Capítulo

7

DDE – Dynamic Data Exchange

[Resumo e adaptação do texto da referência 1]

Autor: Constantino Seixas Filho
UFMG – Departamento de Engenharia Eletrônica

DDE – Dynamic Data Exchange

Este tipo de comunicação é muito antigo, remanescente da versão 3 do Windows, e apresenta muitas desvantagens. Entretanto como muito sistemas de automação ainda utilizam este mecanismo ele será estudado superficialmente, antes do estudo dos métodos mais modernos baseado em COM/DCOM.

No tempo do Windows 3 havia somente três maneiras de se estabelecer a comunicação entre aplicações: o *clipboard* ou área de transferência, a memória compartilhada e o DDE.

DDE usa o mecanismo padrão de mensagens do Windows (PostMessage) combinado com o uso de memória compartilhada.

Neste mecanismo uma aplicação denominada servidor se comunica com aplicações clientes através de mensagens.

O diálogo é iniciado pelo cliente que envia a mensagem WM_DDE_INITIATE a todos os programas que estiverem em execução no Windows. Um parâmetro da mensagem indica a categoria geral dos dados desejados pelo cliente. O servidor que se identificar com a categoria de dados responde a esta mensagem e o diálogo se inicia.

Um servidor pode se comunicar com diversos cliente, mas para cada um deles um diálogo deve ser estabelecido. Cada cliente também pode requisitar dados de diversos servidores, mas mais uma vez um diálogo para cada comunicação deverá ser estabelecido. Para cada diálogo será criada uma janela oculta que será usada no mecanismo de troca de mensagem.

A identificação dos dados desejados por um cliente se dá através de uma tripla de três strings denominados: aplicativo, tópico de dados e item.

Exemplo:

O exemplo extraído do livro texto é o seguinte

Um aplicativo denominado DDEPOP1 possui uma base de dados com as populações de cada estado americano. O nome do tópico é portanto: “US_POPULATION”. Esta aplicação poderia suportar outros tópicos tais como “Area_EUA”, etc.

Dentro de cada tópico o servidor DDE suporta um ou mais itens de dados. A identificação do item usará a sigla do nome do estado. Por exemplo, “NY” para New York., “CA” para Califórnia e “US” para o total.

```
= DDEPOP1|US_POPULATION!US
```

A identificação acima pode ser utilizada para um cliente acessar um dado qualquer. Por exemplo, use esta identificação de dentro de uma planilha Excel e o Excel executará o programa DDEPOP1, se este já não estiver em execução e iniciará um diálogo com o programa. obterá os dados e os exibirá na tela.
Estas informações não são estáticas, mas serão atualizadas na planilha sempre que os dados mudarem no servidor. A cada 5 segundos o servidor irá recalcular os dados da população e notificar o cliente que um item foi alterado.

Tipos de diálogos

Os três tipos de diálogo existentes são:

Elo frio *(cold link*), elo morno (*warm link*) e elo quente (*hot link*).

Elo Frio

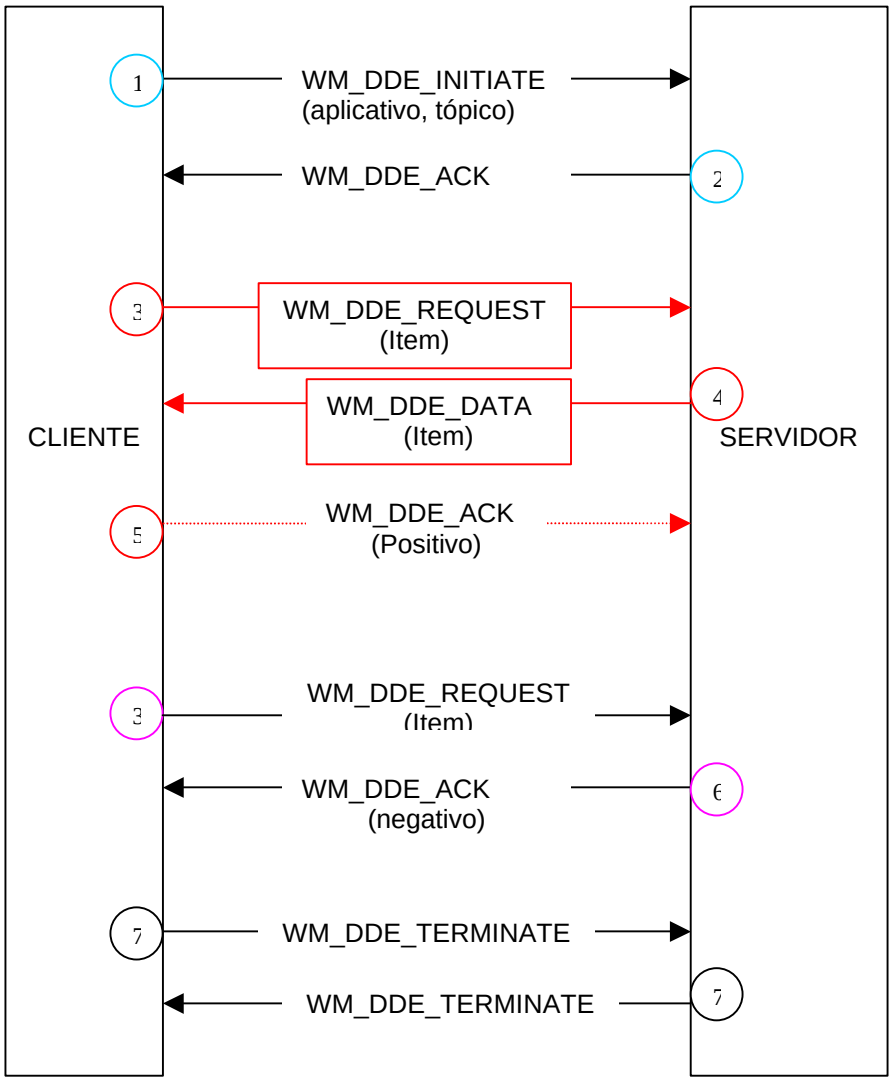
1. O cliente emite a mensagem WM_DDE_INITIATE identificando o aplicativo e o tópico desejado.
2. O servidor responde com WM_DDE_ACK.

3. O cliente requisita um item de dados específico através da mensagem WM_DDE_REQUEST.
4. Se o servidor dispuser dos dados ele responde com WM_DDE_DATA.

5. O cliente pode opcionalmente confirmar o recebimento da mensagem através de WM_DDE_ACK(positivo). O servidor indica se deseja a confirmação através de um sinalizador passado com a mensagem WM_DDE_DATA.

6. Se ao receber uma mensagem WM_DDE_ACK o servidor não dispuser dos dados, irá responder com WM_DDE_ACK (negativo)

7. O diálogo termina quando cliente e servidor trocam as mensagens de encerramento: WM_DDE_TERMINATE:



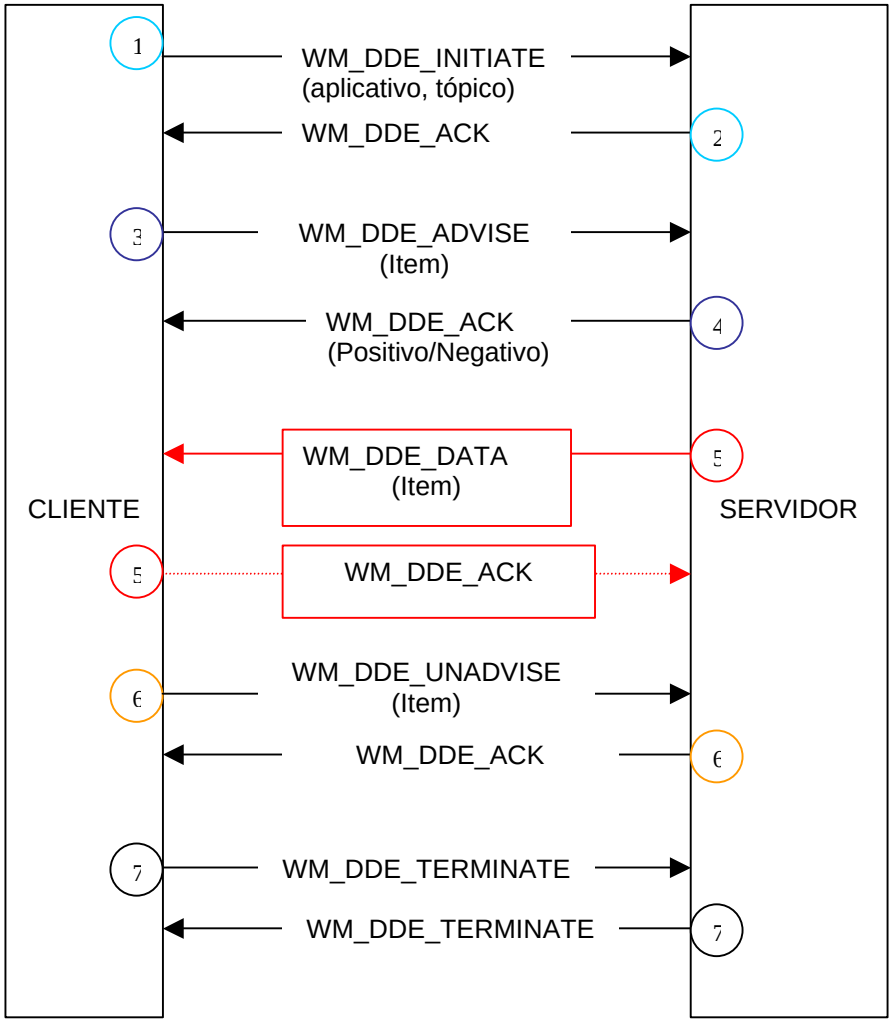
1. O cliente emite a mensagem WM_DDE_INITIATE identificando o aplicativo e o tópico desejado.
2. O servidor responde com WM_DDE_ACK.

3. O cliente requisita um item de dados específico através da mensagem WM_DDE_ADVISE.
4. O servidor irá responder com WM_DDE_ACK positivo ou negativo conforme o caso .

5. O servidor a partir de agora irá notificar o cliente sempre que o valor do item de dados se alterar. Esta notificação usa a mensagem WM_DDE_DATA que pode ou não conter um campo pedindo confirmação de recebimento. Se este campo estiver presente o cliente responderá através da mensagem WM_DDE_ACK.

6. Quando o cliente não desejar mais ser informado das atualizações no item de dados, ele envia uma mensagem WM_DDE_UNADVISE ao servidor e este confirma com WM_DDE_ACK.

7. O diálogo termina quando cliente e servidor trocam as mensagens de encerramento: WM_DDE_TERMINATE:



1. O cliente emite a mensagem WM_DDE_INITIATE identificando o aplicativo e o tópico desejado.
2. O servidor responde com WM_DDE_ACK.

3. O cliente requisita um item de dados específico através da mensagem WM_DDE_ADVISE.
4. O servidor irá responder com WM_DDE_ACK positivo ou negativo conforme o caso .
5. Um sinalizador enviado com a mensagem WM_DDE_ADVISE indica que o cliente quer ser notificado quando um item de dados mudar, mas que não quer receber este item de dados imediatamente. O servidor responde com WM_DDE_DATA com dados NULL.
6. O cliente responde com WM_DDE_ACK.

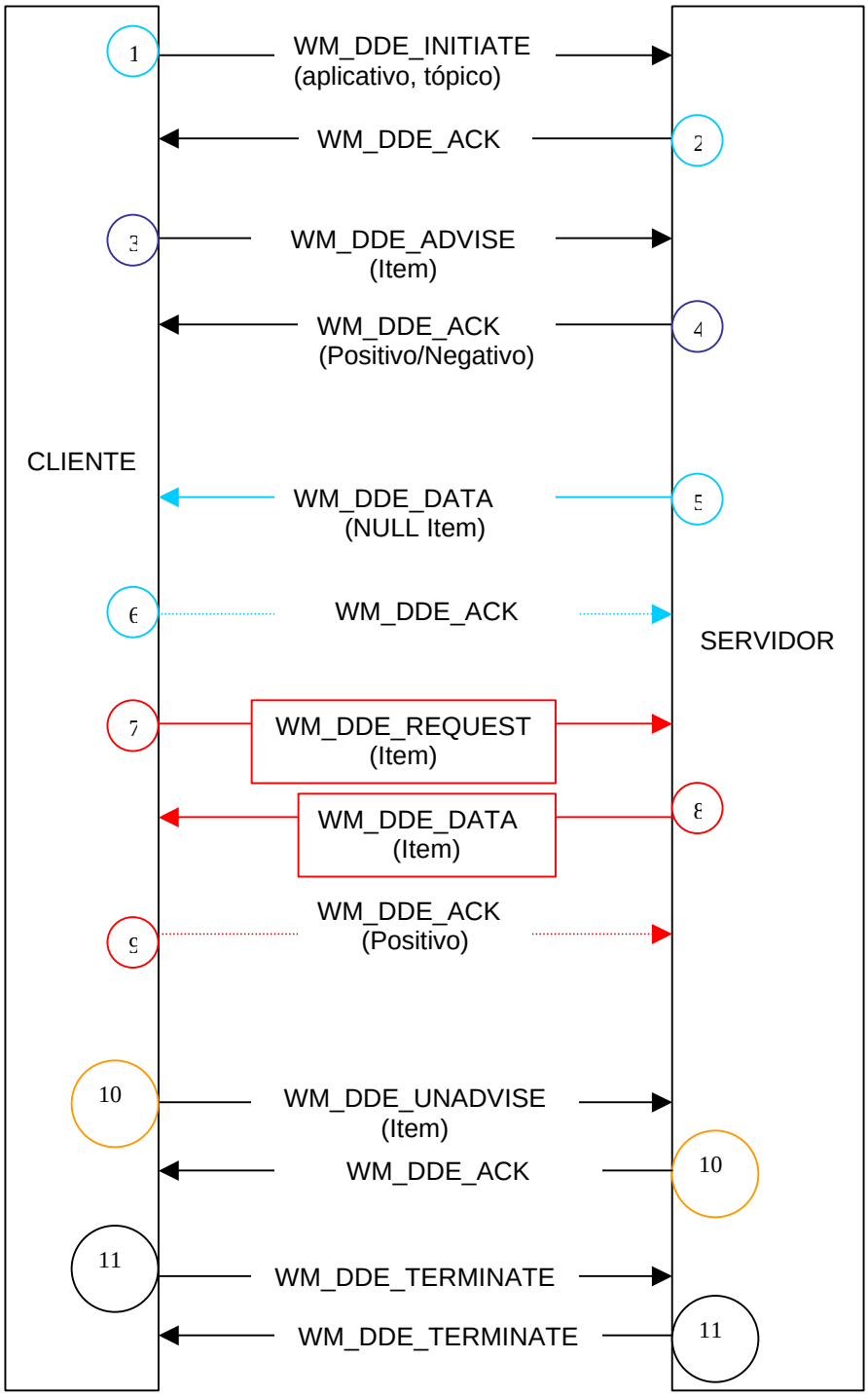
7. O cliente requisita um item de dados específico através da mensagem WM_DDE_REQUEST.
8. Se o servidor dispuser dos dados ele responde com WM_DDE_DATA.
9. O cliente pode opcionalmente confirmar o recebimento da mensagem através de WM_DDE_ACK(positivo). O servidor indica se deseja a confirmação através de um sinalizador passado com a mensagem WM_DDE_DATA.

10. Quando o cliente não desejar mais ser informado das atualizações no item de dados, ele envia uma mensagem WM_DDE_UNADVISE ao servidor e este confirma com WM_DDE_ACK.

11. O diálogo termina quando cliente e servidor trocam as mensagens de encerramento: WM_DDE_TERMINATE:

Apenas dois tipos de mensagens não apareceram nestes diálogos:

WM_DDE_POKE: o cliente passa ao servidor dados não solicitados
WM_DDE_EXECUTE: o cliente envia um string de comandos ao servidor


1
WM_DDE_INITIATE
(aplicativo, tópico)
WM_DDE_ACK
2
3
WM_DDE_ADVISE
(Item)
WM_DDE_ACK
(Positivo/Negativo)
4
CLIENTE
WM_DDE_DATA
(NULL Item)
5
6
WM_DDE_ACK
SERVIDOR
7
WM_DDE_REQUEST
(Item)
WM_DDE_DATA
(Item)
8
WM_DDE_ACK
(Positivo)
9
10
WM_DDE_UNADVISE
(Item)
WM_DDE_ACK
10
11
WM_DDE_TERMINATE
WM_DDE_TERMINATE
11

ÁTOMOS e strings de caracteres

Ao invés de manipular strings diretamente, DDE utiliza o conceito de átomos. Átomos são semelhantes a handles: um valor WORD que referencia strings de caixa livre em uma tabela local ou global.

Instrução	Função
ATOM Atomo;	Declaração
Atomo = AddAtom(lpString);	Adiciona o string à tabela e incrementa o contador de referência do átomo. Retorna o índice do átomo na tabela.
DeleteAtom(atomo);	Decrementa o contador de referências do átomo. Se contador = 0, o átomo e o string são removidos da tabela de átomos.
Atomo = FindAtom(lpString);	Retorna o átomo associado à string ou 0 se string inexistente.
Nbytes = GetAtomName(Atomo, lpBuffer, nTamBuffer);	Devolve o string associado ao átomo.

Funções Globais:

As funções anteriores alocam memória dentro do próprio programa. Para usar átomos DDE necessita-se utilizar um outro conjunto de funções que alocam a memória globalmente, isto é num segmento de dados compartilhado com todas aplicações Windows.

Instrução	Função
ATOM Atomo;	Declaração
Atomo = GlobalAddAtom(lpString);	Adiciona o string à tabela e incrementa o contador de referência do átomo.
GlobalDeleteAtom(atomo);	Decrementa o contador de referências do átomo. Se contador = 0, o átomo e o string são removidos da tabela de átomos.
Atomo =GlobalFindAtom(lpString);	Retorna o átomo associado à string ou 0 se string inexistente.
Nbytes = GlobalGetAtomName(Atomo, lpBuffer, nTamBuffer);	Devolve o string associado ao átomo.

Os átomos serão usados para as strings do aplicativo, tópico e item do DDE. As estruturas de dados transferidas entre programas Windows devem ser alocadas no heap utilizando as instruções Global Alloc ou HeapAlloc. A memória ficará compartilhada entre vários processos, mas o servidor assegurará que o acesso será do tipo *read-only*. Arquivos mapeados em memória também poderia ser utilizados, mas possuem o inconveniente de permitir que a memória comum seja corrompida por uma das aplicações. Apenas o servidor pode alterar o conteúdo do

bloco de memória comum. Quando o servidor envia uma mensagem do tipo WM_DDE_DATA par um cliente, o Windows faz uma cópia da estrutura DDEDATA que é passada à aplicação.

Formato das mensagens

DDEACK

fAck	fBusy	reserved:6	bAppReturnCode:8

DDEADVISE

fAckReq	fDeferUpd	Reserved:14
cfFormat		

DDEDATA

fAckReq	--	fRelease	fResponse	Unused:12
cfFormat				
Value				

Programas

DDE_SERVER

```
// DDEPOP1.C -- DDE Server for Population Data
//  (c) Charles Petzold, 1996

#include <windows.h>
#include <dde.h>
#include <string.h>
#include "ddepop.h"         // Contém mensagens DDE e estruturas de dados

// Serve para marcar que cliente espera uma notificação de mudança de dado
typedef struct
{
    unsigned int fAdvise:1;          // Bit field: 1 bit
    unsigned int fDeferUpd:1;
    unsigned int fAckReq:1;
    unsigned int dummy:13;
    long   lPopPrev ;
 }
```

```
 POPADVISE;

#define ID_TIMER  1
#define DDE_TIMEOUT 3000

LRESULT CALLBACK WndProc  (HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK ServerProc (HWND, UINT, WPARAM, LPARAM);
BOOL  CALLBACK TimerEnumProc (HWND, LONG);
BOOL  CALLBACK CloseEnumProc (HWND, LONG);
BOOL   PostDataMessage (HWND, HWND, int, BOOL, BOOL, BOOL);

char  szAppName[] = "DdePop1" ;
char  szServerClass[] = "DdePop1.Server";

HINSTANCE hInst ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
PSTR szCmdLine, int iCmdShow)
{
 HWND  hwnd ;
 MSG   msg ;
 WNDCLASSEX  wndclass ;

 hInst = hInstance ;

 // Register window class: WndProc: DdePop1
 // Janela principal
 wndclass.cbSize = sizeof (wndclass) ;
 wndclass.style = 0 ;
 wndclass.lpfnWndProc = WndProc ;
 wndclass.cbClsExtra  = 0 ;
 wndclass.cbWndExtra  = 0 ;
 wndclass.hInstance = hInstance ;
 wndclass.hIcon   = LoadIcon (hInstance, szAppName) ;
 wndclass.hCursor   = LoadCursor (NULL, IDC_ARROW) ;
 wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
 wndclass.lpszMenuName  = NULL ;
 wndclass.lpszClassName = szAppName;
 wndclass.hIconSm   = LoadIcon (hInstance, szAppName) ;

 RegisterClassEx (&wndclass) ;

 // Register window class for DDE Server: ServerProc
 // Usada para janelas filhas criadas para manter as conversações
 wndclass.cbSize  = sizeof (wndclass);
 wndclass.style   = 0 ;
 wndclass.lpfnWndProc = ServerProc;
 wndclass.cbClsExtra  = 0;
 // Reserva duas palavras por janela:
 // Windows handle do cliente com a que estamos comunicando
 // handle para memória contendo NUM_STATE estruturas POPADVISE
 wndclass.cbWndExtra  = 2 * sizeof (DWORD);
 wndclass.hInstance = hInstance ;
 wndclass.hIcon   = NULL ;
 wndclass.hCursor   = NULL ;
 wndclass.hbrBackground = NULL ;
 wndclass.lpszMenuName  = NULL ;
 wndclass.lpszClassName = szServerClass ;
 wndclass.hIconSm   = NULL ;
```

```
 RegisterClassEx (&wndclass) ;
 // Cria janela principal
 hwnd = CreateWindow (szAppName, "DDE Population Server",
                 WS_OVERLAPPEDWINDOW,
                 CW_USEDEFAULT, CW_USEDEFAULT,
                 CW_USEDEFAULT, CW_USEDEFAULT,
                 NULL, NULL, hInstance, NULL) ;

 InitPops(); // inicializa estrutura contendo populações dos EUA

 SetTimer (hwnd, ID_TIMER, 5000, NULL);
 // Programa timer de 5s: gera WM_TIMER a cada T

 ShowWindow (hwnd, SW_SHOWMINNOACTIVE); // Exibe janela minimizada
 UpdateWindow (hwnd) ;

 while (GetMessage (&msg, NULL, 0, 0))
 {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
 }

 KillTimer (hwnd, ID_TIMER) ;

 return msg.wParam ;
}  // WinMain



LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam,
LPARAM lParam)
{
 static char szTopic[] = "US_Population" ;
 ATOM  aApp, aTop ;
 HWND  hwndClient, hwndServer ;

 switch (iMsg)
 {
    case WM_DDE_INITIATE :
        // wParam  -- handle da janela enviando a mensagem
        // LOWORD (lParam) -- átomo identificando a app (NULL = qualquer servidor)
        // HIWORD (lParam) -- átomo identificando um tópico (NULL = todos tópicos)
        hwndClient = (HWND) wParam ;

        aApp = GlobalAddAtom (szAppName); // DdePop1
        aTop = GlobalAddAtom (szTopic); // US_Population

        // átomos coincidem ?
        if ((LOWORD (lParam) == NULL || LOWORD (lParam) == aApp) &&
        (HIWORD (lParam) == NULL || HIWORD (lParam) == aTop))
        {       // cria janela para suportar a transação
        hwndServer = CreateWindow (szServerClass, NULL,
                        WS_CHILD, 0, 0, 0, 0,
                        hwnd, NULL, hInst, NULL) ;
        // define primeiro parâmetro da janela como sendo o handle do servidor
        SetWindowLong (hwndServer, 0, (LONG) hwndClient);
        // Envia uma mensagem de ACK para cada tópico suportado
        SendMessage ((HWND) wParam, WM_DDE_ACK,
                        (WPARAM) hwndServer,
                        MAKELPARAM (aApp, aTop)) ;
```

```
        }
        // Otherwise, delete the atoms just created
         else {
                GlobalDeleteAtom (aApp);
                GlobalDeleteAtom (aTop);
        }
        return 0 ;

  case WM_TIMER :
  case WM_TIMECHANGE :
        // Calcula novas populações
        CalcPops () ;
        // Notifica todas as janelas filhas a cada 5s
        // Enumera janelas filhas e passa como parâmetro para função callback
        EnumChildWindows(hwnd, &TimerEnumProc, 0L) ;
        return 0 ;

  case WM_QUERYOPEN :
        return 0 ;

  case WM_CLOSE :
        // Notify all child windows
        EnumChildWindows (hwnd, &CloseEnumProc, 0L);
        break;  // for default processing

  case WM_DESTROY :
        PostQuitMessage (0);
        return 0 ;
  }
 return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}


LRESULT CALLBACK ServerProc (HWND hwnd, UINT iMsg, WPARAM wParam,
LPARAM lParam)
 {
        ATOM  aItem ;
        char  szItem[10] ;
        DDEACK  DdeAck ;
        DDEADVISE  *pDdeAdvise ;
        DWORD   dwTime ;
        GLOBALHANDLE  hPopAdvise, hDdeAdvise, hCommands, hDdePoke ;
        int   i ;
        UINT  uiLow, uiHi ;
        HWND  hwndClient ;
        MSG   msg ;
        POPADVISE  *pPopAdvise ;
        WORD  cfFormat, wStatus ;

switch (iMsg)
{
  case WM_CREATE :
        // Allocate memory for POPADVISE structures

        hPopAdvise = GlobalAlloc (GHND, NUM_STATES * sizeof (POPADVISE));

        if (hPopAdvise == NULL)
                DestroyWindow (hwnd) ;
        else // Salva handle para a memória alocada na segunda pal. reservada da janela
        SetWindowLong (hwnd, 4, (LONG) hPopAdvise) ;
```

```
        return 0 ;

case WM_DDE_REQUEST :

        // wParam  -- sending window handle
        // LOWORD (lParam) -- data format
        // HIWORD (lParam) -- item atom
        hwndClient = (HWND) wParam ;
        cfFormat = LOWORD (lParam) ;
        aItem  = HIWORD (lParam) ;

        // POPDDE1 só suporta formato CF_TEXT
        if (cfFormat == CF_TEXT)
        {  // Busca string associado ao átomo item
            GlobalGetAtomName (aItem, szItem, sizeof (szItem)) ;
            // Compara string com todo os nomes de estados suportados
            for (i = 0 ; i < NUM_STATES ; i++)
                if (strcmp (szItem, pop[i].szState) == 0) break ;
                if (i < NUM_STATES)  // Estado está na lista
                {
                  GlobalDeleteAtom (aItem);  // Deleta átomo e envia dado
                  PostDataMessage (hwnd, hwndClient, i,FALSE, FALSE, TRUE);
                  // Post Data Message irá recriar o átomo para enviar resposta
                  return 0 ;
                }
        }

        // Não achou item
        DdeAck.bAppReturnCode = 0 ;
        DdeAck.reserved   = 0 ;
        DdeAck.fBusy  = FALSE ;
        DdeAck.fAck   = FALSE ;

        wStatus = *((WORD *) &DdeAck) ;

        if (!PostMessage (hwndClient, WM_DDE_ACK, (WPARAM) hwnd,
              PackDDElParam (WM_DDE_ACK, wStatus, aItem)))
 { // Usa átomo e o apaga se o cliente não recebeu (morreu)
        GlobalDeleteAtom (aItem) ;
 }

 return 0 ;

case WM_DDE_ADVISE :
        // wParam -- sending window handle
        // lParam -- handle para DDEADVISE (global) e átomo identificando item
        // Extrai handle e item

        UnpackDDElParam (WM_DDE_ADVISE, lParam, &uiLow, &uiHi) ;
        FreeDDElParam (WM_DDE_ADVISE, lParam); // libera lParam

        hwndClient = (HWND) wParam ;
        hDdeAdvise = (GLOBALHANDLE) uiLow ;
        aItem  = (ATOM) uiHi ;

        pDdeAdvise = (DDEADVISE *) GlobalLock (hDdeAdvise) ;

        // Check for matching format and data item
        if (pDdeAdvise->cfFormat == CF_TEXT)
        {
```

```
        GlobalGetAtomName (aItem, szItem, sizeof (szItem)) ;
        // verifica se item está na lista dos estados
        for (i = 0 ; i < NUM_STATES ; i++)
        if (strcmp (szItem, pop[i].szState) == 0)
        break ;

      // Fill in the POPADVISE structure and acknowledge
      if (i < NUM_STATES)
      {
          hPopAdvise = (GLOBALHANDLE) GetWindowLong (hwnd, 4) ;
          pPopAdvise = (POPADVISE *) GlobalLock (hPopAdvise) ;

          pPopAdvise[i].fAdvise = TRUE;
          // Cliente quer ser notificado de mudanças no item
          pPopAdvise[i].fDeferUpd = pDdeAdvise->fDeferUpd; // TRUE: warm link
          pPopAdvise[i].fAckReq = pDdeAdvise->fAckReq;
          // TRUE; server irá pedir Ack da msg para cliente
          pPopAdvise[i].lPopPrev  = pop[i].lPop;
          // Dado = população corrente do estado

          GlobalUnlock (hDdeAdvise);
          GlobalFree (hDdeAdvise);
          // Server agora irá emitir ACK da mensagem WM_DDE_ADVISE
          DdeAck.bAppReturnCode = 0 ;
          DdeAck.reserved   = 0 ;
          DdeAck.fBusy  = FALSE ;
          DdeAck.fAck   = TRUE ;

          wStatus = *((WORD *) &DdeAck) ;

          if (!PostMessage (hwndClient, WM_DDE_ACK, (WPARAM) hwnd,
                  PackDDElParam (WM_DDE_ACK, wStatus, aItem)))
          {
              GlobalDeleteAtom (aItem) ;
          }
          else
          {
              PostDataMessage (hwnd, hwndClient, i,
                  pPopAdvise[i].fDeferUpd,
                  pPopAdvise[i].fAckReq,
                  FALSE) ;
          }

          GlobalUnlock(hPopAdvise) ;
        return 0 ;
    }
    }

    // Otherwise, post a negative WM_DDE_ACK
    GlobalUnlock (hDdeAdvise) ;

    DdeAck.bAppReturnCode = 0 ;
    DdeAck.reserved   = 0 ;
    DdeAck.fBusy  = FALSE ;
    DdeAck.fAck   = FALSE ;

    wStatus = *((WORD *) &DdeAck) ;

    if (!PostMessage (hwndClient, WM_DDE_ACK, (WPARAM) hwnd,
      PackDDElParam (WM_DDE_ACK, wStatus, aItem)))
```

```
        {
            GlobalFree (hDdeAdvise) ;
            GlobalDeleteAtom (aItem) ;
        }

return 0 ;

  case WM_DDE_UNADVISE :
        // Solicita o servidor a parar de enviar mensagens
        // WM_DDE_DATA quando um item de dado mudar

        // wParam  -- sending window handle
        // LOWORD (lParam) -- data format
        // HIWORD (lParam) -- item atom
        hwndClient = (HWND) wParam ;
        cfFormat = LOWORD (lParam) ;
        aItem  = HIWORD (lParam) ;

        DdeAck.bAppReturnCode = 0 ;
        DdeAck.reserved   = 0 ;
        DdeAck.fBusy  = FALSE ;
        DdeAck.fAck   = TRUE ;

        hPopAdvise  = (GLOBALHANDLE) GetWindowLong (hwnd, 4) ;
        pPopAdvise = (POPADVISE *) GlobalLock (hPopAdvise) ;

        // Check for matching format and data item
        if (cfFormat == CF_TEXT || cfFormat == NULL)
        {
            if (aItem == (ATOM) NULL)
                  for (i = 0 ; i < NUM_STATES ; i++)
                        pPopAdvise[i].fAdvise = FALSE ;
            else
            {
                  GlobalGetAtomName (aItem, szItem, sizeof (szItem)) ;
                  for (i = 0 ; i < NUM_STATES ; i++)
                        if (strcmp (szItem, pop[i].szState) == 0) break ;

                  if (i < NUM_STATES)
                      pPopAdvise[i].fAdvise = FALSE;
                      // Desmarca campo que pede para ser notificado
                  else
                      DdeAck.fAck = FALSE ;
            }
          } // if
          else
                  DdeAck.fAck = FALSE ;

          // Acknowledge either positively or negatively
          wStatus = *((WORD *) &DdeAck) ;

          if (!PostMessage (hwndClient, WM_DDE_ACK, (WPARAM) hwnd,
          PackDDElParam (WM_DDE_ACK, wStatus, aItem)))
          {
              if (aItem != (ATOM) NULL)
              GlobalDeleteAtom (aItem) ;
          }

          GlobalUnlock (hPopAdvise) ;
          return 0 ;
```

```
case WM_DDE_EXECUTE :
        // Responde com ACK negativo
        hwndClient = (HWND) wParam ;
        hCommands  = (GLOBALHANDLE) lParam ;

        DdeAck.bAppReturnCode = 0 ;
        DdeAck.reserved   = 0 ;
        DdeAck.fBusy  = FALSE ;
        DdeAck.fAck   = FALSE ;

        wStatus = *((WORD *) &DdeAck) ;

        if (!PostMessage (hwndClient, WM_DDE_ACK, (WPARAM) hwnd,
                  PackDDElParam (WM_DDE_ACK,
                       wStatus, (UINT) hCommands)))
        {
                  GlobalFree (hCommands) ;
        }
        return 0 ;

case WM_DDE_POKE :
        // Responde com ACK negativo
        UnpackDDElParam (WM_DDE_POKE, lParam, &uiLow, &uiHi) ;
        FreeDDElParam (WM_DDE_POKE, lParam) ;
        hwndClient = (HWND) wParam ;
        hDdePoke = (GLOBALHANDLE) uiLow ;
        aItem  = (ATOM) uiHi ;

        DdeAck.bAppReturnCode = 0 ;
        DdeAck.reserved   = 0 ;
        DdeAck.fBusy  = FALSE ;
        DdeAck.fAck   = FALSE ;

        wStatus = *((WORD *) &DdeAck) ;

        if (!PostMessage (hwndClient, WM_DDE_ACK, (WPARAM) hwnd,
              PackDDElParam (WM_DDE_ACK, wStatus, aItem)))
        {
            GlobalFree (hDdePoke) ;
            GlobalDeleteAtom (aItem) ;
        }

        return 0 ;

case WM_DDE_TERMINATE :
        // Responde com outra mensagem WM_DDE_TERMINATE iMsg
        hwndClient = (HWND) wParam ;
        PostMessage (hwndClient, WM_DDE_TERMINATE, (WPARAM) hwnd, 0L) ;
        DestroyWindow (hwnd) ;
        return 0 ;

case WM_TIMER :
        // Verifica se cliente está cadastrado para ser notificado
        // e se a população mudou. Se mudou:
        // Post WM_DDE_DATA iMsgs for changed populations
        hwndClient = (HWND) GetWindowLong (hwnd, 0) ;
        hPopAdvise = (GLOBALHANDLE) GetWindowLong (hwnd, 4) ;
        pPopAdvise = (POPADVISE *) GlobalLock (hPopAdvise) ;
```

```
        for (i = 0 ; i < NUM_STATES ; i++)
            if (pPopAdvise[i].fAdvise)
              if (pPopAdvise[i].lPopPrev != pop[i].lPop)
              {
                  if (!PostDataMessage (hwnd, hwndClient, i,
                  pPopAdvise[i].fDeferUpd,
                  pPopAdvise[i].fAckReq,
                  FALSE))
                  break ;

                  pPopAdvise[i].lPopPrev = pop[i].lPop ;
              }

            GlobalUnlock (hPopAdvise) ;
            return 0 ;

  case WM_CLOSE :
            // Post a WM_DDE_TERMINATE iMsg to the client
            hwndClient = (HWND) GetWindowLong (hwnd, 0) ;
            PostMessage (hwndClient, WM_DDE_TERMINATE, (WPARAM) hwnd, 0L) ;

            dwTime = GetCurrentTime () ;

            while (GetCurrentTime () - dwTime < DDE_TIMEOUT)
                if (PeekMessage (&msg, hwnd, WM_DDE_TERMINATE,
                    WM_DDE_TERMINATE, PM_REMOVE))
            break ;

            DestroyWindow (hwnd) ;
            return 0 ;

  case WM_DESTROY :
            hPopAdvise = (GLOBALHANDLE) GetWindowLong (hwnd, 4) ;
            GlobalFree (hPopAdvise) ;
            return 0 ;
  }
 return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
 }


BOOL CALLBACK TimerEnumProc (HWND hwnd, LPARAM lParam)
{
        SendMessage (hwnd, WM_TIMER, 0, 0L);
        return TRUE ;
}


BOOL CALLBACK CloseEnumProc (HWND hwnd, LPARAM lParam)
{
        SendMessage (hwnd, WM_CLOSE, 0, 0L) ;
        return TRUE ;
}


BOOL PostDataMessage (HWND hwndServer, HWND hwndClient, int iState,
        BOOL fDeferUpd, BOOL fAckReq, BOOL fResponse)
{
    ATOM   aItem ;
    char   szPopulation[16] ;
    DDEACK   DdeAck ;
```

```
    DDEDATA *pDdeData ;
    DWORD  dwTime ;
    GLOBALHANDLE hDdeData ;
    MSG  msg ;
    WORD   wStatus ;

     aItem = GlobalAddAtom (pop[iState].szState) ;

     // Allocate a DDEDATA structure if not deferred update

    if (fDeferUpd)
    {
        hDdeData = NULL ;
    }
    else
    {
        // Forma string com o valor da população terminado em <CR><LF>
        wsprintf (szPopulation, "%ld\r\n", pop[iState].lPop) ;

        hDdeData = GlobalAlloc (GHND | GMEM_DDESHARE,
        sizeof (DDEDATA) + strlen (szPopulation)) ;

        pDdeData = (DDEDATA *) GlobalLock (hDdeData) ;

        pDdeData->fResponse = fResponse;
        // TRUE= dado em resposta a DDE_DATA
        pDdeData->fRelease  = TRUE;   // Cliente deve liberar bloco de memória
        pDdeData->fAckReq = fAckReq;
        // FALSE = WM_DDE_ACK do cliente não é necessária
        pDdeData->cfFormat  = CF_TEXT;      // Formato texto
        // Copia estrutura
        lstrcpy ((PSTR) pDdeData->Value, szPopulation) ;

        GlobalUnlock (hDdeData) ;
  }

  // Post the WM_DDE_DATA iMsg

 if (!PostMessage (hwndClient, WM_DDE_DATA, (WPARAM) hwndServer,
          PackDDElParam (WM_DDE_DATA, (UINT) hDdeData, aItem)))
 {
  if (hDdeData != NULL)
   GlobalFree (hDdeData) ;

  GlobalDeleteAtom (aItem); // Possivelmente o cliente não mais entre nós
  return FALSE ;
 }

// Espera ACK se confirmação foi solicitada
if (fAckReq)
{
    DdeAck.fAck = FALSE ;
    dwTime = GetCurrentTime () ;

    while (GetCurrentTime () - dwTime < DDE_TIMEOUT)
    {
        if (PeekMessage (&msg, hwndServer, WM_DDE_ACK, WM_DDE_ACK,
        PM_REMOVE))
        {
            wStatus = LOWORD (msg.lParam) ;
```

```
            DdeAck = *((DDEACK *) &wStatus) ;
            aItem  = HIWORD (msg.lParam) ;
            GlobalDeleteAtom (aItem) ;
            break ;
          }
      }

if (DdeAck.fAck == FALSE)
{
   if (hDdeData != NULL)
         GlobalFree (hDdeData) ;

   return FALSE ;
   }
  }

 return TRUE ;
}  // PostDataMessage
```

T e s t e

Para testar o programa abra a planilha Excel e digite em alguma células strings do tipo:
= DDEPOP1|US_POPULATION!US
= DDEPOP1|US_POPULATION!CA
= DDEPOP1|US_POPULATION!FL

Todas as solicitações da planilha Excel são do tipo **warm link**.

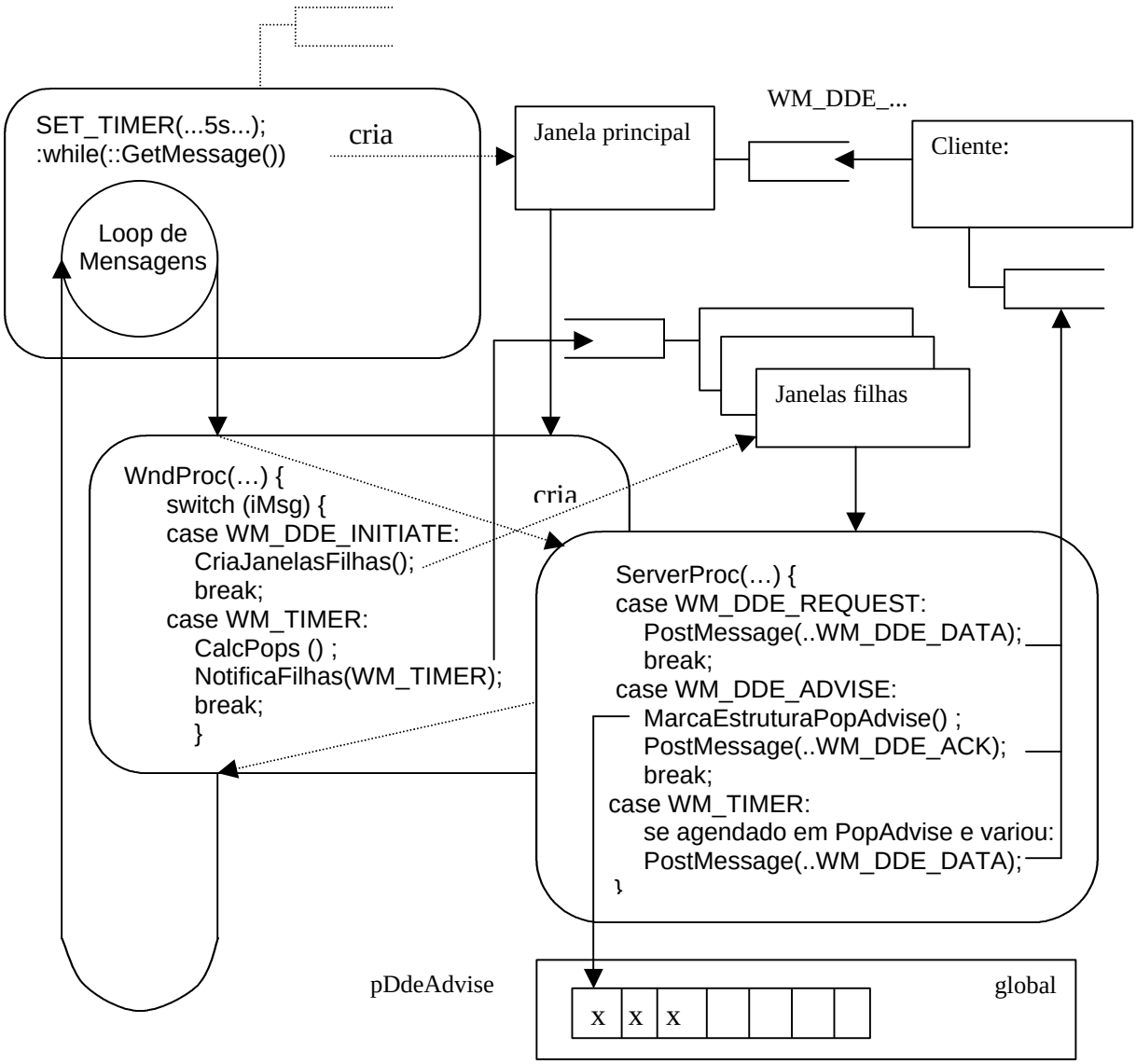


Figura 1: Mensagens trocadas entre as aplicações. Note que a fila de mensagem representada como ligadas às janelas é na verdade uma fila única associada à GUI thread.

DDE_CLIENT

O servidor DDE comunica com o Excel e outros clientes DDE. Vamos construir agora um novo cliente DDE dedicado a esta aplicação.

Compile o programa Showpop1. Agora parta a aplicação DDEPop1 e em seguida a aplicação Showpop1. A listagem da população de todos os estados americanos será mostrada na tela:

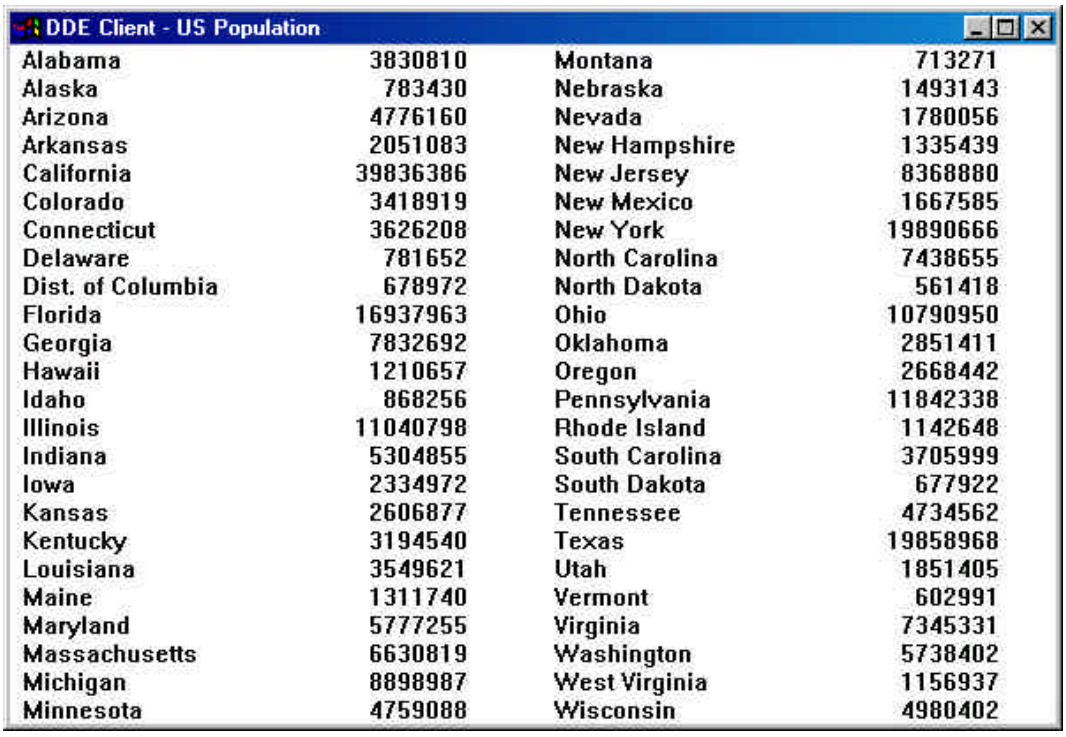


Figura 2: Tela da aplicação DDE Client

DDEPOP1 mantém apenas uma conversação com o servidor, usando a mensagem WM_DDE_ADVISE.

D D E P O P 1

```
/*----------------------------------------
SHOWPOP1.C -- DDE Client using DDEPOP1
(c) Charles Petzold, 1996
  ----------------------------------------*/
#include <windows.h>
#include <dde.h>
#include <stdlib.h>
#include <string.h>
#include "showpop.h"

#define WM_USER_INITIATE (WM_USER + 1)
#define DDE_TIMEOUT   3000

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

char szAppName[] = "ShowPop1" ;


int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
PSTR szCmdLine, int iCmdShow)
{
    HWND    hwnd ;
    MSG             msg ;
    WNDCLASSEX wndclass ;

    wndclass.cbSize = sizeof (wndclass) ;
```

```
    wndclass.style  = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc        = WndProc ;
    wndclass.cbClsExtra  = 0 ;
    wndclass.cbWndExtra  = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = LoadIcon (hInstance, szAppName) ;
    wndclass.hCursor  = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH) ;
    wndclass.lpszMenuName  = NULL ;
    wndclass.lpszClassName = szAppName ;
    wndclass.hIconSm       = LoadIcon (hInstance, szAppName) ;

    RegisterClassEx (&wndclass) ;

    hwnd = CreateWindow (szAppName, "DDE Client - US Population",
                                    WS_OVERLAPPEDWINDOW,
                                    CW_USEDEFAULT, CW_USEDEFAULT,
                                    CW_USEDEFAULT, CW_USEDEFAULT,
                                    NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    SendMessage (hwnd, WM_USER_INITIATE, 0, 0L) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
         TranslateMessage (&msg) ;
         DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}


LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam,
LPARAM lParam)
{
    static BOOL     fDoingInitiate = TRUE ;
    static char     szServerApp[]  = "DdePop1",
                    szTopic[]      = "US_Population" ;
    static HWND hwndServer = NULL ;
    static long     cxChar, cyChar ;
    ATOM            aApp, aTop, aItem ;
    char            szBuffer[24], szPopulation[16], szItem[16] ;
    DDEACK DdeAck ;
    DDEDATA     *pDdeData ;
    DDEADVISE   *pDdeAdvise ;
    DWORD dwTime ;
    GLOBALHANDLE hDdeAdvise, hDdeData ;
    HDC             hdc ;
    MSG             msg ;
    PAINTSTRUCT ps ;
    Short           i ;
    Long            x, y ;
    TEXTMETRIC  tm ;
    WORD            wStatus ;
    UINT            uiLow, uiHi ;

    switch (iMsg)
    {
```

```
case WM_CREATE :
        hdc = GetDC (hwnd) ;
        GetTextMetrics (hdc, &tm) ;
        cxChar = tm.tmAveCharWidth ;
        cyChar = tm.tmHeight + tm.tmExternalLeading ;
        ReleaseDC (hwnd, hdc) ;
        return 0 ;

case WM_USER_INITIATE :
        // Broadcast WM_DDE_INITIATE iMsg
        aApp = GlobalAddAtom (szServerApp); // cria átomo para nome da aplic.
        aTop = GlobalAddAtom (szTopic);  // cria átomo para nome do tópico
        // Envia mensagem para todas as janelas
        SendMessage (     HWND_BROADCAST,
                        WM_DDE_INITIATE, (WPARAM) hwnd,
                        MAKELONG (aApp, aTop)) ;

        // Ninguém respondeu: Parte DDEPop1 primeiro
        // DDEpop1 deve estar no mesmo diretório da aplicação ou no PATH
        if (hwndServer == NULL)
        {
            WinExec(szServerApp, SW_SHOWMINNOACTIVE) ;
            SendMessage (HWND_BROADCAST,
                            WM_DDE_INITIATE, (WPARAM) hwnd,
                            MAKELONG (aApp, aTop)) ;
        }

        // Delete the atoms
        GlobalDeleteAtom (aApp) ;
        GlobalDeleteAtom (aTop) ;
        fDoingInitiate = FALSE ;

        // Ninguém respondeu: exibe mensagem e aborta
        if (hwndServer == NULL)
        {       MessageBox (hwnd, "Cannot connect with DDEPOP1.EXE!",
                        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                return 0 ;
        }

        // Post WM_DDE_ADVISE iMsgs
        for (i = 0 ; i < NUM_STATES ; i++)
        {
            hDdeAdvise = GlobalAlloc (GHND | GMEM_DDESHARE,
                                        sizeof (DDEADVISE)) ;

            pDdeAdvise = (DDEADVISE *) GlobalLock (hDdeAdvise) ;
            pDdeAdvise->fAckReq = TRUE;
            // Server deve enviar WM_DDE_DATA com fAckReq=NULL
            pDdeAdvise->fDeferUpd = FALSE; // Hot link e não warm link
            pDdeAdvise->cfFormat  = CF_TEXT;       // vai receber strings

            GlobalUnlock (hDdeAdvise) ;

            aItem = GlobalAddAtom (pop[i].szAbb) ;
            if (!PostMessage (hwndServer, WM_DDE_ADVISE, (WPARAM) hwnd,
                            PackDDElParam (WM_DDE_ADVISE,
                            (UINT) hDdeAdvise, aItem)))
            {   GlobalFree (hDdeAdvise) ;
                GlobalDeleteAtom (aItem) ;
                break ;
```

```
                }

                DdeAck.fAck = FALSE ;
                dwTime = GetCurrentTime () ;
                while (GetCurrentTime () - dwTime < DDE_TIMEOUT)
                {
                    if (PeekMessage (&msg, hwnd, WM_DDE_ACK, WM_DDE_ACK,
                    PM_REMOVE))
                    {
                        GlobalDeleteAtom (HIWORD (msg.lParam)) ;

                        wStatus = LOWORD (msg.lParam) ;
                        DdeAck = *((DDEACK *) &wStatus) ;

                        if (DdeAck.fAck == FALSE)
                            GlobalFree (hDdeAdvise) ;

                        break ;
                    }
                }  // while

                if (DdeAck.fAck == FALSE) break ;
                // Despacha possíveis mensagens WM_DDE_DATA existentes na // fila
                para WndProc.
                while (PeekMessage(&msg, hwnd, WM_DDE_FIRST, WM_DDE_LAST,
                    PM_REMOVE))
                {
                    DispatchMessage (&msg) ;
                }
            }

            if (i < NUM_STATES)
            {
                MessageBox (hwnd, "Failure on WM_DDE_ADVISE!",
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
            }
            return 0 ;

        case WM_DDE_ACK :
            // In response to WM_DDE_INITIATE, save server window
            if (fDoingInitiate)
            {
                UnpackDDElParam (WM_DDE_ACK, lParam, &uiLow, &uiHi);
                FreeDDElParam (WM_DDE_ACK, lParam);
                hwndServer = (HWND) wParam;
                // recebe hwndServer antes de WndProc retornar do send
                // em WM_USER_INITIATE
                GlobalDeleteAtom ((ATOM) uiLow) ;
                GlobalDeleteAtom ((ATOM) uiHi) ;
            }
            return 0 ;

        case WM_DDE_DATA :
            // wParam -- sending window handle
            // lParam -- DDEDATA memory handle & item atom
            UnpackDDElParam (WM_DDE_DATA, lParam, &uiLow, &uiHi) ;
            FreeDDElParam (WM_DDE_DATA, lParam) ;

            hDdeData = (GLOBALHANDLE) uiLow ;
            pDdeData = (DDEDATA *) GlobalLock (hDdeData) ;
```

```
            aItem  = (ATOM) uiHi ;

            // Initialize DdeAck structure
            DdeAck.bAppReturnCode = 0 ;
            DdeAck.reserved     = 0 ;
            DdeAck.fBusy                = FALSE ;
            DdeAck.fAck                  = FALSE ;

            // Check for matching format and data item
            if (pDdeData->cfFormat == CF_TEXT)
            {
                GlobalGetAtomName (aItem, szItem, sizeof (szItem)) ;

                for (i = 0 ; i < NUM_STATES ; i++)
                    if (strcmp (szItem, pop[i].szAbb) == 0)  break ;

                if (i < NUM_STATES)
                {
                    strcpy (szPopulation, (char *) pDdeData->Value) ;
                    pop[i].lPop = atol (szPopulation) ;
                    InvalidateRect (hwnd, NULL, FALSE) ; // Pede refresh de tela

                    DdeAck.fAck = TRUE ;
                }
            }

            // Acknowledge if necessary
            if (pDdeData->fAckReq == TRUE)
            {
                wStatus = *((WORD *) &DdeAck) ;

                if (!PostMessage ((HWND) wParam, WM_DDE_ACK, (WPARAM) hwnd,
                            PackDDElParam (WM_DDE_ACK,
                            wStatus, aItem)))
                {   GlobalDeleteAtom (aItem) ;
                    GlobalUnlock (hDdeData) ;
                    GlobalFree (hDdeData) ;
                    return 0 ;
                }
            }
            else
            {
                GlobalDeleteAtom (aItem) ;
            }

            // Clean up

            if (pDdeData->fRelease == TRUE || DdeAck.fAck == FALSE)
            {
                  GlobalUnlock (hDdeData) ;
                  GlobalFree (hDdeData) ;
            }
            else
            {
                  GlobalUnlock (hDdeData) ;
            }

            return 0 ;

        case WM_PAINT : // Atualiza população na tela
```

```
            hdc = BeginPaint (hwnd, &ps) ;

            for (i = 0 ; i < NUM_STATES ; i++)
            {
                if (i < (NUM_STATES + 1) / 2)
                {
                    x = cxChar ;
                    y = i * cyChar ;
                }
                else
                {
                    x = 44 * cxChar ;
                    y = (i - (NUM_STATES + 1) / 2) * cyChar ;
                }

                TextOut (hdc, x, y, szBuffer, wsprintf (szBuffer, "%-20s",
                (PSTR) pop[i].szState)) ;

                x += 36 * cxChar ;

                SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;
                TextOut (hdc, x, y, szBuffer,
                wsprintf (szBuffer, "%10ld", pop[i].lPop)) ;

                SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
            }

            EndPaint (hwnd, &ps) ;
            return 0 ;

        case WM_DDE_TERMINATE :

            // Respond with another WM_DDE_TERMINATE iMsg
            PostMessage (hwndServer, WM_DDE_TERMINATE, (WPARAM) hwnd, 0L) ;
            hwndServer = NULL ;  // conversação encerrada
            return 0 ;

        case WM_CLOSE :
            if (hwndServer == NULL)
            break ;

            // Post WM_DDE_UNADVISE iMsg
            PostMessage (hwndServer, WM_DDE_UNADVISE, (WPARAM) hwnd,
            MAKELONG (CF_TEXT, NULL)) ;

            dwTime = GetCurrentTime () ;
            while (GetCurrentTime () - dwTime < DDE_TIMEOUT)
            {
                if (PeekMessage (&msg, hwnd, WM_DDE_ACK, WM_DDE_ACK,
                PM_REMOVE))
                break ;
            }

            // Post WM_DDE_TERMINATE iMsg
            PostMessage (hwndServer, WM_DDE_TERMINATE, (WPARAM) hwnd, 0L) ;

            dwTime = GetCurrentTime () ;
            while (GetCurrentTime () - dwTime < DDE_TIMEOUT)
            {
                if (PeekMessage (&msg, hwnd, WM_DDE_TERMINATE,
```

```
                    WM_DDE_TERMINATE, PM_REMOVE))
                    break ;
                }
                break ;          // for default processing

        case WM_DESTROY :
                PostQuitMessage (0) ;
                return 0 ;
                }
    return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}
```

A figura seguir mostra um diagrama de processamento do lado cliente.

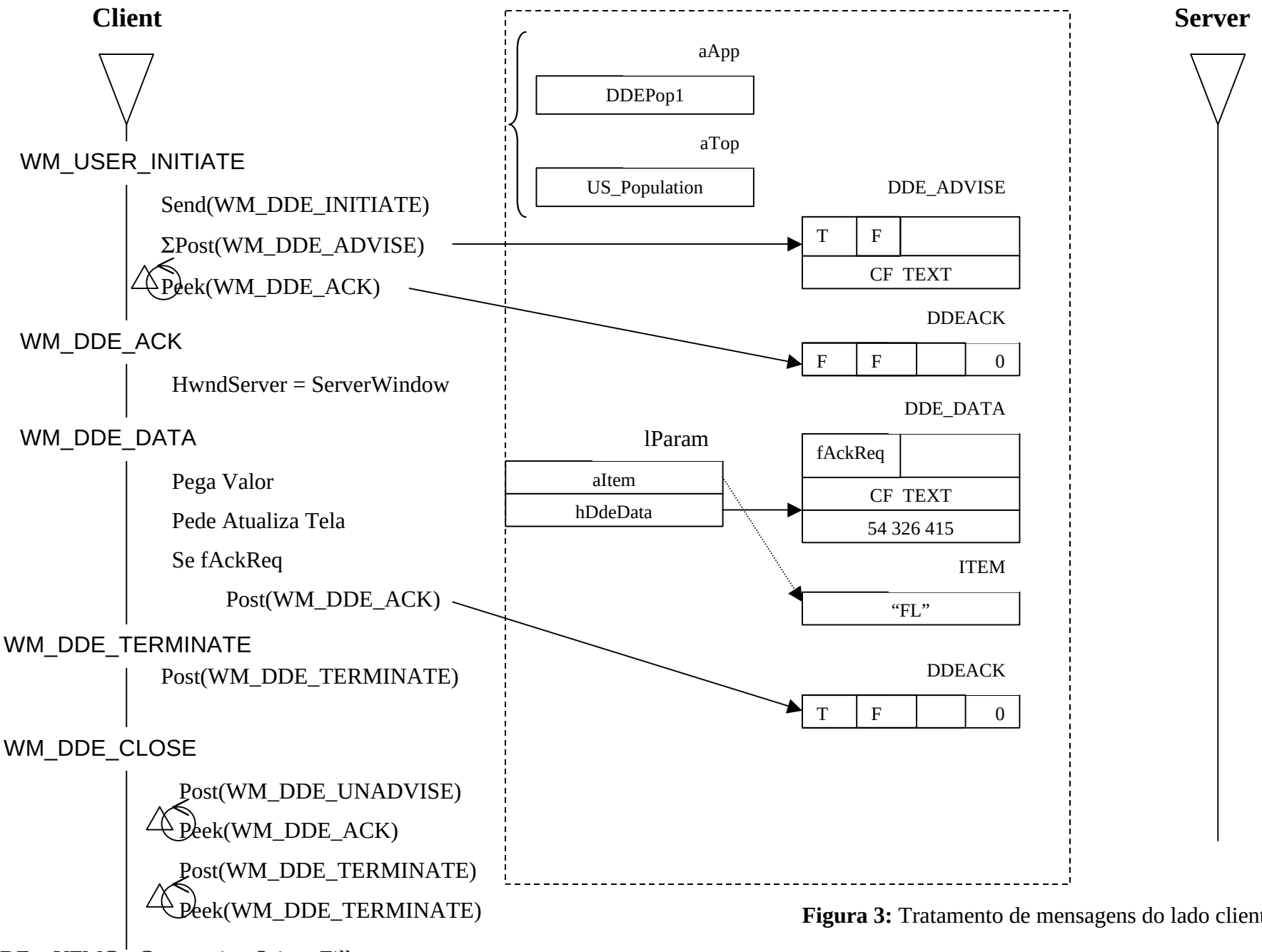


Figura 3: Tratamento de mensagens do lado cliente

DDEML – DDE MANAGEMENT LIBRARY

Uma evolução do uso de mensagens DDL foi a inclusão na API Win32 da biblioteca DDEML: *Dynamic Data Exchange Management L*ibrary (DDEML). DDEML é uma DLL usada pelo Win32 para compartilhar dados. Ao invés de enviar e processar mensagens DLL diretamente, uma aplicação usa as funções DDEML para gerenciar conversações DDE. DDEML supera as complexidades do protocolo DDE através do encapsulamento de mensagens, gerenciamento de átomos e de memória através de uma interface de chamada de funções.

Os arquivos DDEML.H e DDE.H definem cerca de 30 funções iniciadas pelo prefixo Dde. Um programa usando FDDEML requer uma função callback capaz de processar 16 tipos de transações, definidas no arquivo DDEML.H.

No jargão DDEML o nome da aplicação é chamado de nome do serviço.

- O programa se registra na biblioteca DDEML através da chamada de DdeInitialize e obtém um identificador de instancia da aplicação a ser utilizado nas demais chamadas de função. DdeInitialize registra uma função callback para processar as transações DDE.
- A passagem de dados entre aplicações é feita como:
  - Um programa cria um handle para um buffer contendo os dados usando DdeCreateDataHandle.
  - O programa que recebe os dados busca os dados através do handle usando DdeAccessData ou DdeGetData.
  - O handle de dados é liberado através de DdeFreeDataHandle.
- Ao terminar, o programa chama DdeUnitialize.

Existem uma série de funções para manipular strings através de handles:
- DdeCreateStringHandle: cria um handle, dado um string.
- DdeQueryStringHandle: obtém um string a partir de um handle.
- DdeCmpStringHandles: compara strings dados seus handles.
- DdeFreeStringHandle: libera um string, dado o handle.
- DdeKeepStringHandle: mantém um handle para string válido.

A título de ilustração vamos detalhar a função DdeQueryString:

D d e Q u e r y S t r i n g

```
DWORD DdeQueryString(

DWORD idInst,     // identificador de instância
HSZ hsz,          // Handle para string, retornado por DdeCreateStringHandle
LPTSTR psz,       // Apontador para buffer destino
DWORD cchMax,     // Comprimento do buffer
```

```
int iCodePage        // CP_WINANSI or CP_WINUNICODE
);
```

Retorno da função:

Status	**Interpretação**	
0L	Erro	
<> 0L	psz é um apontador válido para buffer	Tamanho do string retornado (não considera o caracter \0)
	psz = NULL	Comprimento do texto associado ao parâmetro hsz.

Exemplo:

```
DWORD idInst ;
char szItem[10];
SZ hszService;

// Inicializa para usar Dde e obtém DDEML instance handle em idInst
if (DdeInitialize (&idInst, (PFNCALLBACK) &DdeCallback,
   APPCLASS_STANDARD | APPCMD_CLIENTONLY, 0L)) {…};
// Cria string e obtém o handle
hszService = DdeCreateStringHandle (idInst, szService, 0) ;
// Obtem string dado o handle
DdeQueryString (idInst, hszService, szItem, sizeof(szItem), 0) ;
```

Exemplo: DDEPOP2

```
/*------------------------------------------------------------------------
DDEPOP2.C -- DDEML Server for Population Data
(c) Charles Petzold, 1996
  ------------------------------------------------------------------------*/

#include <windows.h>
#include <ddeml.h>
#include <string.h>
#include "ddepop.h"

#define WM_USER_INITIATE (WM_USER + 1)
// Mensagem criada pelo usuário
#define ID_TIMER          1

LRESULT  CALLBACK WndProc        (HWND, UINT, WPARAM, LPARAM) ;
// Cabeçalho da função callback que irá tratar as mensagens Dde
HDDEDATA CALLBACK DdeCallback (UINT, UINT, HCONV, HSZ, HSZ,
                                          HDDEDATA, DWORD, DWORD) ;

char     szAppName[] = "DdePop2" ;
char     szTopic[]   = "US_Population" ;
DWORD       idInst ;        // Handle para instância DDE: Global
HINSTANCE hInst ;
HWND         hwnd ;
```

```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
     MSG         msg ;
     WNDCLASSEX wndclass ;

     wndclass.cbSize        = sizeof (wndclass) ;
     wndclass.style         = 0 ;
     wndclass.lpfnWndProc   = WndProc ;
     wndclass.cbClsExtra    = 0 ;
     wndclass.cbWndExtra    = 0 ;
     wndclass.hInstance     = hInstance ;
     wndclass.hIcon         = LoadIcon (hInstance, szAppName) ;
     wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
     wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
     wndclass.lpszMenuName       = NULL ;
     wndclass.lpszClassName      = szAppName ;
     wndclass.hIconSm       = LoadIcon (hInstance, szAppName) ;

     RegisterClassEx (&wndclass) ;

     hwnd = CreateWindow (  szAppName, "DDEML Population Server",
                            WS_OVERLAPPEDWINDOW,
                            CW_USEDEFAULT, CW_USEDEFAULT,
                            CW_USEDEFAULT, CW_USEDEFAULT,
                            NULL, NULL, hInstance, NULL) ;

     // Exibe janela minimizada
     ShowWindow (hwnd, SW_SHOWMINNOACTIVE) ;
     UpdateWindow (hwnd) ;

     // Inicializa para usar DDEML: fornece função callback
     // idInst é o handle da instância DDEML
     if (DdeInitialize (&idInst, ( PFNCALLBACK) &DdeCallback,
                            CBF_FAIL_EXECUTES | CBF_FAIL_POKES |
                            CBF_SKIP_REGISTRATIONS |
                            CBF_SKIP_UNREGISTRATIONS, 0))
     {
         MessageBox (hwnd, "Could not initialize server!",
         szAppName, MB_ICONEXCLAMATION | MB_OK) ;

         DestroyWindow (hwnd) ;
         return FALSE ;
     }

     // Programa timer para gera mensagem WM_TIMER a cada 5s
     SetTimer (hwnd, ID_TIMER, 5000, NULL) ;

     // Outras inicializações serão realizadas em WndProc
     SendMessage (hwnd, WM_USER_INITIATE, 0, 0L) ;

     while (GetMessage (&msg, NULL, 0, 0))
     {
          TranslateMessage (&msg) ;
          DispatchMessage (&msg) ;
     }

     // Clean up
```

```
        DdeUninitialize (idInst) ;
        KillTimer (hwnd, ID_TIMER) ;

        return msg.wParam ;
}       // WinMain


// Retorna o índice de um estado dado o string com a sigla
int GetStateNumber (UINT iFmt, HSZ hszItem)
{
        char szItem[32] ;
        int  i ;

        if (iFmt != CF_TEXT) // Verifica se o formato é o único esperado
                return -1 ;
        // Busca string
        DdeQueryString (idInst, hszItem, szItem, sizeof (szItem), 0) ;
        // Compara se está na lista e retorna a ordem em que se encontra
        for (i = 0 ; i < NUM_STATES ; i++)
                if (strcmp (szItem, pop[i].szState) == 0)
                        break ;

        if (i >= NUM_STATES)
                return -1 ;

        return i ;
}       // GetStateNumber


HDDEDATA CALLBACK DdeCallback (   UINT iType, UINT iFmt, HCONV hConv,
                                  HSZ hsz1, HSZ hsz2, HDDEDATA hData,
                                  DWORD dwData1, DWORD dwData2)
{
        char szBuffer[32] ;
        int  i ;

        switch (iType)
        {
            case XTYP_CONNECT:
                // O cliente emite a mensagem WM_DDE_INITIATE
                // identificando o aplicativo e o tópico desejado.
                // O servidor responde com WM_DDE_ACK.
                // hsz1 = topic
                // hsz2 = service
                DdeQueryString (idInst, hsz2, szBuffer, sizeof (szBuffer), 0) ;
                if (0 != strcmp (szBuffer, szAppName)) // DdePop2
                        return FALSE ;

                DdeQueryString (idInst, hsz1, szBuffer, sizeof (szBuffer), 0) ;
                if (0 != strcmp (szBuffer, szTopic))       // US_Population
                        return FALSE ;

                return (HDDEDATA) TRUE ; // conversação teve início

            case XTYP_ADVSTART:
                // hsz1 = topic
                // hsz2 = item
                // Verifica se é um estado válido e retorna número de ordem
                if (-1 == (i = GetStateNumber (iFmt, hsz2)))
                        return FALSE ;
```

```
                pop[i].lPopLast = 0;
                PostMessage (hwnd, WM_TIMER, 0, 0L) ;

                return (HDDEDATA) TRUE ;

            case XTYP_REQUEST :
            case XTYP_ADVREQ :
                // hsz1 = topic
                // hsz2 = item
                // Checa quanto a formato e item de dados
                if (-1 == (i = GetStateNumber (iFmt, hsz2)))
                        return NULL;

                // Gera string com valor da população
                wsprintf (szBuffer, "%ld\r\n", pop[i].lPop) ;
                //
                return DdeCreateDataHandle (
                idInst, // Instância DDEML
                (unsigned char *) szBuffer, // Buffer com dado a ser copiado para obj DDE
                strlen (szBuffer) + 1,
                0,              // Offset a partir do início do buffer
                hsz2,  // Item
                CF_TEXT, // Formato de dados
                0) ;            // Flags

            case XTYP_ADVSTOP:
                // hsz1 = topic
                // hsz2 = item
                // Checa quanto a formato e item de dados
                if (-1 == (i = GetStateNumber (iFmt, hsz2)))
                        return FALSE ;
                return (HDDEDATA) TRUE ;
        }

        return NULL ;
}


LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam,
LPARAM lParam)
{
        static HSZ hszService, hszTopic ;
        HSZ             hszItem ;
        int             i ;

        switch (iMsg)
        {
            case WM_USER_INITIATE :
                InitPops () ;
                // Cria handle para strings: nome do serviço do servidor e nome do tópico
                hszService = DdeCreateStringHandle (idInst, szAppName, 0) ;
                hszTopic   = DdeCreateStringHandle (idInst, szTopic,   0) ;
                // Registra o nome do serviço
                // As transações serão dirigidas para a callback associada
                DdeNameService (idInst, hszService, NULL, DNS_REGISTER) ;
                return 0 ;

            case WM_TIMER :
            case WM_TIMECHANGE :
```

```
                    // Atualiza cálculo das populações
                    CalcPops () ;

                    for (i = 0 ; i < NUM_STATES ; i++)
                        if (pop[i].lPop != pop[i].lPopLast) // se população mudou
                        {
                            hszItem = DdeCreateStringHandle (idInst, pop[i].szState, 0) ;

                            // Post mensagem DDE_ADVISE com tópico a ser atualizado
                            DdePostAdvise (idInst, hszTopic, hszItem) ;
                            // DdePostAdvise chama sua própria função callback com uma
                            // transação XTYP_ADVREQ para cada conversação onde o cliente
                            // pediu um hot/warm link para o estado que variou

                            DdeFreeStringHandle (idInst, hszItem) ;

                            pop[i].lPopLast = pop[i].lPop ; // atualiza população anterior
                        }
                        return 0 ;

                case WM_QUERYOPEN :
                    return 0 ;

                case WM_DESTROY :
                    // Desfaz registro da aplicação
                    DdeNameService (idInst, hszService, NULL, DNS_UNREGISTER) ;
                    DdeFreeStringHandle (idInst, hszService) ;
                    DdeFreeStringHandle (idInst, hszTopic) ;

                    PostQuitMessage (0) ;
                    return 0 ;
                }
            return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
        } // WndProc
```

Exemplo: SHOWPOP2

```
/*--------------------------------------------------------------------
  SHOWPOP2.C -- DDEML Client using DDEPOP2
 (c) Charles Petzold, 1996
 --------------------------------------------------------------------*/
#include <windows.h>
#include <ddeml.h>
#include <stdlib.h>
#include <string.h>
#include "showpop.h"

#define WM_USER_INITIATE (WM_USER + 1)
#define DDE_TIMEOUT    3000

LRESULT  CALLBACK WndProc         (HWND, UINT, WPARAM, LPARAM) ;
HDDEDATA CALLBACK DdeCallback (       UINT, UINT, HCONV, HSZ, HSZ,
                                      HDDEDATA, DWORD, DWORD) ;

char  szAppName[] = "ShowPop2" ;
DWORD idInst ;
HCONV hConv ;
HWND  hwnd ;
```

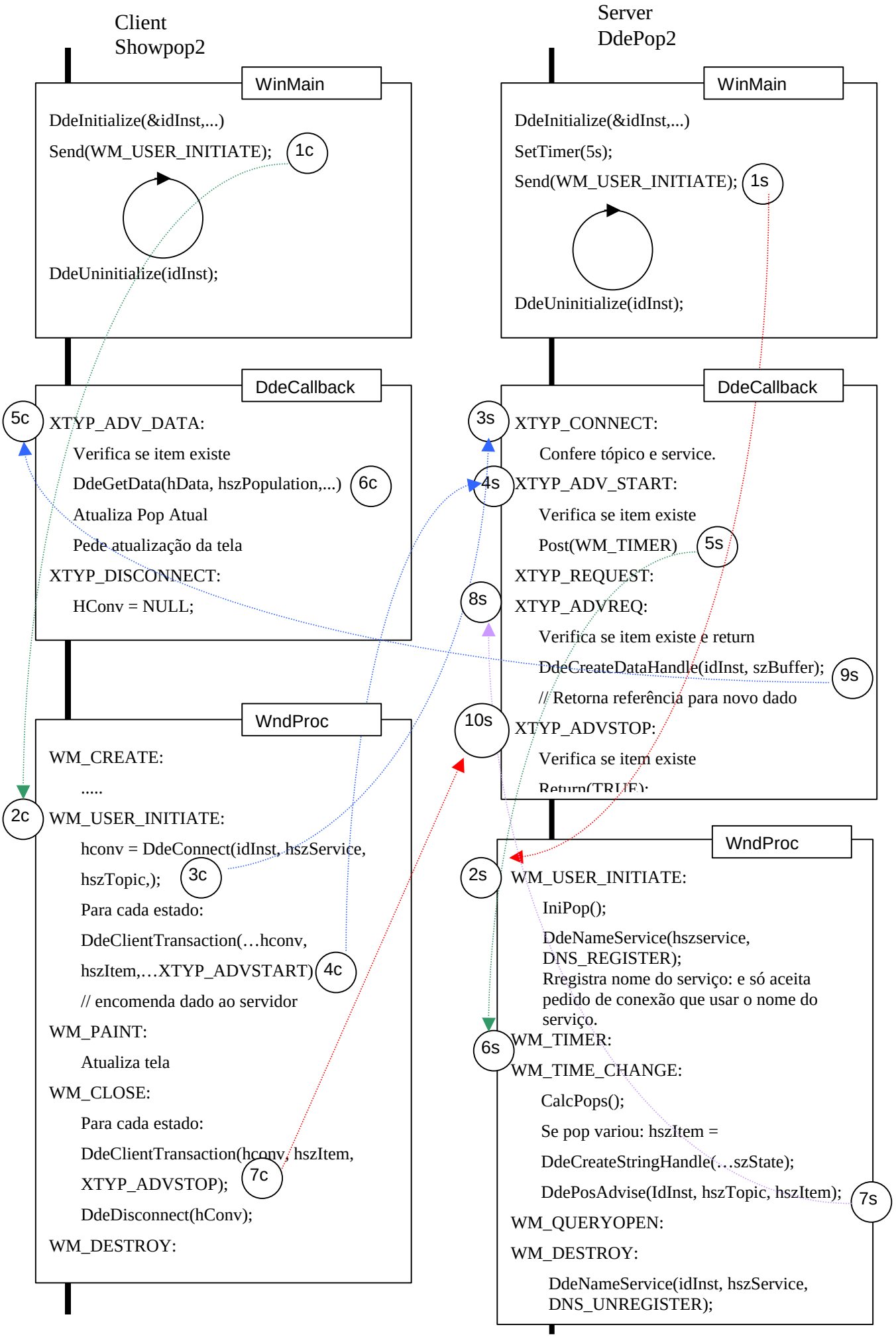
```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                PSTR szCmdLine, int iCmdShow)
{
        MSG         msg ;
        WNDCLASSEX wndclass ;

        wndclass.cbSize         = sizeof (wndclass) ;
        wndclass.style          = CS_HREDRAW | CS_VREDRAW ;
        wndclass.lpfnWndProc    = WndProc ;
        wndclass.cbClsExtra     = 0 ;
        wndclass.cbWndExtra     = 0 ;
        wndclass.hInstance      = hInstance ;
        wndclass.hIcon          = LoadIcon (hInstance, szAppName) ;
        wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW) ;
        wndclass.hbrBackground  = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
        wndclass.lpszMenuName = NULL ;
        wndclass.lpszClassName = szAppName ;
        wndclass.hIconSm    = LoadIcon (hInstance, szAppName) ;

        RegisterClassEx (&wndclass) ;
        hwnd = CreateWindow (  szAppName, "DDEML Client - US Population",
                               WS_OVERLAPPEDWINDOW,
                               CW_USEDEFAULT, CW_USEDEFAULT,
                               CW_USEDEFAULT, CW_USEDEFAULT,
                               NULL, NULL, hInstance, NULL) ;

        ShowWindow (hwnd, iCmdShow) ;
        UpdateWindow (hwnd) ;

        // Inicializa para usar e obtém DDEML instance handle em idInst
        if (DdeInitialize (&idInst, (PFNCALLBACK) &DdeCallback,
                APPCLASS_STANDARD | APPCMD_CLIENTONLY, 0L))
        {
            MessageBox (   hwnd, "Could not initialize client!",
                                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
            DestroyWindow (hwnd) ;
            return FALSE ;
        }

        // Start things going
        SendMessage (hwnd, WM_USER_INITIATE, 0, 0L) ;
        while (GetMessage (&msg, NULL, 0, 0))
        {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }

        // Uninitialize DDEML
        DdeUninitialize (idInst) ;

        return msg.wParam ;
}       // WinMain


HDDEDATA CALLBACK DdeCallback (     UINT iType, UINT iFmt, HCONV hConv,
                                    HSZ hsz1, HSZ hsz2, HDDEDATA hData,
                                    DWORD dwData1, DWORD dwData2)
{
        char szItem[10], szPopulation[16] ;
```

```
        int  i ;

        switch (iType)
        {
          case XTYP_ADVDATA:
                // hsz1  = topic
                // hsz2  = item
                // hData = data

                // Verifica formato
                if (iFmt != CF_TEXT)
                        return DDE_FNOTPROCESSED;

                // Verifica se estado existe
                DdeQueryString (idInst, hsz2, szItem, sizeof(szItem), 0) ;
                for (i = 0 ; i < NUM_STATES ; i++)
                        if (strcmp (szItem, pop[i].szAbb) == 0)
                        break ;

                if (i >= NUM_STATES)
                        return DDE_FNOTPROCESSED ;

                // Store the data and invalidate the window
                DdeGetData (hData, (unsigned char *) szPopulation,
                sizeof (szPopulation), 0) ;

                pop[i].lPop = atol (szPopulation) ;

                InvalidateRect (hwnd, NULL, FALSE) ;

                return (HDDEDATA) DDE_FACK ;

          case XTYP_DISCONNECT :
                hConv = NULL ;
                MessageBox (hwnd, "O servidor se disconectou.",
                        szAppName, MB_ICONASTERISK | MB_OK) ;

                return NULL ;
        }

        return NULL ;
}


LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam,
LPARAM lParam)
{
        static char szService[] = "DdePop2", szTopic[]   = "US_Population" ;
        static long cxChar, cyChar ;
        char            szBuffer[24] ;
        HDC             hdc ;
        HSZ             hszService, hszTopic, hszItem ;
        PAINTSTRUCT     ps ;
        int             i ;
        long            x, y ;
        TEXTMETRIC      tm ;

        switch (iMsg)
        {
            case WM_CREATE :
```

```
            hdc = GetDC (hwnd) ;
            GetTextMetrics (hdc, &tm) ;
            cxChar = tm.tmAveCharWidth ;
            cyChar = tm.tmHeight + tm.tmExternalLeading ;
            ReleaseDC (hwnd, hdc) ;

            return 0 ;

        case WM_USER_INITIATE :
            // Try connecting
            hszService = DdeCreateStringHandle (idInst, szService, 0) ;
            hszTopic   = DdeCreateStringHandle (idInst, szTopic,   0) ;

            // Conecta e obtém handle para coversação
            hConv = DdeConnect (idInst, hszService, hszTopic, NULL) ;

            // Se handle inválido, carrega aplicação e tenta de novo
            if (hConv == NULL)
            {
                    WinExec (szService, SW_SHOWMINNOACTIVE) ;
                    hConv = DdeConnect (idInst, hszService, hszTopic, NULL) ;
            }

            // Free the string handles
            DdeFreeStringHandle (idInst, hszService) ;
            DdeFreeStringHandle (idInst, hszTopic) ;

            // If still not connected, display message box
            if (hConv == NULL)
            {
                    MessageBox (hwnd, "Nao consigo conectar com DDEPOP2.EXE!",
                    szAppName, MB_ICONEXCLAMATION | MB_OK) ;

                    return 0 ;
            }

            // Estabelece hot links para os 50 estados + USA + ColumbiaDistrict
            for (i = 0; i < NUM_STATES; i++)
            {
                    hszItem = DdeCreateStringHandle (idInst, pop[i].szAbb, 0) ;
                    // Encomenda os dados
                    DdeClientTransaction (NULL, 0, hConv, hszItem, CF_TEXT,
                    XTYP_ADVSTART | XTYPF_ACKREQ,
                    DDE_TIMEOUT, NULL) ;
                    DdeFreeStringHandle (idInst, hszItem) ;
            } // for

            if (i < NUM_STATES)
            {
                    MessageBox (hwnd, "Falha em WM_DDE_ADVISE!",
                                    szAppName, MB_ICONEXCLAMATION | MB_OK) ;
            }

            return 0 ;

        case WM_PAINT :
            hdc = BeginPaint (hwnd, &ps) ;
            for (i = 0 ; i < NUM_STATES ; i++)
            {
                if (i < (NUM_STATES + 1) / 2)
```

```
                    {
                        x = cxChar ;
                        y = i * cyChar ;
                    }
                    else
                    {
                        x = 44 * cxChar ;
                        y = (i - (NUM_STATES + 1) / 2) * cyChar ;
                    }

                    TextOut (hdc, x, y, szBuffer,
                    wsprintf (szBuffer, "%-20s", (PSTR) pop[i].szState)) ;

                    x += 36 * cxChar ;

                    SetTextAlign (hdc, TA_RIGHT | TA_TOP);
                    TextOut (hdc, x, y, szBuffer,
                    wsprintf (szBuffer, "%10ld", pop[i].lPop)) ;

                    SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
                }

                EndPaint (hwnd, &ps) ;
                return 0 ;

            case WM_CLOSE :
                if (hConv == NULL) break ;

                // Stop the advises
                for (i = 0 ; i < NUM_STATES ; i++)
                {
                    hszItem = DdeCreateStringHandle (idInst, pop[i].szAbb, 0) ;
                    DdeClientTransaction (NULL, 0, hConv, hszItem, CF_TEXT,
                            XTYP_ADVSTOP, DDE_TIMEOUT, NULL) ;
                    DdeFreeStringHandle (idInst, hszItem) ;
                }

                // Disconnect the conversation
                DdeDisconnect (hConv) ;
                break ;                         // for default processing

            case WM_DESTROY :
                PostQuitMessage (0) ;
                return 0 ;
        }
        return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
} // WndProc
```

O gráfico na próxima página mostra todo o fluxo de mensagens entre cliente e servidor passando pelo MLDDE.


Client
Showpop2
WinMain
DdeInitialize(&idInst,...)
Send(WM_USER_INITIATE);
1c
DdeUninitialize(idInst);
DdeCallback
5c
XTYP_ADV_DATA:
Verifica se item existe
DdeGetData(hData, hszPopulation,...)
6c
Atualiza Pop Atual
Pede atualização da tela
XTYP_DISCONNECT:
HConv = NULL;
WndProc
WM_CREATE:
.....
2c
WM_USER_INITIATE:
hconv = DdeConnect(idInst, hszService,
hszTopic,);
3c
Para cada estado:
DdeClientTransaction(…hconv,
hszItem,…XTYP_ADVSTART)
4c
// encomenda dado ao servidor
WM_PAINT:
Atualiza tela
WM_CLOSE:
Para cada estado:
DdeClientTransaction(hconv, hszItem,
XTYP_ADVSTOP);
7c
DdeDisconnect(hConv);
WM_DESTROY:
Server
DdePop2
WinMain
DdeInitialize(&idInst,...)
SetTimer(5s);
Send(WM_USER_INITIATE);
1s
DdeUninitialize(idInst);
DdeCallback
3s
XTYP_CONNECT:
Confere tópico e service.
4s
XTYP_ADV_START:
Verifica se item existe
Post(WM_TIMER)
5s
XTYP_REQUEST:
8s
XTYP_ADVREQ:
Verifica se item existe e return
DdeCreateDataHandle(idInst, szBuffer);
9s
// Retorna referência para novo dado
10s
XTYP_ADVSTOP:
Verifica se item existe
Return(TRUE);
WndProc
2s
WM_USER_INITIATE:
IniPop();
DdeNameService(hszservice,
DNS_REGISTER);
Rregistra nome do serviço: e só aceita
pedido de conexão que usar o nome do
serviço.
6s
WM_TIMER:
WM_TIME_CHANGE:
CalcPops();
Se pop variou: hszItem =
DdeCreateStringHandle(…szState);
DdePosAdvise(IdInst, hszTopic, hszItem);
7s
WM_QUERYOPEN:
WM_DESTROY:
DdeNameService(idInst, hszService,
DNS_UNREGISTER);

Exercícios

1) Modifique o programa servidor (DDEPOP1) de forma a que ele forneça um trace de todos as mensagens WM_DDE_XXX recebidas e das respostas enviadas.

2) Enumere as vantagens e desvantagens da comunicação DDE em relação a COM/DCOM.

3) Compare a comunicação DDE com outros tipos de comunicação que você conhece: pipes nomeados, mailslots.

4) Gere as fichas de detalhamento das seguintes funções:

   DdeCreateStringHandle

   DdeCmpStringHandle

   DdeFreeStringHandle

   DdeKeepStringHandle

5) Faça um programa cliente em MFC que permita programar a leitura a frio ou a quente da população de qualquer estado, a partir da digitação do aplicativo, tópico e item. Utilize a biblioteca DDEML. Construa uma interface simples, mas comunicativa.

Bibliografia

[Petzold 96] Charles Petzold, Programming Windows 95, Microsoft Press, 1996

[Technet 01] Network Dynamic Data Exchange, Microsoft TechNet, chapter 11, www.microsoft.com/technet/WFW/wfw31/5_ch11.asp?a=printable

[AngelFire 01] Dynamic Data Exchange, Dynamic Data Exchange (DDE) and NetDDE FAQ, Why it is not replaced by COM, how it works, network DDE, links to other sources of information, http://www.angelfire.com/biz/rhaminisys/ddeinfo.htm