Paginas 88 a 106

```
Unit Memory;

Interface

Uses Global, Disck;

Const Num_Pag_Mem = 32;

Type { estrutura do ponteiro para uma Página de memória}
Pont 2 = ^FliaMem;
FilaMem = Record
Pag,
PagLog,
Tabpag: integer;
Porx: pont2
end;

Var
{memória Pirncipal}
Memória: array [ 0..Num_Pag_Mem – 1, 0..7] of Palavra;

{ponteiros para paginas auxiliares e filas e filas de paginas de memória}
Fila_Mem_Livre,
Fila_substituiveis,
Fila_Nao_Substituiveis: Pont 2;

Nova_Pag,
Pag_Loader: Pont 2;

Num_Mem_Livre: integer; {Numero de paginas livres de memória}

{insere o ponteiro de uma pagina de memória em um das filas}
procedure Insere_Pag_Mem (var P2, Fila: Pont 2);

{procedimento que libera as paginas de memória alocadas por um job}
procedure Libera_Pag_Nem (NumTabPag : integer);

{transfere uma página de BCP na fila NFILA, do disco para a memória}
procedure Transf_Pag_Mem (FilaDisco : Pont3; TabPag, NumPag: integer

{aloca uma pagina de memória}
```

```
procedure Aloca_Pag_Mem (var 2: Pont2);

{apaga as informações da página de memória}
procedure Limpa_Pag_Mem (Num_Pag: integer);

{verifica se a fila de memória correspondente está vazia}
function Fila_Vazia_Mem ( Fila : Pont2) : Boolean;

Implementation

Function Fila_Vazia_Mem ( Fila: Pont2) : Boolean;
begin
Fila_Vazia_Mem:= Fila=Nil;
End; {Fila_Vazia_Mem}

Procedure Limpa_Pag_Mem ( num_pag: integer);
Var I: integer;
bengin
for I: =0 to 7
do with Memoria [Num_Pag, I] do
begin
Tipo: = ´ ´;
Operação: = ´        ´;
Campo1: = 0;
Pag: =0;
Desl: =0;
end;

end {Limpa_Pag_Mem}

procedure Insere_Pag_Mem (var P2, Fila : Pont2);
var Pont : Pont2;
bengin
P2^.Prox:=Nil;
If  Fila=Nil
Then Fila: = P2
else begin
Pont: = Fila;
While Pont^.Prox<>Nil
do Pont: = Pont^.Prox;
Pont^.Prox:P2;
End;
P2=Nil;
End; {Insere_Pag_Mem}

Procedure Libera_Pag_Mem (NumTabPag: integer);
Var Patras, P2: Pont2;
```

```
begin
P2: = Fila_Substuiveis;
Patras:= P2;
While(P2<>Nil) {Liberando paginas entre as substituíveis}
do if(P2^.TabPag = NumTabPag)
then begin
if P2= Fila_Substituiveis
then begin
        Fila_Substituiveis: Fila_Substituiveis^.PROX;

        Insere_Pag_Mem (P2, Fila_Mem_Livre);
        Num_ Mem_Livre: =Num_Mem_Livre +1;
        P2:= Fila_Substituiveis;
        PATRAS:=P2
        end{ then}
        else  begin
        Patras^.Prox: =P2^.PROX;
        Insere_Pag_Mem (P2, Fila_Mem_Livre);
        Num_Mem_Livre:= Num_Mem_Livre +1;
        P2:= Patras^.Prox;
        end;{else}
end
else begin
Patras:=P2;
P2:=P2^.Prox;
end; {else}

P2:= Fila_Nao_Substituiveis;
Patras: = P2;
While (P2<> Nil) {Liberando paginas entre as não substituíveis}
then begin
Fila_Nao_Substituiveis:= Fila_Nao_Substituiveis^.Prox
Insere_Pag_Mem (P2, Fila_Mem_Livre);
P2:= Fila_Nao_Substituiveis;
Patras:=P2;
Num_Mem_Livre:= Num_Mem_Livre+1;
End {then}
Else begin
Patras^.Prox:=P2^Prox;
Insere_Pag_Mem(p2, Fila_Mem_Livre);
P2:=Patras^Prox;
Num_Mem_Livre:= Num_Mem_Livre + 1;
End; {else}
End
else begin
Patras:=P2;
P2:=P2^.Prox;
```

```
End; {else}
P2: =Nil;
Patras:=Nil;
end; {Libera_Pag_Mem}

procedure Transf_Pag_Mem (FilaDisco: Pont3; TabPag, NumPag: Integer);
var PagDisco, PagMem, I: integer;
P3: Pont3;
bengin
P3:= FilaDisco;
{avanca ponteiro de indicações para acessar pagina fisica de disco}
for I:= 1 to NumPag
do P3:= P3^.Prox;

PagDisco:=P3^.Pag;
PagMem:=Memória[TabPag, Num].Pag;

{Transfere pagina de instrução do disco para a memoria}
for I:= 0 to7 do
begin
Memoria [PagMem, I].Tipo:=Disco[PagDisco, I].Tipo;
Memoria [PagMem, I].Operacao:=Disco[PagDisco, I].Operacao;
Memoria [PagMem, I].Campo1:=Disco[PagDisco, I].Campo1;
Memoria [PagMem, I].Pag:=Disco[PagDisco, I].Pag;
Memoria [PagMem, I].Tipo:=Disco[PagDisco, I].Desl;
end;
end; {Transf_Pag_Prog_Mem}

procedure Aloca_Pag_Mem (var P2: Pont2);
begin
if not Fila_Vazia_Mem ( Fila_Mem_Livre) {existem Paginas livres}
then begin
P2:= Fila_Mem_Livre;
Fila_Mem_Livre:=Fila_Mem_Livre^.Prox;
End

else if not Fila_Vazia_Mem (Fila_Substituiveis)
then begin  {aloca pagina substituível}
P2:= Fila_Substituiveis: Fila_Substituiveis^.Prox;
{atualiza ausência da pagina na tabela de paginas}
Memória [P2^.TabPag, P2^.PagLog]. Campo1: =1;
End
Else P2: = Nil;
end; {Aloca_Pag_Mem}

Pocedure Inicia_Memoria;
var I: integer;
```

```
Pont: Pont2;
begin
Fila_Mem_Livre: = Nil;
Fila_Substituiveis: = Nil;
Fila_Nao_Substituiveis: = Nil;
Pag_Loader:= Nil;
Tab_Loader:= Nil;
Nova_Pag:=Nil;

for I:= 0 to (Num_Pag_Mem -1) do
begin
new(Pont);
Pont^.Pag:=I;
Limpa_Pag_Mem(Pont, Fila_Mem_Livre);
end;

Num_Mem_Livre:=Num_Pag_Mem;
end; {Inicia_Memoria}
begin
Inicia_Memoria;
End.

Unit L_R_U;

Interface

Uses Global, Memory;
Type{Estrutura da fila de ponteiros para o critério LRU}
PontLRU = ^ LRU;
LRU = Record
      Pag: integer;
      Prox: PontLRU
     end;

Var{Ponteiros para a fila LRU}
FilaLRU: PontLRU; {Fila auxiliar (apoia ao processo LRU)}

procedure Insere_LRU (pag:intereger);
procedure Atualiza_LRU;

Implementation

Procedure Insere_LRU ( Pag : integer);
VarPont, AUX: PontLRU;
begin
Pont: = FilaLRU;
AUX:= Pont;
```

```
While(Pont <>nil) and (Pont^.Pag<>Pag) do
begin
AUX:=Pont;
Pont:=Pont^.Prox;
end;
if Pont<> nil {pag já existia na Fila}
then if Point = FilaLRU
then FilaLRU:=Pont^.Prox
else AUX^.Prox:=Pont^.Prox
else begin {aloca novo ponteiro}
new(Pont);
Pont^.Pag:=Pag;
end;
{insere Ponteiro no fim da Fila}
Pont^.Prox:= nil;
If FilaLRU=nil
then FilaLRU:=Pont
else begin
    Aux:=FilaLRU;
    while Aux^.Prox<>nil
    do Aux:= Aux^.Prox;
   Aux^.Prox:=Pont;
    end;
Pont:=nil;
Aux:=nil;
end; {Insere_LRU}

procedure Atualiza_LRU;
var Pont, ANT: Pont2;
AUX: PontLRU;
begin
  while FilaLRU<>nil do{ha pag na Fila}
   begin
   Pont:=Fila_Substituiveis;;
   while (Pont<>nil) and (Pont^.Pag <> FilaLRU^.Pag) do
   begin
   ANT:=Pont;
   Pont:= Pont^.Prox
   end;

 if (Pont <>nil) {pag presente na Fila}
then begin
       if Pont= Fila_Substituiveis
       then Fila_Substituiveis:= Pont^.Prox;
       Pont^.Prox:= nil;
       Insere_Pag_Mem (Pont, Fila_Substituiveis);
       end;
```

```
{Elimina Pag da Fila do LRU}
AUX:= FilaLRU;
FilaLRU:=FilaLRU^.Prox;
Dispose (AUX);
end;
end; {Atualiza_LRU}
begin
FilaLRU:= nil;  {iniciação da Fila auxiliar de Páginas}
End. {L_R_U}

Unit Leitura;
Interface

Uses Crt, Global, Buffer;

Procedure Leitura_Inicio;
Procedure Leitura_Fim;

Implementation

{Preenche o buffer com as informções lidas no cartão (arquivo de entrada)
procedure Leitura_Inicio;
begin

if(NumBuffLivre > 1) and {Existe buffer livre}
   (RC.Leitura = 0) and       {Leitura disponível}
   (not eof (Entrada))          {Ha dados de entrada no dispositivo}

then begin
   if Informa
   then begin
        Writeln (Acompanhamento, ´Leitura Inicio');
         Delay (Tempo_Espera);
         end;
 Executou=true;

{Aloca um buffer Livre}
BufLeitura:=Pega_Buffer (Fila_Buffer_Livre);
NumBuffLivre:=NumBuffLivre – 1;

{Ativa Leitora para leitura de dado do dispositivo de entrada
Comanda(1,1);
end;
end; {Leitura_Inicio}

procedure Leitura_Fim;
```

```
begin

if ProcSimples [1]   {2 parte do processo assinalada}
then begin
   if Informa
   then begin
WriteLn (Acompanhamento, 'Leitura fim');
Delay (TempoEspera);
end;
ProcSimples [1]: =false;
Executou:= true;

{preenche o buffer como o conteudo do cartão}
Read (ENTRADA, BufLeitura^.Linha);

{ins buffer na Fila SPOOl_IN}
Insere_Buffer (BufLeitura, Fila_Buffer_Spool);
BufLeitura:= nil;
end;
end; {Leitura_Fim}
end{Leitura}


Unit Sp_In;

Interface

Uses Ctr, Global, B_C_P, Buffer, Memory, Disk;
{faz a interpretação dos Buffers, e quando consegue interpretá-los com sucesso
insere o programa na fila dos programas residentes no disco
Procedure Spool_In_Inicio;

Procedure Spool_In_Fim;

Var

Pag_Cod,                {nro de paginas de codigo do programa do usuário}
Pag_Dado,               {nro de paginas de Dados do programa do usuário}
Pag_Rasc,               {nro de paginas de Rascunho do programa do usuário}
Pag_Impr: integer       {nro de paginas de Impressão do programa do usuário}

Estado_Spool_In: 1..5;

Implementation

Procedure Spool_In_Inicio;
{procedimentos locais}
```

```
{Atualiza estados quando ocorre erro na entrada do programa}
procedure Erro_De_Entrada;
begin

Libera_Buffer (BufEntrada);              {Libera Buffer lido}
Estado_Spool_In:=1;                      {Atualiza Estado de Spool_In}
Fila_BCP [1]^.CC[12]:=true;              {marca Erro_De_Entrada do CC do prog}

Troca_Fila (1,7);                        {Insere prog na Fila de Jobs Terminados}
end; {Erro_De_entrada}

{aloca paginas de rascunho depois das paginas de codigo do programa}
procedure Aloca_Pag_Rasc;
var I: integer;
P3: Pont 3;
begin

with Fila BCP[1]^
do for I:= 1 to Pag_Rasc do
 begin
P3:= Aloca_PagDisco;
Insere_Pag_Disco (P3, InicProg);
end;
P3:=nil;
{Ativa disco para a escrita de pagina}
Comanda (3,1);
end;{Aloca_ Pag_Rasc}

{Aloca paginas de impressão em disco para o programa}
procedure Aloca_Pag_Impr;
Var I: intereger;
P3: Pont 3;
begin

with Fila_BCP[1]^do
begin
for I:= 1 to Pag_Impr do
begin
P3:= Aloca_Pag_Disco;
Insere_Pag_Disco (P3, InicImpr);
end;

CorrImpr:= InicImpr; (atualiza pont pag corrente impressão)
End;
P3:= nil;
{Ativa disco para escrita de página}
Comanda(3,1);
```

```
end; {Aloca_Pag_Impr)

{procedimento que faz a análise do cartão de Identificação do programa}
procedure Espera_Job;
begin
Troca_Fila(0,1); {aloca um BCP disponível}
Whit BufEntrada^do
If( linha [0].Tipo<> '*') or (Linha [0].Campo1<>0)
Begin {erro no codigo da pagina de controle}
Libera_Buffer(BufEntrada);
Libera_BCP(1);
end

else
begin
{le Indentificador do programa}
Fila_BCP[1]^.Indentif:=Linha[1].campo1;

if (Linha[2].Tipo<> '*') or (Linha[2].campo1 < Tempo_Minimo)
then Erro_De_Entrada  {erro no tempo de execução do prog}

else
begin
{le tempo previsto p/ execução do prog}
Fila_BCP[1]^.Timer:=Linha[2].Campo1;
if (Linha[3].Campo1 < 1)
then Erro_De_Entrada  {Erro nro pag codigo}

else
begin
{le numero de pag do prog}
Fila_BCP[1]^.TamProg:=Linha[3].Campo1;

If (Linha[4].Tipo<>'*') or (Linha [4].Campo1<0

Or
(Linha[4].Tipo<>'*') or Linha[4].Campo1 > 7)
then Erro_De_Entrada {Erro nro pag rascunho}
else
begin
{le nro do rascunho}
PAG_RASC:= Linha[4].Campo1;

If (Linha [5].Tipo<>'*')
or (linha[5].Campo1>30)
or (linha[5].Campo1<0)
then Erro_De_Entrada {erro nro pag Impressão}
```

```
else
if Linha[5].Campo1<>0
then            {prog necessita pag impr}

if(Num_Disco_Livre> Linha[5].Campo1
then       {Existem pag disco suficientes}

begin
{le nro pag impressão}
Pag_Impr:=Linha[5].Campo1;
Aloca_Pag_Impr;

Estado_Spool_In:= 2;
{atualiza contadores pag}
Pag_Cod:=0;
Pag_Dado:=0;
end

else{ não há pag disco suficiente}
begin
Libera_BCP (1);
{Devolve Buffer p/ ser reinterpretado até que haja pag}
Devolve_Buffer(BufEntrada, Fila_Buffer_Spool);
end
else { prog não necessita pag impr}
begin

Libera_Buffer(BufEntrada);
Estado_Spool_In:=2;
{atualiza contadores pag}
Pag_Cod:=0;
Pag_Dado:=0;
end
end {else}
end {else}
end {else}
end {else}
end; {Espera_Job}
{Espera o cartão de controle de Inicio das paginas de programa}
procedure Espera_Prog;
begin
if BufEntrada^.Linha[0].Tipo´*´ {cartão de controle}
then case BufEntrada^.Linha[0].Campo1 of

0: begin    {inicio de novo job}
{Buffer a ser interpretado}
```

```
Devolve_Buffer(BufEntrada, Fila_Buffer_Spool);
{erro na intrepretação do job que estava tratando}
Fila_BCP[1]^.CC[12]:=true;
{insere prog na fila dos Jobs terminados}
Troca_Fila (1,7);
Estado_Spool_In:= 1;
end;

1: begin {inicio do código}
Libera_Buffer (BufEntrada);
Estado_Spool_In:=3;
end;

else Erro_De_Entrada; {cartão não esperado}
end {case}

else Erro_De_Entrada; {Cartão não Esperado}
end; {Espera_Prog}

{procedimento que encadeia as paginas de programa do usuário}
procedure Espera_Dados_Ou_Fim;
var I. PAGD: integer;
P3: PONT3;
begin
Case BufEntrada^.Linha[0].Tipo of

'*': {cartão de controle}
case BufEntrada^Linha[0].campo1 of

0: begin  {inicio de novo job}
            {Buffer será interpretado}
Devolve_Buffer (BufEntrada, Fila_Buffer_Spool);
{erro na interpretação do Job que estava tratando}
Fila_BCP[1]^.CC[12]:=true;
{insere programa na fila de Jobs terminados}
Troca_Fila(1,7);
Estado_Spool_In:=1;
End;

1: { Inicio pag codigo}
if Fila_BCP[1]^.InicProg<>nil { já leu pag codigo }
then Erro_de_Entrada
else Libera_Buffer ( BufEntrada ); { fica no estado }

2: { Inicio pag dado }
if Fila_BCP[1]^.InicProg<>nil { já leu pag codigo }
then begin
```

```
        Estado_spool_In :=4;

        If ( Fila_BCP[1]^;TamProg = Pag_coid+pag_Rasc }
          Then { nro correto pag codigo e rascunho }
            Begin
              If Pag_Rasc > 0 { requer pag rascunho }
                    then if Num_Disco_livre >= PagRasc
                          Then { há pag disponiveis }
                                Begin
                                      Aloca_Pag_Rasc;
                                      Estado_spool_In:=4;
                                End
                          Else { não há pag disponiveis }
                                Devolve_Buffer ( BufEntrada, Fila Buffer_Livre)
      Else { não requer pag rascunho }
        Begin
              Libera_Buffer { BufEntrada);
              Estado_spool_in:=4;
        End
End { then}
Else Erro_de_entrada; { nro incorreto de pag}
end {then}
Else Erro_de_entrada; { não leu pag codigo }

3: { Fim de Job }
if Fila_BCP[1]^.InicProg<>nil
ithe { já leu pag codigo }
begin
if ( Fila_BCP[1]^.TamProg = Pag_Cod+PagRasc)
then { nro correto pag codigo e rascunho }
begin
if Pag_Rasc >0
then { requer pag rascunho }
if Num_disco_Livre >=Pag_Rasc
then { há pagina disponivel }
begin
Aloca_pag_Rasc;
{ insere prog na fila dos }
{ prog residentes em disco }
Troca_Fila ( 1,2);
Etado_Spool_In :=1;
End
Else { não há pag disponeiveis }
Devolve_Buffer ( Bufentrada, Fila_Buffer_Spool )
Else { não requer pag rascunho }
begin
Libera_Buffer ( BufEntrada )
```

```
{ insere prog na fila dos}
{ prog residentes em disco }
Troca_Fila(1,2);
Estado_Spool_In:=1;
End
End { then }
Else Erro_de_Entrada; { nro incorreto pag}
end { then }
Else Erro_de_Entrada; { não leu pag codigo}
End ; { case }
´C´ : { pagina de codigo }
if Pag_Cod<8 { nro limute de pag }
then begin
P3:= Aloca_Pag_Disco;
PagD:= P3^.Pag: { nro da pagina no disco }
{ insere  pag na fila de pag de codigo do prog }
with Fila_BCP[1]^
Do Insere_Pag_Disco ( P3, InicProg );
P3 = nil;
{atribui conteudo do Buffer p/ pag do disco }
with BufEntrada^
do for I := 0 to 7
do Disco [ PagD,I ] := Linha[I];
{ incrementa nro de pag codigo do prog }
Pag_Cod := Pag_Cod +1;
{ Ativa disco para escrita de pagina }
Comanda ( 3,1 );
end { then }
else erro_de_entrada; { nro pag fora do limite }
else Erro_de_entrada;
end; { case }
end; {Espera_Dados_Ou_Fim }
{ procedimento que interpreta a pagina de dados do usuario }
procedure Espera_Fim;
        var I, PagD: integer;
        P3:Ponts3;
Begin
Case BufEntrada^.Linha[0].Campo1 of
0: { inicio de novo Job }
begin
{ Buffer sera reinterpretado }
Devolve_Buffer ( BufEntrada, Fila_Buffer_Spool );
{ erro na interpretacao do Joob que estava tratanto }
Fila_BCP[1]^.CC[12] := true ;
{ insere prog na fila dos Jobs terminados }
Troca_Fila(1,7);
Estado_Spool_In := 1;
```

```
End;

1: { inicio pag codigo }
erro_de_entrada;

2: { Inicio pag dados }
of Fila_BCP[1]^.INICDADOS<> nil
the Erro_de_Entrada { já leu pag dados }
else Libera_Buffer ( BufEntrada);

3: { Fim de Job }
begin
{ atualiza pont pag dados }
Fila_BCP[1]^.CorrDados := Fila_BCP [1]^.InicDados;
{ insere prog na fila dos progs residentes em disco }
Troca_Fila (1,2);
Estado_Spool_In := 1;
End;
End; { case}

´D´ : { pagina de dados }
if Pag_dado < 8 { nro limite pag }
then begin
P3 := Aloca_Pag_Disco;
PagD := P3^.Pag; { nro pag no disco }
{ insere pag na fila de pag de dados do prog }
With Fila_BCP[1]^.
 Do Insere_Pag_Disco ( P3, InicDados );
P3 := nil;
{ atribui conteudo do Buffer p/ pag do disco }
with BufEntrada^
 do for I := 0 to 7
 do Disco [ PagD,I ] := Linha[I];
{ incrementa nro de pag dados no prog }
Pag_Dado := Pag_Dado +1 ;
{ Ativa disco para escrita de pagina }
Comanda ( 3,1);
end
else Erro_De_Entrada; { nro pag fora do limite }
else Erro_De_Entrada; { cartao não esperado }
end; { case }
end; { Espera_Fim }
begin { Spool_in_Inicio }
if      ( RC.Disco = 0 ) and
        ( not Fila_Vazia_Buffer ( Fila_Buffer_Spool ) ) and
        ( nor Fila_Vazia_Disco ( Fila_Disco_Livre ) ) and
        ( ( not Fila_Vazia_BCP ( 1 ) or ( not Fila_Vazia_BCP ( 0 ) ) )
```

```
then begin
if Informa
then begin
writen ( Acompanhamento, ´Spool In inicio´ );
Delay ( TempoEspera );
End;
Executou := true;
{ Pega um buffer de fila dos buffer p/ Spool_In }
BufEntrada := Pega_Buffer ( Fila_Buffer_Spool );

Case Estado_Spool_In of
1: Espera_Job;
2: Espera_Prog;
3: Espera_Dados_ou_Fim;
4: Espera_Fim;
end;
end;
end; { Spool_ in_Fim }
procedure Spool_In_fim;
begin
 if ProcSimples[3] { 2 parte do Processo Assinalada }
then begin
if Informa
then begin
writen ( Acompanhamento, ´Spool in Fim´ );
delay( Tempo_espera );
end;
Executou := true;
ProcSimples [3] := false;
Libera_Buffer ( BufEntrada );
End;
End; {spool_in_fim}
begin
Estado_Spool_In :=1;
End. { Sp_In}
```