durante a execução da rotina, através das seguintes teclas:

- <INS> - troca entre modo de inserção ou sobreposição;
- <DEL> - apaga o caráter sobre o qual o cursor está;
- <BackSpace> - apaga o caráter anterior ao cursor;
- <Seta Esquerda> - volta o cursor um caráter sem apagar;
- <Seta Direita> - avança o cursor para a direita uma posição;
- <HOME> - Coloca o cursor no início da cadeia;
- <END> - Coloca o cursor no final da cadeia e desabilita modo inserção;
- <F1> - Restaura o caráter seguinte da cadeia anterior;
- <F3> - Restaura a cadeia anterior. Note-se que a cadeia anterior vai sendo substituída por novos caracteres teclados.

Caracteres imprimíveis são colocados na cadeia. Caracteres não imprimiveis distintos das teclas de controle são ignorados. Os caracteres de controle <Seta Acima>, <Seta Abaixo>, <PgUp>, e <PgDn> podem ser considerados terminadores do processo de edição ou nao, dependendo do valor da variável *como*: se 0 não os considera terminadores; se 1 esses caracteres são terminadores. As teclas <RETURN>, <Line Feed> e <^C> sempre são consideradas como terminadores. A rotina devolve como valor de retorno o código do caráter que terminou a edição. Quando a cadeia é totalmente preenchida, os demais caracteres sobrepoem-se ao último anterior. Se o tamanho da cadeia for menor ou igual a zero, a rotina não é executada, e retorna com o valor -1.

RETORNO

Valor de Retorno da Rotina (tipo int):

-1 - Cadeia indicada tem tamanho nulo ou negativo.

>0 - Código ASCII do caráter que encerrou a entrada da cadeia.

Pode ser os caracteres:

<Line Feed>	(10)	
<Return>	(13)	
<^C>	(3)	
<Seta Acima>	(200)	se *como* == 0
<Seta Abaixo>	(208)	se *como* == 0
<PgUp>	(201)	se *como* == 0
<PgDn>	(209)	se *como* == 0

RESTRIÇÕES

- A cadeia deve ter capacidade para armazenar ao menos 1 caracter.
- Devem estar determinados os parâmetros básicos do Sistema de Janelas, tais como fonte corrente, e o mecanismo de comunicação com o hospedeiro.

ERROS

-1 - Cadeia nula ou de tamanho negativo.

VEJA TAMBÉM

v_insflag(3LIEGSP).

EXEMPLO

```
unsigned char cad[32]={30,0};
i_getk(cad, 0);
```

NOME

i_le_imagem - Lê no hospedeiro uma imagem para uma janela.

SINOPSE

```
#include <data_str.h>

int erro = i_le_imagem (file, xlo, ylo, titulo,
*nlin, *ncol);

int erro;          Valor de retorno da rotina.
int file;          Arquivo de onde será lida a
imagem.
int xlo, ylo       Posição relativa à janela
                   corrente, do canto superior
                   esquerdo da janela a ser aberta
                   para mostrar a imagem.
int *nlin, *ncol;  Tamanho da janela criada pela
                   rotina para armazenar a imagem.
cadeia titulo;     Cabeçalho a ser colocado na
janela.
```

DESCRIÇÃO

Esta rotina lê do arquivo indicado (que deve ser previamente aberto com a rotina *i_open*(3LIEGSP)) uma imagem no formato de imagens, obtendo o número de linhas (*nlin*) e colunas (*ncol*), e abrindo uma janela filho da janela corrente, nas coordenadas relativas à essa janela correspondente à *xlo* e *ylo*. A imagem é lida e transferida diretamente para essa janela. Note-se que a janela corrente deve ter tamanho suficiente para mostrar a imagem a partir da coordenada indicada, caso contrário a imagem será truncada. Caso a cadeia *titulo* não seja nula, seu conteúdo é mostrado no cabeçalho da janela criada. A janela criada retorna da rotina como a janela corrente.

RETORNO

Valor de Retorno da Rotina (tipo int):

-1: o arquivo não pode ser aberto.

valor positivo: indica o índice do arquivo aberto para a leitura da janela (*file*).

RESTRIÇÕES

- A janela corrente deve ter tamanho suficiente para mostrar a imagem a partir da coordenada indicada, caso contrário a imagem será truncada.
- O arquivo indicado deve ter uma imagem armazenada segundo o padrão de imagens.

ERROS

-1 - o arquivo indicado não está acessível, ou não está aberto.

retorno > 0 - código do arquivo indicado para leitura da imagem.

VEJA TAMBÉM

Jan_corrente(3LIEGJ), i_open(3LIEGSP).

EXEMPLO

```
/*  Mostrar a imagem armazenada no arquivo
      "minha.img" numa janela criada diretamente
      sobre a tela. */
#include <data_str>
extern lie_janela raiz;
int file, nl, nc;

file=i_open(kcvt("-.minha.img",0);
Jan_corrente (&raiz);
i_le_imagem (file, 50, 40, &nl, &nc,
                  kcvt("-.Minha Imagem") );
```

NOME

i_pollc - Verifica se há caráter teclado.

SINOPSE

```
int flag = i_pollc()
```

DESCRIÇÃO

A rotina *i_pollc* verifica se foi digitado um caráter no teclado do Hospedeiro IBM-pc. Caso não exista um caráter digitado, a rotina retorna com o valor ZERO (0). Caso exista um caráter teclado, o valor de retorno é o valor ASCII do caráter digitado, porém o caráter não é retirado do buffer.

RETORNO

Valor de Retorno da Rotina (tipo int):

0 - não existe caráter digitado.

>0 - Valor ASCII do caráter lido/escrito.

RESTRIÇÕES

- Apesar da rotina retornar o código do caráter digitado, este não é retirado do buffer, de maneira que chamadas sucessivas desta rotina sempre retornará o mesmo caracter. Para retirar o caracter, é necessário que seja chamada a rotina *i_getc*.

VEJA TAMBÉM

i_getc(3LIEGSP).

EXEMPLO

```
int c;
while (1)
      if (i_pollc() == 0)
            faz_outra_coisa();
         else
            trata_caracter(i_getc());
```

NOME

i_printk - Saída formatada de dados no vídeo do hospedeiro.
v_printk - Saída formatada de dados no vídeo do GSP.

SINOPSE

```
#include <data_str.h>

void i_printk (fmt, args)
void v_printk (fmt, args)
char *fmt;
char args;
```

DESCRIÇÃO

A rotina *i_printk* envia caracteres para o vídeo do hospedeiro IBM-PC.

A rotina *v_printk* envia caracteres para o vídeo do próprio TMS.

Ambas as rotinas usam o esquema de formatação da mesma maneira que a rotina *printf*(3).

VEJA TAMBÉM

i_putc(3LIEGSP), v_wpuc(3LIEGSP).

EXEMPLO

```
i_printk
    ("Isto é escrito no hospedeiro: %d\n",
     10);
v_printk ("Isto é escrito no TMS: %d\n",
          10);
```

NOME

POSICIONA - Posiciona o cursor na janela especificada.

SINOPSE

int Posiciona(jan, x, y)
int x, y;
lie_janela *jan,

DESCRICAO

Posiciona o cursor dentro da janela especificada - "jan".

RETORNO

Valor de Retorno da Rotina:
Erro (vide ERROS:).

RESTRICOES

A janela deve ter sido definida corretamente, e sua estrutura deve estar integra.

ERROS

0 - Tudo OK.
1 - Estrutura da janela indicada danificada.

VEJA TAMBEM

move_cursor(3LIEGJ).

NOME

init_sdb - Inicializa o GSP.

SINOPSE

```
int init_sdb()
```

DESCRIÇÃO

Esta rotina inicializa o processador gráfico GSP, de maneira a que as operações de manipulação da tela e dos recursos de E/S gráficas possam ser efetuadas. Esta rotina deve ser chamada uma única vez por qualquer programa, antes que qualquer operação de exibição ou comunicação deste pacote possam ser executadas.

Esta rotina é chamada pela rotina *Inicializa*(3LIESJ), a qual prepara também a estrutura de janelas para operação do pacote gráfico. Assim, esta rotina somente deverá ser chamada por programas que não utilizem a rotina *Inicializa*(3LIESJ), quando as operações de manipulação de janelas não forem necessárias, realizando-se diretamente o acesso à tela do TMS, sem utilização do Sistema de Janelas.

RETORNO

Não existe valor de retorno.

RESTRIÇÕES

- Esta rotina deve ser chamada uma única vez, antes de qualquer outra operação do sistema gráfico.
- Caso esta rotina seja chamada, a rotina *Inicializa*(3LIESJ) não pode ser usada, bem como o Sistema de Janelas.

EXEMPLO

```
main()
{
/* Declaração de variáveis */
 . . .
init_sdb();
 . . .
```


NOME

kcvt - Converte vetor para cadeia de caracteres.

SINOPSE

```
cadeia cad = kcvt (vet)
cadeia cad;
unsigned char *vet;
```

DESCRIÇÃO

Esta rotina determina o número de caracteres que existe em um vetor de caracteres desde a posição de índice 2 até a ocorrência do primeiro caráter <NULL>. Para isso é necessário que as posições de índice 0 e 1 contenham respectivamente os caracteres <-> e <>, os quais são substituídos pela rotina pelo número de caracteres determinado. Dessa maneira o vetor passa a representar uma cadeia no padrão CCMX&CO (Cadeia com campo máximo e corrente - Sistema GeO [TRA_89].

A finalidade desta rotina é converter para o formato padrão CCMX&CO um vetor inicializado como uma constante string pelo compilador.

Note-se que os dois primeiros caracteres devem ser <-> e <>, os quais são mudados para o tamanho máximo e corrente da cadeia, o que faz com que apenas a primeira chamada da rotina para um mesmo vetor seja efetiva. Chamadas consecutivas não encontrarão os caracteres <-> e <>, e serão rejeitadas, deixando a cadeia inalterada. Por outro lado, se a cadeia for modificada posteriormente, o valor da string original inicialmente inicializada pelo compilador passará a conter o valor modificado.

RETORNO

Valor de Retorno da Rotina (tipo int):

Ponteiro para a cadeia alterada, o qual é o mesmo ponteiro passado como o argumento da rotina.

RESTRIÇÕES

• O tamanho máximo da cadeia não pode ultrapassar 128 bytes, caso contrário esse será o tamanho considerado.
• A cadeia original, caso esteja com os caracteres <-> e <.> deve ser encerrada com um caráter <NULL>.

EXEMPLO

Em ambos os exemplos abaixo a cadeia "cad" passa a representar uma cadeia de tamanho máximo 7, tamanho corrente 6, contendo o valor "cadeia":

Exemplo 1:

```
char cad[]="-.cadeia";
 . . .
kcvt(cad);
```

Exemplo 2:

```
char *cad;
 . . .
cad=kcvt("-.cadeia");
```

NOME

move_rect - Move retângulo de pixels.

SINOPSE

```
void move_rect (ncol, nlin, xorg, yorg, xdest,
ydest)
long ncol, nlin;    Número de colunas e de linhas de
                    ambos os retângulos.
long xorg, yorg;    Coordenada do canto superior
                    esquerdo do retângulo origem.
long xdest, ydest;      Coordenada do canto
                        superior esquerdo do
                        retângulo destino.
```

DESCRIÇÃO

A rotina *move_rect* copia o retângulo origem especificado através de seu número de colunas *ncol* e de linhas *nlin*, cujo canto superior esquerdo está na coordenada de tela indicada por *xorg* e *yorg*, para o retângulo destino, de mesmo tamanho, cujo canto superior esquerdo está nas coordenadas de tela indicada por *xdest* e *ydest*.

RESTRIÇÕES

• Esta rotina opera na região de memória de tela, e portanto pode ser utilizada para a cópia de retângulos entre diferentes páginas de tela, porém não pode ser utilizada para a cópia de dados entre tela e matrizes de pixels.
• Como a cópia é efetuada diretamente pixel a pixel da origem para o destino, se houver sobreposição entre os retângulos origem e destino, o resultado pode ser imprevisível.

EXEMPLO

```
/* Copia um retângulo 20 linhas abaixo */
move_rect (10, 20,  20, 20, 40, 20)
```

NOME

pchar - Desenha um caráter na posição corrente do cursor.

SINOPSE

```
int pchar (caract)
char caract;
```

DESCRIÇÃO

A rotina *pchar* desenha um caráter na posição indicada pelo cursor, dentro da janela corrente, e segundo o fonte corrente. Uma vez desenhado o caracter, o cursor é deslocado na linha para que chamadas consecutivas da rotina desenhem os caracteres em seqüência. Os seguintes caracteres de controle são também interpretados:

<Back Space> - volta o cursor o espaço de um caracter;
<Line Feed> - Avança o cursor para a linha seguinte;
<Return> - Posiciona o cursor no início da linha corrente;
<Tab> - Posiciona o cursor numa posição múltipla de 8 caracteres na linha corrente, a frente da posição atual do cursor.

Antes do caráter ser escrito, a posição corrente do cursor é avaliada para verificar se está dentro da janela corrente. Caso esteja fora da janela, abaixo da mesma, considera-se a necessidade de um rolamento vertical da janela, para posicionar o cursor dentro da mesma. Caso o cursor esteja acima da janela, nenhuma ação é tomada, o mesmo ocorrendo caso o cursor esteja a direita ou a esquerda da janela.

RETORNO

Valor de Retorno da Rotina (tipo int):

A rotina sempre retorna a constante Zero.

ERROS

0 - Tudo OK.

NOTA

Esta rotina é usada internamente para as operações de saída de caracteres na tela do TMS, tendo portanto a intensão de ser utilizada como rotina interna do sistema.

VEJA TAMBÉM

EXEMPLO

```
/* Escrever todos os caracteres do fonte corrente,
      10 caracteres por linha: */
for (i=32; i<128; i++ )
      if ( 10*(int)(i/10)==i) {
            pchar (10); pchar (13)
            }
           else {
            pchar(i);
```

NOME

set_color0 - Estabelecer cor de fundo.
set_color1 - Estabelecer cor de frente.

SINOPSE

```
int set_color0 (cor)
int set_color1 (cor)

long cor;
```

DESCRIÇÃO

Estas rotinas determinam a cor respectivamente de fundo e de frente com que são desenhadas as primitivas gráficas e as letras nas chamadas subseqüentes. A cor é determinada em função do número de planos de cor do sistema corrente. Se o tamanho do pixel é de *n* bits, o padrão de cor deve ser replicado na cor 32/n vezes, até que se completem os 32 bits do valor da *cor*.

RETORNO

Valor de Retorno da Rotina (tipo int):
Valor de cor especificado.

RESTRIÇÕES

• Caso o valor não replique a cor ao longo dos 32 bits, cores diferentes serão alocadas em diferentes colunas de pixels, resultando em um padrão heterogêneo de cores.

EXEMPLO

```
/* Alocar a cor de frente 0x45, e a cor 0x8 para um
sistema com 8 planos de bits: */
set_color0 (0x08080808);
set_color1 (0x45454545);
```

NOME

v_insflag - Estabelece condição inicial para insersão/sobreposição de caracteres.

SINOPSE

```
int flagv = v_insflag(flag)
register short flag, vflag;
```

DESCRIÇÃO

A rotina *v_insflag* estabelece a condição inicial de chamada da rotina *i_getk*(3LIEGSP). Se *flag*=0, a rotina sobrepõe os caracteres digitados, caso contrário os caracteres são inseridos. Note-se que o conteúdo antigo da cadeia é sempre descartado, independentemente do valor de *flag*. O valor de retorno desta rotina é o valor de *flag* anterior. Note-se que o uso da tecla <INS> durante a execução da rotina *i_getk*(3LIEGSP) altera o valor do *flag*, mesmo para outras chamadas da rotina *i_getk*(3LIEGSP).

RETORNO

Valor de Retorno da Rotina (tipo int):
Valor anterior de *flag*.

NOTA

Existe um *flag* do sistema que registra a condição atual de sobreposição ou inserção. Esse valor é sobreposto pela chamada da rotina *v_insflag* ou através do uso da tecla <INS> quando a rotina *i_getk* está ativa.

VEJA TAMBÉM

i_getk(3LIEGSP)

EXEMPLO

```
/* ler uma cadeia com inserção ligada a
      princípio: */
v_insflag(1);
i_getk(cad, 0);
```

NOME

v_lugc - Obter a posição do cursor na janela corrente.
v_posic - Posicionar o cursor na janela corrente.

SINOPSE

```
void v_lugc (plin, pcol)
void v_posic (lin, col)

short *plin, *pcol;
short lin, col;
```

DESCRIÇÃO

A rotina *v_lugc* obtém a coordenada corrente do cursor na janela corrente, em coordenadas relativas a essa janela (em coordenadas de linha e coluna - posição de caracteres em fonte corrente).

A rotina *v_posic* coloca o cursor da janela corrente na posição indicada.

RETORNO

plin e *pcol* retomadas pela rotina *v_lugc*:
As variáveis apontadas por *plin* e *pcol* obtêm a posição corrente do cursor nessa janela.

EXEMPLO

```
/* Avançar o cursor 3 pixels na horizontal, e 5
pixels na vertical: */

short lin, col;
v_lugc (&lin, &col);
v_posic ((short(lin+5), short(col+3));
```

NOME

v_wputc - Escreve caráter na janela corrente.
v_wputk - Escreve cadeia de caracteres na janela corrente.
v_wputs - Escreve vetor de caracteres na janela corrente.

SINOPSE

```
int v_wputc(c)
v_wputs(str)
v_wputk(cad)

short c;
char *str;
cadeia cad;
```

DESCRIÇÃO

A rotina *v_wpuc* coloca o caráter *c* indicado no vídeo, na posição atual do cursor, na janela corrente.

A rotina *v_wputs* coloca o vetor de caracteres *str* indicado no vídeo, a partir da posição atual do cursor. Avança o cursor para o final do vetor escrito. Os caracteres são escritos com o atributo de vídeo corrente, na janela corrente.

A rotina *v_wputk* coloca a cadeia de caracteres *cad* indicada no vídeo, a partir da posição atual do cursor. Avança o cursor para o final da cadeia escrita. Os caracteres são escritos com o atributo de vídeo corrente, na janela corrente.

Os caracteres são colocados na janela corrente, na posição corrente do cursor, utilizando-se os atributos de vídeo corrente, tal como cores de frente e fundo, e o fonte de caracteres corrente.

RETORNO

Valor de Retorno da Rotina *v_wputc* (tipo int):

Código do caráter escrito.

Valor de Retorno das Rotinas *v_wputs* e *v_wputk* (tipo int):

Constante zero.

EXEMPLO

```
unsigned char cad[42]= {40, 3, 'a', 'b',
                        'c'};
v_wputc ((short)'>');
v_wputs
 ("Cadeia entrada pelo teclado:\n");
v_getk (cad,0); v_putk(cad);
```