3.4 – Listagem do código dos programas implementados

```
(* UNIT contendo os tipos e variáveis de utilização geral *)

Unit Global;

Interface

        procedure Comanda ( Canal, Processo : integer );

{ declaracoes pre-definidas }

const Tempo_Mínimo = 10;   { tempo mínimo de execucao de um job }

type
        tipo_op = string [ 3 ];          { tipo do campo do minemonico da instrucao }

        { estrutura da Palavra }
        Palavra = record
                Tipo : Char;             { identifica o tipo da palavra }
                Operacao : Tipo_Op;      { minemonico da instrucao }
                Campo1,                  { instrucao/dado/tabela de Pagina }
                Pag,
                Desl : integer;
            end;

{ estrutura de uma Pagina }
Pagina = array [ 0..7 ] of Palavra;

{ tipo do codigo de condicao }
Tipo_CC = array [ 0..12 ] of boolean;
        { 0 - protecao de memoria
          1 - codigo dee operacao invalido
          2 - overflow
          3 - falta Pagina
          4 - timer
          5 - time slice
          6 - leia
          7 - imprima
          8 - pare
          9 - canal 1 - leitura
       10 - canal 2 - impressao
       11 - canal 3 - disco
       12 - erro na entrada do Job
```

```
Var         { declaracoes das variaveis globais }

  { vetor de processos simples - segunda parte }
  ProcSimples : array [ 1..7 ] of boolean;
                { 1- leitura
                      2- impressao
                      3- spool in
                      4- spool out
                      5- Paginacao
                      6- loader
                      7- esusuario }


  CC : Tipo_CC;   { codigo de condicao : vetor de interrupcoes }

  Interrupcao : -1..12;   { nro da interrupcao ocorrida }

  { contador de tempo para os canais }
  T : array [ 1..3 ] of integer;

  { tempo padrao para cada canal }
  TPadrao : array [ 1..3 ] of integer;
              { 1- leitura
                2- impressao
                3- disco }

  TS_Padrao : integer;  { fatia de tempo de execucao de cada programa }

  { registrador de comandos }
  RC : record
              Leitora   : 0..1;
              Impressao : 0..1;
              Disco     : 0..5
             end;                    { Disco :  0 - livre
                                                1 - spool in
                                                2 - spool out
                                                3 - loader
                                                4 - Paginacao
                                                5 - E/s usuario }

  { registrador de tabela de Paginas }
  TP : record
              Tam : 1..8;          { nro de Paginas do programa }
              Pag : 0..31;  { endereco da tabela de Paginas }
             end;
```

```
  { contador de programa }
  CP : record
            Pag,                    { Pagina logica do programa }
            Desl : integer  { deeslocamento dentro da Pagina }
          end;

  RI : Palavra;        { registrador de instrucao }
```



```
  PPagEf,                { Endereco da Pagina efetiva que contem a instrucao }
  PPagOp : Palavra;  { End Pag referida pela instrucao : contem o operando }

  ACC,                          { acumulador }
  FP,                           { indica a Pagina rm falta no programa }
  Timer,                        { timer }
  CK,                           { relogio simulador do clock do sistema }
  TS : integer;                 { time-slice }

  Forever,              { condicao de fim do programa supervisor }
  Executou : boolean            { indica execucao de um processo simples }

  Tipo_Pag_Falta : char;        { tipo da pagina em falta no programa do usuario }

  Entrada : file of Pagina;     { arquivo de entrada }

  Saida : text;                     { arquivo de saida }

  Acompanhamento : text;  { Arquivo para impressao de mensagens de }
                                { acompanhamento do programa }

  informa : boolean;{ Condicao para impressao de mensagens }
                                { de acompanhamento do programa }

  TempoEspera : integer;        { Retardo na impressao de msg de acompanhamento }


Implementation


{ simulacao do tempo de funcionamento de um periferico }
{ Canal : nro do periferico; Processo : nro do Processo requisitante }
procedure Comanda ( Canal, Processo, integer );
  begin
```

```
    case Canal of
     1 : RC.Leitora   := Processo;
     2 : RC.Impressao := Processo;
     3 : RC.Disco     := Processo;
    end;

    T [ Canal ] := CK + TPadrao [ Canal ];

   end; { Comanda }
```



```
{ procedimento que faz as iniciacoes gerais do SOS e do HARDWARE }
procedure Iniciacao;
  var I, J : integer;
  begin

    Forever := false;     { condicao de execucao do programa }

   { Iniciacao de Processos simples }
   for I := 1 to 7
     do ProcSimples [I] := false;

   { Iniciacao dos tempos de operacao perifericos }
   TPadrao [1] := 20;
   TPadrao [2] :=20;
   TPadrao [3] :=6;

   TS_Padrao    :=20;

   { Iniciacao dos canais }
   for I := 1 to 3
   do T [I] := 0;

   { Iniciacao do Registrador de Controle }
   with RC do
   begin
   Leitora   := 0;
   Impressao := 0;
   Disco     := 0;
end;

   { Iniciacao do array de condicao de interrupcao (CC) }
   for I := 0 to 12
      do CC [I] := false;
```

```
  { Iniciacoes necessarias }
  ACC          := 0;
  Timer      := 0;
  TS             := 0;
  CK             := 0;
  FP             := 0;
  CP.Pag     := 0;
  CP.Desl    := 0;
  RI.Campo1  := 0;
  RI.Pag     := 0;
  RI.Desl    := 0;

  with PPagOp do
  begin
  Campo1  := 0;
  Pag     := 0;
  Desl      := 0;
end;
```

Pag.76

```
     with PPagEf do
            begin
              Campo1  := 0;
              Pag     := 0;
              Desl    := 0;
            end;

    end;  { Iniciacao }

begin

  Iniciacao;

end. { Global }

Unit B_C_P;
Interface
Uses Global, Disk;
Const Num_BCP = 7;
Type
{ estrutura de um BCP }
Pont1 = ^BCP;
BCP = record
Identif, {identificador do programa }
EndTabPag, {endereco da tabela de Pagina }
```

```
ACC, {acumulador}
Timer, { timer }
TS : integer; { time slice }
FP : 0..7; { falta Pagjna }
TamProg : 1..8; { tamanho do programa }

CP: record { contador de programas }
Pag, { endereco da Pagina }
Desl : integer { deslocamento }
end; { record }

InicProg, { Pagina inicial do programa no disco }
InicDados, { Pagina inicial dos dados no disco }
CorrDados, { Pagina atual dos dados no disco }
InicImpr, { Pagina inicial da impressao no disco }
CorrImpr : Pont3; { Pagina atual da impressao no disco }

CC : Tipo_CC; { codigo de condicao }

Lig : Pont1; { ligacao pra outros BCP's }

end;

Var
{declaracao da fila de BCP's }
Fila_BCP : array [ 0..8 ] of PONT1; { ponteiro pra a fila de BCP's}
{ 0: livres
1 : programa em spool in
2 : programas residentes em disco
3 : programas residentes em memoria : prontos para execucao
4 : programa em execucao
5 : programas aguardando E/S
6 : programas aguardando Paginacao
7 : programas acabados
8 : programa em spool out }
79

{ apaga as informacoes contidas no BCP }
Procedure Zera_BCP ( Var P1 : Pont1 );

{ procedimento que limpa e libera BCP´s das filas }
Procedure Libera_BCP ( NroFila : integer );

{ transfere BCP do comeco da fila X para o fim da fila Y }
Procedure Troca_Fila (x, Y : integer );

{ insere o BCP no comeco da fila dos prontos quando o job interrompido }
```

```
{ tinha um time-slice <=5 }
Procedure Insere_Comeco;

Procedure Insere_BCP ( Var P1, Fila : Pont1 );

{ verifica se a fila de BCP´s correspondente esta vazia }
Functin Fila_Vazia_BCP ( Nro Fila : integer ) : boolean;

Implementation

Procedure Zera_BCP ( Var P1 : Pont1 );
        Var I   : integer;
        Begin

                with P1^do
                        Begin

                                InicProg := nil;
                                InicDados := nil;
                                CorrDados := nil;
                                InicImpr := nil;
                                CorrImp := nil;

                                CP.Pag          := 0;
                                CP.Desl         := 0;
                                FP              := 0;
                                Acc             := 0;
                                Timer           := 0;
                                TS              := 0;
                                TamProg         := 1;
                                EndTabPag       := 0;
                                Identif         := 0;

                                for I := 0 to 12
                                        do CC [ I ] := faise;
                        End;

        End; { Zera_BCP }
```



```
Procedure Insere_Comeco;
        Begin

                Fila_BCP [ 4 ]^.Lig := Fila_BCP [ 3 ];
                Fila_BCP [ 3 ] := Fila_BCP [ 4 ];
```

```
                Fila_BCP [ 4 ] := nil;

        End; { Insere_Comeco }


Procedure Insere_BCP ( Var P1, Fila : Pont1 );
        Var Pont : Pont1;
        Begin

                P1^.Lig := nil;

                If Fila = nil
                        then Fila := P1
                        else Begin
                                Pont := Fila;
                                while Pont^.Lig <> nil
                                        do Pont := Pont^.Lig;
                                Pont^.Lig := P1;
                        End;

                P1 := nil;

        End; { Insere_BCP }

Procedure Troca_Fila (X, Y : integer );
        Var Pont : Pont1;
        Begin

                If not Fila_Vazia_BCP ( X )
                        then Begin
                                Pont := Fila_BCP [ X ] := Fila_BCP [ X ]^.Lig;
                                Insere_BCP ( Pont, Fila_BCP [ Y ] );
                        End;

        End; { Troca_Fila }

Procedure Libera_BCP (NroFila : integer );
        Begin

                Zera_BCP ( Fila_BCP [ NroFila ] );
                Troca_Fila ( NroFila, 0 );

        End; { Libera_BCP }
```



```
Function Fila_Vazia_BCP ( NroFila : integer ) : boolean;
```

```
        Begin

                Fila_Vazia_BCP := Fila_BCP [ NroFila ] = nil;

        End; { Fila_Vazia_BCP }

Procedure Inicia_BCP;
        Var I : integer;
                Pont : Pont1;
        Begin

                for I := 0 to 8
                        do Fila_BCP [ I ] := nil;

                for I := 1 to Num_BCP do
                        Begin
                                new ( Pont );
                                Zera_BCP ( Pont );
                                Insere_BCP ( Pont, Fila_BCP [ 0 ] );
                        End;

        End; { Inicia_BCP }

Begin

        Inicia_BCP;

End.

Implementation


Procedure limpa_buffer (var p4 : Pont4 );
  var I : integer ;
  begin

with p4 ^do
 begin Tipo : =
 N : = 0;
 for I : = 0 to 7 do
   begin
      Linha [I]. Tipo         :=’  ‘ ;
      Linha [I] . Operacao : = ‘      ‘   ;
      Linha [I] . Campo 1  :=  0 ;
      Linha [I] . Pag         :=  0 ;
      Linha [I] . Des1        :=  0 ;
        end;
```

```
      end;
end;
{  Limpa_Buffer  }

Procedure  Insere_Buffer (var  p4 , Fila : Pot4);
  var Pont : Pont4 ;
  begin

     P4 ^. Prox : = nil

     if Fila = nil

       the Fila : = P4
      else begin
                Pont :  - Fila ;
                while Pont ^. Prox <> nil
                  do Pont : =Pont ^. Prox ;
                 Pont ^. Prox : = P4
              end;
     P4 := nil ;

end {Insere_Buffer }
Function Pega_Buffer (var Fila : Pont4 ) : Pont4;
     begin
       Pega_Buffer := Fila ;
       if Fila <> nil ^. Prox;
end; {Pega_Buffer}

procedure Libera_Buffer v (var P4 : Pont4);
   begin
       Limpa_Buffer (P4) ;
       Insere_Buffer (P4 ,Fila_Buffer_Livre);
       NumB uffLivre : =NumBuffLivre +1;
       P4 := nil /
    end ; { Libera_ Buffer }

procedure Devolve_Buffer (var P4 , Fila :Pont4 );
   begin
     P4 ^.Prox : = Fila ;
     Fila :=P4;
     P4: nil;
  end ; { Devolve_Buffer }

function Fila_Vazia_Buffer (Fila : Pont4 ) : boolean;
  begin
     Fila_Vazia_B uffer }
```

```
Procedure Inicia_B uffer ;
  var I : integer ;
      Pont ; Pont4;
Begin

     Fila_Buffer_Livre     : = nil ;
     Fila_Buffer_Impr      : = nil ;
     Fila_Buffer_Spool     : = nil ;
     BufLetura     : = nil ;
     BufEntada     : = nil ;
     BufSaida      : = nil ;
     BufImpr       : = nil :

for I : = 1 to Num_Buffer do
   begin
       new (Pont);
       Limpa_Buffer (Pont);
      Insere_buffer (Pont, Fila_Buffer_Livre );
  end;
NumBuffLivre := Num_Buffer

End ;{Inicia_Buffer }

Begin

   Inicia_Buffer}
end. {Buffer;}
Unit Disk ;

Interface

Uses Global;

Cont Num_Pag_Disco = 256;

Type { estrutura do ponteiro para uma Pagina do disco }
       Pont3  = ^FilaDisco;
       Fila  = record
                   Pag : integer;
                   Prox : pont3
               end;
Var
        { Disco }
       Disco :array [ 0..Nuim_Pag_Disco – 1, 0..7] of Palavra ;

Fila_Disco_Livre  :Pont3; {Fila das paginas livres do Disco}
```

```
{Numero de paginas de discodisponiveis}
{Num_Disco_Livre : integer;

{ aloca uma pagina do Disco (NumPag : integer );
function Aloca_Pag_Disco : Pont3;

{apaga as informacoes da pagina do Disco}
procedure Limpa_Pag_Disco(NumPag : integer);

{ libera uma pagina do disco , colocando-a na fila de pagina livres}
procedure insere_Pag_Disco (var P3, Pont3);]

{ verifica se a fila de paginas do disco corresponde esta vazia }
function Fila_Vazia_Disco (Fila : Pont3):boolean;)

impremention

function Aloca_Pag_Disco: Pont3;
  begin
       Aloca_Pag_Disco:=Fila_Disco_Livre;
        if Fila_Disco_Livre <>nil
          then begin

                    Fila_Disco_Livre : =Fila_Disco_Livre^. Prox;
                    Num_Disco_Livre : = Num_Disco_Livre –1:
              end;End; {aloca_Pag_Disco}
procedure Limpa_pag_Disco( NumPag : integer );
    var I : integer;
     begin

        for I : = 0 to 7
          do wit Disco [ NumPag , []do
                begin
                   Tipo : = ‘    ‘ ;
                   Operacao : = ‘        ‘ ;
                   Campo1 :  = 0 ;
                   Pag  := 0 ;
                Desl ; : =0 ;
           end ;

end;{Limpa_Pag_Disco}

procedure insere_pag_Disco(var P3, Fila : Pot3);
     Var Pont: pont3;
      Begin

P3^. Prox : nil ;
```

```
if Fila = nil
     then Fila : = P3
     else begin
                Pont : = Fila ;
               while Pont^. Prox <> nil
                     do Pont : = Poont^. Prox;
               Pont^. Prox : = P3;
           end ;

    P3 : =nil;

end ; {insere_Pag_Disco}

Procedure Lbera_Pag_Disco(var P3 : Pont3);
Begin

                Limpa_Pag_Disco(p3^. Pag);

                Insere_Pag_Disco (P´3, Fila_Disco_Livre + 1;
                P3: = nil

end; { Libera_Pag_Disco}


function Fila_Vazia_Disco (Fila : Pont3 ) : boolean;
     begin

          Fila_Vazia_Disco : = nil;
end ; {Fila_Vazia_Disco }
Procedure inicia_Disco_Livre;

   Var I : integer ;
        Pont  : pont3;
Begin

    Fila_Disco_Livre  : = nil;

    For I : = 0 to (num_Pag_Disco -1 ) do
Begin
     New (Pont ) ;
      Limpa_Pag_Disco (I)
      Pont ^. Pag : =I;
      Insere_Pag_Disco (Pont, Fila_Disco_livre );
   end;
Num_Disco_livre: Num_Pag_Disco;
```

```
end ;{Inic ia_Disco_Livre}

Begin

Inicia_Disco_Livre;

end .
```