

Arquitectura e Ingeniería de Computadores

4º Ingeniería Informática. UEx

Simulador SMPCache

Fecha de entrega	
20 de Mayo de 2011	
Nombre y apellidos de los alumnos del grupo	
Mario Corchero Jiménez	
Manuel Cantonero Chamorro	
Interfaz del simulador	

Observaciones del profesor							
Presentación	Manual de Usuario	Manual de Programador	Código	Pruebas de Calidad	Examen de Defensa	Otros	Calificación

Contenido

Introducción:	4
Bus.h	5
Bus.cpp	6
Cache.h	9
Cache.cpp	10
ErrorConfig.h	10
Procesador.h	21
Procesador.cpp	21
SMPCache.h	22
SMPCache.cpp	22
SMPCacheView.h	25
SMPCacheView.cpp	28
Manual del programador	44
Introducción	44
Análisis	44
Identificación de clases potenciales	44
Diagrama de modelo conceptual	44
Descripción de las clases Conceptuales	44
Diseño	45
Diagrama de clases	45
Algoritmos de especial interés	45
Definición de entradas/salidas	45
Variables o instancias más significativas	45
Lista de errores que el programa controla	45
Historial de desarrollo y Valoración personal	46
Manual de Usuario	47
Rectángulo rojo:	47
Datos:	47
Estadísticas:	47
Accesos a memoria:	47
Instrucciones:	48
Datos leídos:	48
Datos escritos:	48

Aciertos y fallos:	48
Control de ejecución:	48
Iniciar:	48
Pausa:	48
Ejecución completa:	48
Salir:	48
Ejemplo de ejecución:	48
Conclusión:	53

Introducción:

Se ha propuesto la realización de un simulador de memoria caché en multiprocesadores.

Observaremos como dentro de las posibles jerarquías de memorias la más importante es la de los multiprocesadores con memoria compartida por bus. A su vez implementaremos distintos protocolos de coherencia como son MSI, MESI y DRAGON, implementado como trabajan cada uno de estos protocolos. Terminamos esta breve introducción indicando que hemos realizado la práctica en el entorno de Visual Studio.

Listado del Código

Bus.h

```
#pragma once

#include "Cache.h"
#include <map>

class Bus
{
public:
    void RegistrarCache(Cache* cache);
    bool comparteCaches(long dirmem, int idPetidora);
    static Bus* getInstancia();
    bool realizarIteracion();
    void indicarProtocoloArbitracion(ARBITRACION pArbi);
    ~Bus();

    //variable de cambio de estado
    map<ESTADO_CACHE, map<ESTADO_CACHE, int>> cambios_estado;

    // numero de transiciones de estado
    int numTransBus;
    // numero de bloques tranferidos
    float numBloqTrans;
    // numero de transicioes de estado
    int numTranEstado;
    //aciertos en cache
    int aciertos;
    //fallos en cache
    int fallos;
    //lectura de instruccion
    int instLeidas;
    //accesos a memoria
    int memAccesos;
    //lecturas
    int lecturas;
    //escrituras
    int escrituras;
    //caches activas, con traza pendiente
    bool activo[8];
    Cache* caches[8];

    //Origen del bloque que necesita la cache. -1 = memoria
    int origDat;

    //variable para la actualizacion de la flecha del mensaje bus de
    la cache
    struct TipoFlechaMensajeCache{int id; MENSAJEBUS mensaje;};
    struct TipoFlechaMensajeCache flechaMensajeCache;

private:
    Bus(void);
    int usos[8];
    int ultAccesoAbus[8];
    int numCaches;
    static Bus* instancia;
    ARBITRACION pArbitracion;
};
```

Bus.cpp

```
#include "StdAfx.h"
#include "Bus.h"

Bus* Bus::instancia = NULL;

Bus::Bus(void)
{
    numCaches=0;
    numTransBus = 0;
    numBloqTrans =0;
    numTranEstado = 0;
    aciertos = 0;
    fallos = 0;
    instLeidas = 0;
    memAccesos = 0;
    lecturas = 0;
    escrituras = 0;
    aciertos = 0;
    fallos = 0;
    for(int i = 0; i<8;i++)
    {
        usos[i] = 0;
        ultAccesoAbus[i] = 0;
        activo[i] = true;
    }

    for(int h = 0; h< 6; h++)
    {
        for(int j = 0; j< 6; j++)
            cambios_estado[(ESTADO_CACHE)h][(ESTADO_CACHE)j] = 0;
    }
}

bool Bus::comparteCaches(long dirmem, int idPetidora)
{
    bool res = false;
    for(int i = 0;i<numCaches; i++)
    {
        if(caches[i]->getId() != idPetidora) res = res || caches[i]-
>BuscarDirMem(dirmem);
    }
    return res;
}

Bus* Bus::getInstancia()
{
    if(instancia == NULL) instancia = new Bus();
    return instancia;
}

void Bus::RegistrarCache(Cache* cache)
{
    caches[numCaches++] = cache;
}

//true = queda por ejecutar, false = trazas finalizadas
bool Bus::realizarIteracion()
{
    flechaMensajeCache.id = -1;
```

```

origDat = -1;
bool peticionesBus[8]={false};
int cachesAArbitrar[8];
int numCachesAArbitrar = 0;
for(int i =0;i< numCaches;i++)
    if(activo[i]) peticionesBus[i] = caches[i]->peticionBus();
    else peticionesBus[i] = false;

for(int i=0;i<numCaches;i++)
{
    if(activo[i]){
        if(peticionesBus[i])
cachesAArbitrar[numCachesAArbitrar++] = i;
        else if(caches[i]->ejecutar().tipo == M_VACIO)//no
guardamos el mensaje, pues va a ser NULL o VACIO
        {
            activo[i] = false;
        }
    }
}
if(numCachesAArbitrar != 0)
{
    //Aplicar el algoritmo
    int elegida;
    switch(pArbitracion)
    {
        case A_ALEATORIO:
            elegida = cachesAArbitrar[rand()%numCachesAArbitrar];
            break;
        case A_LRU:
            int minimo;
            minimo = ultAccesoAbus[cachesAArbitrar[0]];
            elegida = cachesAArbitrar[0];
            for(int i = 1 ; i< numCachesAArbitrar; i++)
            {
                if( ultAccesoAbus[cachesAArbitrar[i]] < minimo)
                {
                    elegida = cachesAArbitrar[i];
                    minimo =
ultAccesoAbus[cachesAArbitrar[i]];
                }
            }
            break;
        case A_LFU:
            float minimo2;
            minimo2 = usos[cachesAArbitrar[0]]/(float)numTransBus;
            elegida = cachesAArbitrar[0];
            for(int i = 1 ; i< numCachesAArbitrar; i++)
            {
                if(usos[cachesAArbitrar[i]]/(float)numTransBus
< minimo2)
                {
                    elegida = cachesAArbitrar[i];
                    minimo2 =
usos[cachesAArbitrar[i]]/(float)numTransBus;
                }
            }
            break;
    }
    MENSAJEBUS mensaje = caches[elegida]->ejecutar();
    numTransBus++;
}

```

```

        if(mensaje.tipo != M_BUSUPD)
            numBloqTrans++;
        ultAccesoAbus[elegida] = numTransBus;
        usos[elegida]++;
        flechaMensajeCache.id = caches[elegida]->getId();
        flechaMensajeCache.mensaje = mensaje;
        if(mensaje.tipo == M_BUSRDUPLD) //si se produce 1 busrd y
busupd son 2 mensajes realmente
        {
            numTransBus++;
            mensaje.tipo = M_BUSRDUPLD;
            for(int i = 0; i < numCaches; i++)
            {
                if(i != elegida)
                {
                    caches[i]->tratarMensajeBus(mensaje);
                }
            }
            mensaje.tipo = M_BUSUPD;
            for(int i = 0; i < numCaches; i++)
            {
                if(i != elegida)
                {
                    caches[i]->tratarMensajeBus(mensaje);
                }
            }
        }
    else
    {
        for(int i = 0; i < numCaches; i++)
        {
            if(i != elegida)
            {
                caches[i]->tratarMensajeBus(mensaje);
            }
        }
    }

    //Vemos si ha terminado
    bool nofin = false;
    for(int i = 0; i < numCaches; i++)
    {
        nofin = nofin || activo[i];
    }
    if(!nofin)
    {
        AfxMessageBox(_T("A continuacion se realizaran las
postescrituras"));
        for(int i = 0; i < numCaches; i++)
            caches[i]->postEscrituras();
    }
    return nofin;
}

void Bus::indicarProtocoloArbitracion(ARBITRACION parbi)
{
    this->pArbitracion=parbi;
}

Bus::~~Bus()
{

```



```

        for(int i = 0; i<numCaches; i++)
            delete caches[i];
        instancia = NULL;
    }

```

Cache.h

```

#pragma once

#include <vector>

#include "constantes.h"
#include "Procesador.h"

using namespace std;

class Cache
{
public:
    Cache(int id, int bloqCaches, FCORRESPONDENCIA funcion, int
numCjtos, AREMPLAZAMIENTO algoritmo, PROTOCOLO pCoherencia, int
bloqMem, int palBloque, CString ficheroTraza);
    MENSAJEBUS ejecutar();
    bool peticionBus();
    bool BuscarDirMem(long dirMem);
    inline int getId(){return id_cache;}
    void tratarMensajeBus(MENSAJEBUS mensaje);
    void postEscrituras();

    char textoCache[100];

private:
    int id_cache;
    vector<vector<ENTRADA>> datos;
    int nBloques;
    int numCjtos;
    int bloqPorCjto;
    int bloqMem;
    int palBloque;
    PROTOCOLO pCoherencia;
    FCORRESPONDENCIA fCorrespondencia;
    AREMPLAZAMIENTO remplazamiento;
    Procesador proc;

    bool enEspera;
    LINEATRAZA tActual;

    bool BuscarBloque(int idBloque);
    ENTRADA *getEntrada(long dirMem);
    bool IntroducirBloque(int idBloque);

    int dirToEtiqueta(long dirmem);
    int dirToCjto(long dirmem);
    int bloqueToEtiqueta(int dirbloque);
    int bloqueToCjto(int dirbloque);
    int bloqueMasAntiguo(int idConjunto);
    int bloqueMasNuevo(int idConjunto);
    int bloqueLFU(int idConjunto);
    MENSAJEBUS read(long dirmem);
    MENSAJEBUS readX(long dirmem);

```

```

        void incrementarBV();
};

```

Cache.cpp

```

#include "StdAfx.h"
#include <algorithm>
#include "Cache.h"
#include "Bus.h"

```

```

Cache::Cache(int id, int bloqCaches, FCORRESPONDENCIA funcion, int
numCjtos, AREEMPLAZAMIENTO algoritmo, PROTOCOLO pCoherencia , int
bloqMem, int palBloque, CString ficheroTraza)
:proc(ficheroTraza)
{
    id_cache = id;
    this->nBloques = bloqCaches;
    if(numCjtos == 0) numCjtos = 1;
    if(funcion == F_DIRECTA)
    {
        bloqPorCjto = 1;
        numCjtos = bloqCaches;
    }
    else
    {
        bloqPorCjto = bloqCaches / numCjtos;
    }
    this->numCjtos = numCjtos;
    this->palBloque = palBloque;
    this->bloqMem = bloqMem;

    //Inicializacion de datos
    ENTRADA aux;
    aux.bv=0;
    aux.usados = 0;
    aux.estado = C_I;
    aux.ultimoUso = 0;
    vector<ENTRADA> vecAux;
    for(int j=0;j<bloqPorCjto;j++)
        vecAux.push_back(aux);
    for(int i=0;i<numCjtos;i++)
        datos.push_back(vecAux);
    fCorrespondencia = funcion;

    Bus::getInstancia()->RegistrarCache(this);

    fCorrespondencia = funcion;
    this->pCoherencia = pCoherencia;
    this->reemplazamiento = algoritmo;
    enEspera = false;
}

int Cache::bloqueToEtiqueta(int dirbloque)
{
    return dirbloque / numCjtos;
}

```

```

}

int Cache::bloqueToCjto(int dirbloque)
{
    return dirbloque % numCjtos;
}

int Cache::dirToEtiqueta(long dirMem)
{
    return bloqueToEtiqueta(dirMemToBloq(dirMem, bloqMem, palBloque));
}

int Cache::dirToCjto(long dirMem)
{
    return bloqueToCjto(dirMemToBloq(dirMem, bloqMem, palBloque));
}

bool Cache::BuscarBloque(int idBloque)
{
    if(buscarEtiqueta(datos[bloqueToCjto(idBloque)], bloqueToEtiqueta(idBloque))) return true;
    return false;
}

bool Cache::IntroducirBloque(int idBloque)
{
    bool finded = false;
    for(int i = 0; i < bloqPorCjto && !finded; i++)
        if(datos[bloqueToCjto(idBloque)][i].bv == 0)
        {
            datos[bloqueToCjto(idBloque)][i].bv = 1;
            datos[bloqueToCjto(idBloque)][i].usados = 0;
            datos[bloqueToCjto(idBloque)][i].etiqueta =
bloqueToEtiqueta(idBloque);
            finded = true;
        }
    if(finded) return true; //fallo forzoso

    //REEMPLAZO
    //si esta en M o S hay que hacer postescritura
    int bloqdestino = 0;
    switch(reemplazamiento)
    {
        case R_ALEATORIO:
            bloqdestino = rand() % bloqPorCjto;
            break;
        case R_LFU:
            bloqdestino = bloqueLFU(bloqueToCjto(idBloque));
            break;
        case R_LRU:
            bloqdestino =
bloqueMasAntiguamenteUsado(bloqueToCjto(idBloque));
            break;
        case R_FIFO:
            bloqdestino =
bloqueMasAntiguo(bloqueToCjto(idBloque));
            break;
    }
    //tratamiento del remplazamiento y la postescritura si necesario

```

```

        ESTADO_CACHE estado =
datos[bloqueToCjto(idBloque)][bloqdestino].estado;
        if(estado == C_M || estado == C_SM)
        {
            Bus::getInstancia()->numBloqTrans++;
            Bus::getInstancia()->numTransBus++;
            Bus::getInstancia()->cambios_estado[estado][C_I]++;
            Bus::getInstancia()->numTranEstado++;
        }

        datos[bloqueToCjto(idBloque)][bloqdestino].estado = C_I;
        datos[bloqueToCjto(idBloque)][bloqdestino].bv = 1;
        datos[bloqueToCjto(idBloque)][bloqdestino].usados = 0;
        datos[bloqueToCjto(idBloque)][bloqdestino].etiqueta =
bloqueToEtiqueta(idBloque);
        datos[bloqueToCjto(idBloque)][bloqdestino].ultimoUso =
Bus::getInstancia()->memAccesos;
        return false; //fallo que implica la salida de un dato
    }

int Cache::bloqueMasAntiguo(int idConjunto)
{
    int aux = 0;
    for(int i = 0; i < bloqPorCjto; i++)
    {
        if(datos[idConjunto][i].bv > datos[idConjunto][aux].bv) aux
= i;
    }
    return aux;
}

int Cache::bloqueMasAntiguamenteUsado(int idConjunto)
{
    int aux = 0;
    for(int i = 0; i < bloqPorCjto; i++)
    {
        if(datos[idConjunto][i].ultimoUso <
datos[idConjunto][aux].ultimoUso) aux = i;
    }
    return aux;
}

int Cache::bloqueLFU(int idConjunto)
{
    int aux = 0;
    for(int i = 0; i < bloqPorCjto; i++)
    {
        if(datos[idConjunto][i].bv != 0 &&
datos[idConjunto][i].usados / (1.0 * datos[idConjunto][i].bv) <
datos[idConjunto][aux].usados / (1.0 * datos[idConjunto][aux].bv))
            aux = i;
    }
    return aux;
}

int Cache::bloqueMasNuevo(int idConjunto)
{
    int aux = 0;
    for(int i = 0; i < bloqPorCjto; i++)
    {

```

```

        if(datos[idConjunto][i].bv < datos[idConjunto][aux].bv &&
datos[idConjunto][i].bv !=0) aux = i;
    }
    return aux;
}

void Cache::incrementarBV()
{
    for(int i = 0; i < numCjtos; i++)
        for(int j = 0; j < bloqPorCjto; j++)
            if(datos[i][j].bv != 0)datos[i][j].bv++;
}

MENSAJEBUS Cache::ejecutar()
{
    enEspera = false;
    if(tActual.tIns == IVACIO)
    {
        MENSAJEBUS mensajevacio;
        mensajevacio.tipo= M_VACIO;
        return mensajevacio;
    }
    //si no tiene la
    if(!BuscarDirMem(tActual.dirMem))
    {
        IntroducirBloque(dirMemToBloq(tActual.dirMem,bloqMem,palBloque));
        MENSAJEBUS mensaje;
        mensaje.dirmem = tActual.dirMem;
        if(tActual.tIns != IESCRITURA)
            mensaje.tipo = M_BUSRD;
        else
            mensaje.tipo = M_BUSRDX;
        Bus::getInstancia()->fallos++;
    }else
    {
        Bus::getInstancia()->aciertos++;
    }
    Bus::getInstancia()->memAccesos++;
    getEntrada(tActual.dirMem)->ultimoUso = Bus::getInstancia()-
>memAccesos;
    getEntrada(tActual.dirMem)->usados++;
    switch(tActual.tIns)
    {
        case ICAPTURA: Bus::getInstancia()->instLeidas++;return
read(tActual.dirMem);break;
        case ILECTURA: Bus::getInstancia()->lecturas++;
            return read(tActual.dirMem);break;
        case IESCRITURA:Bus::getInstancia()->escrituras++;
            return readX(tActual.dirMem);break;
    }
}

MENSAJEBUS Cache::read(long dirmem)
{
    Bus::getInstancia()->numTranEstado++;

    incrementarBV();
    MENSAJEBUS mensaje;
    mensaje.dirmem = dirmem;
    ENTRADA *entrada = getEntrada(dirmem);

```

```

mensaje.dirmem = dirmem;
ESTADO_CACHE estado_siguiente;
estado_siguiente = C_NULL;
switch (pCoherencia)
{
    case P_MSI:
        switch (entrada->estado)
        {
            case C_I: mensaje.tipo = M_BUSRD;
                estado_siguiente = C_S;
                break;
            case C_S: mensaje.tipo= M_NULL;break;
            case C_M: mensaje.tipo= M_NULL;break;
        }
        break;
    case P_MESI:
        switch (entrada->estado)
        {
            case C_I: mensaje.tipo = M_BUSRD;
                if (Bus::getInstancia()->comparteCaches (dirmem,
id_cache))
                    estado_siguiente = C_S;
                else
                    estado_siguiente = C_E;
                break;
            case C_S: mensaje.tipo= M_NULL;
                break;
            case C_E: mensaje.tipo= M_NULL;
                break;
            case C_M: mensaje.tipo= M_NULL;
                break;
        }
        break;
    case P_DRAGON:
        switch (entrada->estado)
        {
            case C_I: mensaje.tipo=M_BUSRD;
                if (Bus::getInstancia()-
>comparteCaches (dirmem,id_cache))
                    estado_siguiente = C_SC;
                else
                    estado_siguiente = C_E;
                break;
            case C_E: mensaje.tipo = M_NULL;
                break;
            case C_M: mensaje.tipo = M_NULL;
                break;
            case C_SC: mensaje.tipo = M_NULL;
                break;
            case C_SM: mensaje.tipo = M_NULL;
                break;
        }
        break;
    }
    if (estado_siguiente == C_NULL)
        estado_siguiente = entrada->estado;
    Bus::getInstancia()->cambios_estado[entrada-
>estado][estado_siguiente]++;
    entrada->estado = estado_siguiente;
    return mensaje;
}

```

```

MENSAJEBUS Cache::readX(long dirmem)
{
    Bus::getInstancia()->numTranEstado++;

    incrementarBV();
    MENSAJEBUS mensaje;
    mensaje.dirmem = dirmem;
    ENTRADA* entrada = getEntrada(dirmem);
    ESTADO_CACHE estado_siguiente;
    estado_siguiente = C_NULL;
    switch (pCoherencia)
    {
        case P_MSI:
            switch(entrada->estado)
            {
                case C_I: mensaje.tipo = M_BUSRDX;
                    estado_siguiente = C_M;
                    break;
                case C_S: mensaje.tipo = M_BUSRDX;
                    estado_siguiente = C_M;
                    break;
                case C_M: mensaje.tipo= M_NULL;break;
            }
            break;
        case P_MESI:
            switch(entrada->estado)
            {
                case C_I: mensaje.tipo = M_BUSRDX;
                    estado_siguiente = C_M;
                    break;
                case C_S: mensaje.tipo = M_BUSRDX;
                    estado_siguiente = C_M;
                    break;
                case C_E: mensaje.tipo= M_NULL;
                    estado_siguiente = C_M;
                    break;
                case C_M: mensaje.tipo= M_NULL;
                    break;
            }
            break;
        case P_DRAGON:
            switch(entrada->estado)
            {
                case C_I:
                    if(Bus::getInstancia()-
>comparteCaches(dirmem,id_cache))
                    {
                        mensaje.tipo=M_BUSRDUPE;
                        Bus::getInstancia()-
>numBloqTrans+=1.0/palBloque;
                        estado_siguiente = C_SM;
                    }
                    else
                        estado_siguiente = C_M;
                    break;
                case C_E: estado_siguiente = C_M;mensaje.tipo =
M_NULL;
                    break;
                case C_M: mensaje.tipo = M_NULL;

```

```

        break;
    case C_SC:
        if (Bus::getInstancia() -
>comparteCaches(dirmem,id_cache))
            estado_siguiente = C_SM;
        else
            estado_siguiente = C_M;
            mensaje.tipo = M_BUSUPD;
            Bus::getInstancia() -
>numBloqTrans+=1.0/palBloque;
            break;
    case C_SM:
        if (Bus::getInstancia() -
>comparteCaches(dirmem,id_cache))
            mensaje.tipo = M_BUSUPD;
        else
        {
            estado_siguiente = C_M;
            mensaje.tipo = M_BUSUPD;
        }
        Bus::getInstancia() -
>numBloqTrans+=1.0/palBloque;
        break;
    }
    break;
}

if(estado_siguiente == C_NULL)
    estado_siguiente = entrada->estado;
    Bus::getInstancia()->cambios_estado[entrada-
>estado][estado_siguiente]++;
    entrada->estado = estado_siguiente;
    return mensaje;
}

void Cache::tratarMensajeBus(MENSAJEBUS mensaje)
{
    if(!BuscarDirMem(mensaje.dirmem)) return;
    Bus::getInstancia()->origDat = this->getId();
    Bus::getInstancia()->numTranEstado++;
    ESTADO_CACHE estado_siguiente;
    estado_siguiente = C_NULL;

    ENTRADA* entrada = getEntrada(mensaje.dirmem);
    switch (pCoherencia)
    {
        case P_MSI:
            switch(entrada->estado)
            {
                case C_I: break;
                case C_S: if(mensaje.tipo == M_BUSRDX)
                    {
                        estado_siguiente = C_I;entrada-
>bv=0;
                    }

                break;
            case C_M: if(mensaje.tipo == M_BUSRD)
                    {
                        estado_siguiente = C_S;
                        Bus::getInstancia() -
>numTransBus++;
                    }
            }
    }
}

```



```

        //tratar vaciado(flechas)
    }
    else
    {
        estado_siguiente = C_I;entrada-
        //tratar vaciado(flechas)
    }
}
break;
}
break;
case P_MESI:
    switch(entrada->estado)
    {
        case C_I: break;
        case C_S: if(mensaje.tipo == M_BUSRD){}
            else
            {
                estado_siguiente = C_I;entrada-
            }
            break;
        case C_E: if(mensaje.tipo == M_BUSRD)
            {
                estado_siguiente = C_S;
            }
            else
            {
                estado_siguiente = C_I;entrada-
            }
            break;
        case C_M: if(mensaje.tipo == M_BUSRD)
            {
                estado_siguiente = C_S;
                Bus::getInstancia()->numTransBus++;
                //tratar vaciado(flechas)
            }
            else
            {
                estado_siguiente = C_I;entrada-
                //tratar vaciado(flechas)
            }
            break;
    }
}
break;
case P_DRAGON:
    switch(entrada->estado)
    {
        case C_E: if(mensaje.tipo==
M_BUSRD)estado_siguiente = C_SC;break;
        case C_SC: if(mensaje.tipo== M_BUSUPD) {}
break;//      TODO: ACTUALIZA = SI
        case C_SM:
            if(mensaje.tipo== M_BUSUPD)
            {
                estado_siguiente = C_SC;
                //TODO: ACTUALiza
            }else
            {

```

```

        //TODO: datbus
    }
        break;
    case C_M: if(mensaje.tipo== M_BUSRD)
        estado_siguiente = C_SM;
        //TODO: datbus
        break;
    }
    break;
}
if(estado_siguiente == C_NULL)
    estado_siguiente = entrada->estado;
Bus::getInstancia()->cambios_estado[entrada->estado][estado_siguiente]++;
entrada->estado = estado_siguiente;
}

//Lee la siguiente linea y devuelve si necesitará acceso a bus
bool Cache::peticionBus()
{
    if(!enEspera)
    {
        tActual = proc.getLinea();
    }
    enEspera = true;
    if(tActual.tIns == IVACIO)
    {
        return false; //El procesador a terminado
    }
    if(!BuscarDirMem(tActual.dirMem))
    {
        if(tActual.tIns != IESCRITURA)

        sprintf_s(textoCache, "\nPrRd\n%d", dirMemToBloq(tActual.dirMem, bloqMem, palBloque));
        else

        sprintf_s(textoCache, "\nPrWr\n%d", dirMemToBloq(tActual.dirMem, bloqMem, palBloque));
        return true;
    }
    if(tActual.tIns != IESCRITURA)

    sprintf_s(textoCache, "\nPrRd\n%d", dirMemToBloq(tActual.dirMem, bloqMem, palBloque));
    else

    sprintf_s(textoCache, "\nPrWr\n%d", dirMemToBloq(tActual.dirMem, bloqMem, palBloque));
    //El bloque esta en la cache
    switch (pCoherencia)
    {
        case P_MSI:
            switch (getEntrada(tActual.dirMem)->estado)
            {
                case C_I: return true;
                case C_S: if(tActual.tIns == IESCRITURA) return true;
                        else return false;
                case C_M: return false;
            }
        break;
    }
}

```

```

        case P_MESI:
            switch(getEntrada(tActual.dirMem)->estado)
            {
                case C_I: return true;
                case C_S: if(tActual.tIns == IESCRITURA) return true;
                           else return false;
                case C_E: return false;
                case C_M: return false;
            }
        break;
        case P_DRAGON:
            switch(getEntrada(tActual.dirMem)->estado)
            {
                case C_I: return true;
                case C_E: return false;
                case C_M: return false;
                case C_SC: if(tActual.tIns == IESCRITURA) return true;
                           else return false;
                case C_SM: if(tActual.tIns == IESCRITURA) return true;
                           else return false;
            }
        break;
    }
    return false;
}

ENTRADA *Cache::getEntrada(long dirMem)
{
    int idBloque = dirMemToBloq(dirMem,bloqMem,palBloque);
    ENTRADA* ent
=obtenerBloque(datos[bloqueToCjto(idBloque)],bloqueToEtiqueta(idBloque
));
    return ent;
}

bool Cache::BuscarDirMem(long dirMem)
{
    return BuscarBloque(dirMemToBloq(dirMem,bloqMem,palBloque));
}

void Cache::postEscrituras()
{
    int i,j;
    for(i = 0;i< datos.size();i++)
    {
        for(j = 0;j< datos[i].size();j++)
        {
            if(datos[i][j].estado == C_M)
            {
                Bus::getInstancia()->numBloqTrans++;
                Bus::getInstancia()->numTransBus++;
                Bus::getInstancia()->numTranEstado++;
                switch(pCoherencia)
                {
                    case P_MSI: Bus::getInstancia()-
>cambios_estado[datos[i][j].estado][C_S]++;
                    case P_MESI: Bus::getInstancia()-
>cambios_estado[datos[i][j].estado][C_E]++;
                    case P_DRAGON: Bus::getInstancia()-
>cambios_estado[datos[i][j].estado][C_E]++;
                }
            }
        }
    }
}

```

```

    }
    if (datos[i][j].estado == C_SM)
    {
        Bus::getInstancia()->numBloqTrans++;
        Bus::getInstancia()->numTransBus++;
        Bus::getInstancia()->numTranEstado++;
        switch (pCoherencia)
        {
            case P_DRAGON: Bus::getInstancia()-
>cambios_estado[datos[i][j].estado][C_SC]++;
        }
    }
}
}
}

```

ErrorConfig.h

```

#pragma once

#include <queue>
#include "afxwin.h"
#include "afxcmn.h"

using namespace std;

class ErrorConfig
{
public:

    ErrorConfig(void)
    {

    }

    void add(CString error)
    {
        problemas.push(error);
    }

    void what()
    {
        while (!problemas.empty())
        {
            AfxMessageBox(problemas.front());
            problemas.pop();
        }
    }

    void check()
    {
        if (!problemas.empty())
            throw this;
    }

    ~ErrorConfig(void)
    {

    }

    queue<CString> problemas;

```

```
};
```

Procesador.h

```
#pragma once

#include <queue>
#include "constantes.h"

using namespace std ;

typedef queue<LINEATRAZA> miCola;

class Procesador
{
public:
    Procesador(CString ficheroTraza);
    LINEATRAZA getLinea();
private:
    miCola lineas;
};
```

Procesador.cpp

```
#include "StdAfx.h"
#include "Procesador.h"

Procesador::Procesador(CString ficheroTraza)
{
    CStdioFile file;
    CString str;
    int i = 0;
    if(!file.Open(ficheroTraza, CFile::modeRead))
    {
        AfxMessageBox(_T("Error: No se ha conseguido abrir un
archivo"));
        throw ("Archivo no encontrado");
    }
    LINEATRAZA aux;
    while(file.ReadString(str))
    {
        aux.tIns= ((TINSTRUCCION)_wtoi(str.Left(1)));
        aux.dirMem=wcstoul(str.Right(6),0,16);
        lineas.push(aux);
    }
    file.Close();
}

LINEATRAZA Procesador::getLinea()
{
    LINEATRAZA aux;
    aux.tIns = IVACIO;
    if(!lineas.empty())
    {
        aux = lineas.front();
        lineas.pop();
    }
    return aux;
}
```

SMPCache.h

```
// SMPCache.h: archivo de encabezado principal para la aplicación
SMPCache
//
#pragma once

#ifndef __AFXWIN_H__
    #error "incluir 'stdafx.h' antes de incluir este archivo para
PCH"
#endif

#include "resource.h"           // Símbolos principales

// CSMPCacheApp:
// Consulte la sección SMPCache.cpp para obtener información sobre la
implementación de esta clase
//

class CSMPCacheApp : public CWinApp
{
public:
    CSMPCacheApp();

// Reemplazos
public:
    virtual BOOL InitInstance();
    virtual BOOL OnIdle(LONG lCount);

// Implementación
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

extern CSMPCacheApp theApp;
```

SMPCache.cpp

```
// SMPCache.cpp : define los comportamientos de las clases para la
aplicación.
//

#include "stdafx.h"
#include "SMPCache.h"
#include "MainFrm.h"

#include "SMPCacheDoc.h"
#include "SMPCacheView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CSMPCacheApp

BEGIN_MESSAGE_MAP(CSMPCacheApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, &CSMPCacheApp::OnAppAbout)
    // Comandos de documento estándar basados en archivo
    ON_COMMAND(ID_FILE_NEW, &CWinApp::OnFileNew)
```

```

        ON_COMMAND(ID_FILE_OPEN, &CWinApp::OnFileOpen)
END_MESSAGE_MAP()

// Construcción de CSMPCacheApp

CSMPCacheApp::CSMPCacheApp()
{
    // TODO: agregar aquí el código de construcción,
    // Colocar toda la inicialización importante en InitInstance
}

// El único objeto CSMPCacheApp

CSMPCacheApp theApp;

// Inicialización de CSMPCacheApp

BOOL CSMPCacheApp::InitInstance()
{
    CWinApp::InitInstance();

    // Inicialización estándar
    // Si no utiliza estas características y desea reducir el tamaño
    // del archivo ejecutable final, debe quitar
    // las rutinas de inicialización específicas que no necesite
    // Cambie la clave del Registro en la que se almacena la
configuración
    SetRegistryKey(_T("Practica SMPCache. Arquitectura de
Computadores. Manuel y Mario."));
    LoadStdProfileSettings(4); // Cargar opciones de archivo INI
estándar (incluidas las de la lista MRU)
    // Registrar las plantillas de documento de la aplicación. Las
plantillas de documento
    // sirven como conexión entre documentos, ventanas de marco y
vistas
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CSMPCacheDoc),
        RUNTIME_CLASS(CMainFrame),           // Ventana de marco SDI
principal
        RUNTIME_CLASS(CSMPCacheView));
    if (!pDocTemplate)
        return FALSE;
    AddDocTemplate(pDocTemplate);

    // Habilitar apertura de ejecución DDE
    EnableShellOpen();
    RegisterShellFileTypes(TRUE);

    // Analizar línea de comandos para comandos Shell estándar, DDE,
Archivo Abrir
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Enviar comandos especificados en la línea de comandos.
    Devolverá FALSE si

```

```

        // la aplicación se inició con los modificadores /RegServer,
/Registrar, /Unregserver o /Unregister.
        if (!ProcessShellCommand(cmdInfo))
            return FALSE;

        // Se ha inicializado la única ventana; mostrarla y actualizarla
m_pMainWnd->ShowWindow(SW_SHOWMAXIMIZED);
m_pMainWnd->UpdateWindow();
        // Llamar a DragAcceptFiles sólo si existe un sufijo
        // En una aplicación SDI, esto debe ocurrir después de
ProcessShellCommand
        // Habilitar apertura de arrastrar y colocar
m_pMainWnd->DragAcceptFiles();
        return TRUE;
    }

// Cuadro de diálogo CAboutDlg utilizado para el comando Acerca de

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Datos del cuadro de diálogo
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);    //
Compatibilidad con DDX/DDV

// Implementación
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// Comando de la aplicación para ejecutar el cuadro de diálogo
void CSMPCacheApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

BOOL CSMPCacheApp::OnIdle(LONG lCount)
{
    CSMPCacheView* miVista = (CSMPCacheView *) m_pMainWnd-
>GetWindow(GW_CHILD);

```



```

        if(miVista->enEjecucion && !miVista->enPausa)
        {
            if(!miVista->realizarIteracion())delete Bus::getInstancia();
        }
        return TRUE;
    }
}

```

SMPCacheView.h

```

// SMPCacheView.h: interfaz de la clase CSMPCacheView
//

```

```

#pragma once
#include "afxwin.h"
#include "afxcmn.h"
#include "Procesador.h"
#include "Cache.h"
#include "Bus.h"
#include "constantes.h"

class CSMPCacheView : public CFormView
{
public:
    void actualizarControles();
protected: // Crear sólo a partir de serialización
    CSMPCacheView();
    DECLARE_DYNCREATE(CSMPCacheView)

public:
    enum{ IDD = IDD_Principal };

    // Atributos
public:
    CSMPCacheDoc* GetDocument() const;
    // variable de control para las imagenes de los procesadores
    CStatic proc[8];
    // variable de control para las flechas a la izquierda de los
    procesadores
    CStatic flechasproc[8];
    // variable de control para las flechas a la izquierda de las
    caches
    CStatic flechasCI[8];
    // variable de control para las flechas a la derecha de las
    caches
    CStatic flechasCD[8];
    //variable de control de la flecha de la memoria
    CStatic flechaMem;
    // variable de control para el texto a la izquierda de los
    procesadores
    CStatic textoproc[8];
    // variable de control para el texto a la izquierda de los cache
    CStatic textoCI[8];
    // variable de control para el texto a la derecha de los cache
    CStatic textoCD[8];

    //booleanos para saber si la flecha apunta hacia arriba
    bool fParriba[8];
    bool fCIarriba[8];
    bool fCDarriba[8];

```

```

//variables para mostrar los cambios de estado
CStatic cambios[5][5];
CStatic labelCambios[5][2];

//opciones de configuracion
int procn;
PROTOCOLO protocolo;
ARBITRACION arbitracion;
int anchoPalabra;
int palabrasBloque;
int bloqMemoria;
int bloqCache;
FCORRESPONDENCIA fcorrespondencia;
int nConjuntos;
AREMPLAZAMIENTO aremplazamiento;
int nCache;
int pescritura;
CString fTrazas[8];

// Operaciones
public:
    void ocultarInteraccionesCache();
    void mostrarProc();
    void ocultarProc();
    void mostrarFCache(int pos);
    void ocultarFCache(int pos);
    void actualizarVista();
    void cambiarFlechaProc(int pos);
    void cambiarTextoProc(int pos, CString texto);
    void cambiarFlechaCI(int pos);
    void cambiarTextoCI(int pos, CString texto);
    void cambiarFlechaCD(int pos);
    void iniciarSimulacion();

// Reemplazos
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void DoDataExchange(CDataExchange* pDX);    //
Compatibilidad con DDX/DDV
protected:
    virtual void OnInitialUpdate(); // Se llama la primera vez
    después de la construcción

// Implementación
public:
    virtual ~CSMPCacheView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Funciones de asignación de mensajes generadas
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnOpcionesConfigurarmultiprocesador();
    afx_msg void OnOpcionesMemoria();
    afx_msg void OnOpcionesCaches();
    afx_msg void OnFileOpen();

```

```

afx_msg void OnBnClickedBiniciar();
//estao de la simulacion
bool enEjecucion;
// boton iniciar
CButton bIniciar;
//estado del boton de pausa
bool enPausa;

// boton de pausa o continuar
CButton bPausa;
// boton de ejec completa
CButton bCompleta;
// boton para salir
CButton bSalir;
// pasos de ejecucion
int steps;

afx_msg void OnBnClickedBpausa();
afx_msg void OnBnClickedBCompleta();
afx_msg void OnBnClickedBsalir();
bool realizarIteracion();
afx_msg void OnEnChangeSteps();
// cuadro de texto de steps
CEdit TSteps;
CStatic TotalAccesos;
CStatic TotalInst;
// numero de datos leidos
CStatic TotalLeidas;
// numero de datos escritos
CStatic TotalEscritos;
// variable de control de accesos
CStatic Caccesos;
// variable de control de instrucciones
CStatic Cinstrucciones;
CStatic Cleidos;
CStatic Cescritos;
// Control para las transacciones del bus
CStatic CtransBus;
// Control de los bloques transferidos
CStatic Cbloqtrans;
// Control para las transacciones de estado
CStatic Ctranestado;
// Control del los aciertos en cache
CStatic Caciertos;
// Control del porcentaje de aciertos en cache
CStatic CPorAciertos;
// Control de los fallos en cache
CStatic Cfallos;
// Control del porcentaje de fallos en cache
CStatic CPorFallos;
};

#ifdef _DEBUG // Versión de depuración en SMPCacheView.cpp
inline CSMPCacheDoc* CSMPCacheView::GetDocument() const
{ return reinterpret_cast<CSMPCacheDoc*>(m_pDocument); }
#endif

```

SMPCacheView.cpp

```
// SMPCacheView.cpp: implementación de la clase CSMPCacheView
//

#include "stdafx.h"
#include "SMPCache.h"

#include "SMPCacheDoc.h"
#include "SMPCacheView.h"

#include "ProcConf.h"
#include "MemConf.h"
#include "CacheConf.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CSMPCacheView

IMPLEMENT_DYNCREATE(CSMPCacheView, CFormView)

BEGIN_MESSAGE_MAP(CSMPCacheView, CFormView)
    ON_COMMAND(ID_FILE_OPEN, &CSMPCacheView::OnFileOpen)
    ON_COMMAND(ID_OPCIONES_CONFIGURARMULTIPROCESADOR,
&CSMPCacheView::OnOpcionesConfigurarmultiprocesador)
    ON_COMMAND(ID_OPCIONES_MEMORIA,
&CSMPCacheView::OnOpcionesMemoria)
    ON_COMMAND(ID_OPCIONES_CACHES, &CSMPCacheView::OnOpcionesCaches)
    ON_BN_CLICKED(IDC_BIniciar, &CSMPCacheView::OnBnClickedBiniciar)
    ON_BN_CLICKED(IDC_BPausa, &CSMPCacheView::OnBnClickedBpausa)
    ON_BN_CLICKED(IDC_BDetener, &CSMPCacheView::OnBnClickedBcompleta)
    ON_BN_CLICKED(IDC_BSalir, &CSMPCacheView::OnBnClickedBsalir)
    ON_EN_CHANGE(IDC_STEPS, &CSMPCacheView::OnEnChangeSteps)
END_MESSAGE_MAP()

// Construcción o destrucción de CSMPCacheView

CSMPCacheView::CSMPCacheView()
    : CFormView(CSMPCacheView::IDD)
    , steps(1)
{
    enEjecucion = false;
    enPausa = false;
    Bus::getInstancia()->fallos = 0;
    Bus::getInstancia()->aciertos = 0;
    Bus::getInstancia()->numTranEstado = 0;
    Bus::getInstancia()->numBloqTrans = 0;
    Bus::getInstancia()->numTransBus = 0;
    // TODO: agregar aquí el código de construcción
    procn = 8;
    for(int i = 0 ; i<8; i++)
    {
        fParriba[i] = false;
        fCIarriba[i] = false;
        fCDarriba[i] = false;
    }
}

CSMPCacheView::~CSMPCacheView()
```

```

{
}

void CSMPCacheView::iniciarSimulacion()
{
    for(int i =0 ;i <procn;i++)
    {
        Bus::getInstancia()-
>indicarProtocoloArbitracion(arbitracion);
        if(!fTrazas[i].IsEmpty())
        {
            new
Cache(i,bloqCache,fcorrespondencia,nConjuntos,areemplazamiento,protocol
o,bloqMemoria,palabrasBloque,fTrazas[i]);
        }
    }
    char cadena[10];
    _itoa_s(1,cadena,10);
    TSteps.SetWindowTextW((CString)cadena);
}

void CSMPCacheView::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    DDX_Control(pDX, IMG_Proc1, proc[0]);
    DDX_Control(pDX, IMG_Proc2, proc[1]);
    DDX_Control(pDX, IMG_Proc3, proc[2]);
    DDX_Control(pDX, IMG_Proc4, proc[3]);
    DDX_Control(pDX, IMG_Proc5, proc[4]);
    DDX_Control(pDX, IMG_Proc6, proc[5]);
    DDX_Control(pDX, IMG_Proc7, proc[6]);
    DDX_Control(pDX, IMG_Proc8, proc[7]);
    DDX_Control(pDX, FLE_Proc1, flechasCI[0]);
    DDX_Control(pDX, FLE_Proc2, flechasCI[1]);
    DDX_Control(pDX, FLE_Proc3, flechasCI[2]);
    DDX_Control(pDX, FLE_Proc4, flechasCI[3]);
    DDX_Control(pDX, FLE_Proc5, flechasCI[4]);
    DDX_Control(pDX, FLE_Proc6, flechasCI[5]);
    DDX_Control(pDX, FLE_Proc7, flechasCI[6]);
    DDX_Control(pDX, FLE_Proc8, flechasCI[7]);
    DDX_Control(pDX, FLE_Proc9, flechasproc[0]);
    DDX_Control(pDX, FLE_Proc10, flechasproc[1]);
    DDX_Control(pDX, FLE_Proc11, flechasproc[2]);
    DDX_Control(pDX, FLE_Proc12, flechasproc[3]);
    DDX_Control(pDX, FLE_Proc13, flechasproc[4]);
    DDX_Control(pDX, FLE_Proc14, flechasproc[5]);
    DDX_Control(pDX, FLE_Proc15, flechasproc[6]);
    DDX_Control(pDX, FLE_Proc16, flechasproc[7]);
    DDX_Control(pDX, FLE_Mem, flechaMem);
    DDX_Control(pDX, FLE_Proc17, flechasCD[0]);
    DDX_Control(pDX, FLE_Proc18, flechasCD[1]);
    DDX_Control(pDX, FLE_Proc19, flechasCD[2]);
    DDX_Control(pDX, FLE_Proc20, flechasCD[3]);
    DDX_Control(pDX, FLE_Proc21, flechasCD[4]);
    DDX_Control(pDX, FLE_Proc22, flechasCD[5]);
    DDX_Control(pDX, FLE_Proc23, flechasCD[6]);
    DDX_Control(pDX, FLE_Proc24, flechasCD[7]);
    DDX_Control(pDX, TXT_Proc1, textoproc[0]);
    DDX_Control(pDX, TXT_Proc2, textoproc[1]);
    DDX_Control(pDX, TXT_Proc3, textoproc[2]);
    DDX_Control(pDX, TXT_Proc4, textoproc[3]);
}

```

```

DDX_Control(pDX, TXT_Proc5, textoproc[4]);
DDX_Control(pDX, TXT_Proc6, textoproc[5]);
DDX_Control(pDX, TXT_Proc7, textoproc[6]);
DDX_Control(pDX, TXT_Proc8, textoproc[7]);
DDX_Control(pDX, TXT_Proc9, textoCI[0]);
DDX_Control(pDX, TXT_Proc10, textoCI[1]);
DDX_Control(pDX, TXT_Proc11, textoCI[2]);
DDX_Control(pDX, TXT_Proc12, textoCI[3]);
DDX_Control(pDX, TXT_Proc13, textoCI[4]);
DDX_Control(pDX, TXT_Proc14, textoCI[5]);
DDX_Control(pDX, TXT_Proc15, textoCI[6]);
DDX_Control(pDX, TXT_Proc16, textoCI[7]);
DDX_Control(pDX, TXT_Proc17, textoCD[0]);
DDX_Control(pDX, TXT_Proc18, textoCD[1]);
DDX_Control(pDX, TXT_Proc19, textoCD[2]);
DDX_Control(pDX, TXT_Proc20, textoCD[3]);
DDX_Control(pDX, TXT_Proc21, textoCD[4]);
DDX_Control(pDX, TXT_Proc22, textoCD[5]);
DDX_Control(pDX, TXT_Proc23, textoCD[6]);
DDX_Control(pDX, TXT_Proc24, textoCD[7]);
DDX_Control(pDX, IDC_BIniciar, bIniciar);
DDX_Control(pDX, IDC_BPausa, bPausa);
DDX_Control(pDX, IDC_BDetener, bCompleta);
DDX_Control(pDX, IDC_BSalir, bSalir);
DDX_Text(pDX, IDC_LTBus, Bus::getInstancia()->numTransBus);
DDX_Text(pDX, IDC_LTBloques, Bus::getInstancia()->numBloqTrans);
DDX_Text(pDX, IDC_LTestado, Bus::getInstancia()->numTranEstado);
DDX_Control(pDX, IDC_STEPS, TSteps);
DDX_Control(pDX, IDC_SAMemoria2, TotalAccesos);
DDX_Control(pDX, IDC_SInstrucciones2, TotalInst);
DDX_Control(pDX, IDC_SDLeidos2, TotalLeidas);
DDX_Control(pDX, IDC_SDEscritos2, TotalEscritos);
DDX_Control(pDX, IDC_SAMemoria, Caccesos);
DDX_Control(pDX, IDC_SInstrucciones, Cinstrucciones);
DDX_Control(pDX, IDC_SDLeidos, Cleidos);
DDX_Control(pDX, IDC_SDEscritos, Cescritos);
DDX_Control(pDX, IDC_LTBus, CtransBus);
DDX_Control(pDX, IDC_LTBloques, Cblogtrans);
DDX_Control(pDX, IDC_LTestado, Ctranestado);
DDX_Control(pDX, IDC_SNAciertos, Caciertos);
DDX_Control(pDX, IDC_SPACiertos, CPorAciertos);
DDX_Control(pDX, IDC_SNFallos, Cfallos);
DDX_Control(pDX, IDC_SPFallos, CPorFallos);
DDX_Control(pDX, IDC_Estados0, cambios[0][0]);
DDX_Control(pDX, IDC_Estados1, cambios[0][1]);
DDX_Control(pDX, IDC_Estados2, cambios[0][2]);
DDX_Control(pDX, IDC_Estados3, cambios[0][3]);
DDX_Control(pDX, IDC_Estados16, cambios[0][4]);
DDX_Control(pDX, IDC_Estados4, cambios[1][0]);
DDX_Control(pDX, IDC_Estados5, cambios[1][1]);
DDX_Control(pDX, IDC_Estados6, cambios[1][2]);
DDX_Control(pDX, IDC_Estados7, cambios[1][3]);
DDX_Control(pDX, IDC_Estados17, cambios[1][4]);
DDX_Control(pDX, IDC_Estados8, cambios[2][0]);
DDX_Control(pDX, IDC_Estados9, cambios[2][1]);
DDX_Control(pDX, IDC_Estados10, cambios[2][2]);
DDX_Control(pDX, IDC_Estados11, cambios[2][3]);
DDX_Control(pDX, IDC_Estados18, cambios[2][4]);
DDX_Control(pDX, IDC_Estados12, cambios[3][0]);
DDX_Control(pDX, IDC_Estados13, cambios[3][1]);
DDX_Control(pDX, IDC_Estados14, cambios[3][2]);

```

```

        DDX_Control(pDX, IDC_Estados15, cambios[3][3]);
        DDX_Control(pDX, IDC_Estados19, cambios[3][4]);
        DDX_Control(pDX, IDC_Estados20, cambios[4][0]);
        DDX_Control(pDX, IDC_Estados21, cambios[4][1]);
        DDX_Control(pDX, IDC_Estados22, cambios[4][2]);
        DDX_Control(pDX, IDC_Estados23, cambios[4][3]);
        DDX_Control(pDX, IDC_Estados24, cambios[4][4]);
        DDX_Control(pDX, IDC_CEstadoL1A, labelCambios[0][0]);
        DDX_Control(pDX, IDC_CEstadoL2A, labelCambios[1][0]);
        DDX_Control(pDX, IDC_CEstadoL3A, labelCambios[2][0]);
        DDX_Control(pDX, IDC_CEstadoL4A, labelCambios[3][0]);
        DDX_Control(pDX, IDC_CEstadoL5A, labelCambios[4][0]);
        DDX_Control(pDX, IDC_CEstadoL1B, labelCambios[0][1]);
        DDX_Control(pDX, IDC_CEstadoL2B, labelCambios[1][1]);
        DDX_Control(pDX, IDC_CEstadoL3B, labelCambios[2][1]);
        DDX_Control(pDX, IDC_CEstadoL4B, labelCambios[3][1]);
        DDX_Control(pDX, IDC_CEstadoL5B, labelCambios[4][1]);
    }

    BOOL CSMPCCacheView::PreCreateWindow(CREATESTRUCT& cs)
    {
        // TODO: modificar aquí la clase Window o los estilos cambiando
        // CREATESTRUCT cs
        return CFormView::PreCreateWindow(cs);
    }

    void CSMPCCacheView::OnInitialUpdate()
    {
        CFormView::OnInitialUpdate();
        GetParentFrame()->RecalcLayout();
        ResizeParentToFit();
    }

    // Diagnósticos de CSMPCCacheView

#ifdef _DEBUG
    void CSMPCCacheView::AssertValid() const
    {
        CView::AssertValid();
    }

    void CSMPCCacheView::Dump(CDumpContext& dc) const
    {
        CView::Dump(dc);
    }

    CSMPCCacheDoc* CSMPCCacheView::GetDocument() const // La versión de no
    depuración es en línea
    {
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CSMPCCacheDoc)));
        return (CSMPCCacheDoc*)m_pDocument;
    }
#endif // _DEBUG

    // Controladores de mensaje de CSMPCCacheView

    void CSMPCCacheView::OnOpcionesConfigurarmultiprocesador()
    {
        ProcConf formulario;
        formulario.DoModal();
    }

```

```

}

void CSMPCacheView::OnOpcionesMemoria()
{
    MemConf formulario;
    formulario.DoModal();
}

void CSMPCacheView::OnOpcionesCaches()
{
    CacheConf formulario;
    formulario.DoModal();
}

void CSMPCacheView::mostrarFCache(int pos)
{
    flechasCD[pos].ShowWindow(true);
    flechasCI[pos].ShowWindow(true);
    textoCI[pos].ShowWindow(true);
    textoCD[pos].ShowWindow(true);
}

void CSMPCacheView::ocultarFCache(int pos)
{
    flechasCD[pos].ShowWindow(false);
    flechasCI[pos].ShowWindow(false);
    textoCI[pos].ShowWindow(false);
    textoCD[pos].ShowWindow(false);
}

void CSMPCacheView::OnFileOpen()
{
    // TODO: Agregue aquí su código de controlador de comandos
    CString sFiltro;
    sFiltro="Ficheros configuracion (*.cfg)|*.cfg|Fichers traza
(*.prg)|*.prg||";

    CFileDialog Dialogoabrir(TRUE,NULL,NULL,4|2,sFiltro);

    if(Dialogoabrir.DoModal() != IDOK)
        return;

    ErrorConfig varControl;

    CStdioFile file;
    CString str;
    int i = 0;
    file.Open(Dialogoabrir.GetPathName(), CFile::modeRead);
    file.ReadString(str);
    file.ReadString(str);
    procn = _wtoi(str);
    if(procn < 1 || procn > 8)
        varControl.add(_T("Error en la configuracion del numero de
procesadores!"));
    file.ReadString(str);
    file.ReadString(str);
    protocolo = int2Protocolo(_wtoi(str),varControl);
    file.ReadString(str);
    file.ReadString(str);
    arbitracion = int2Arbitracion(_wtoi(str),varControl);
    file.ReadString(str);

```



```

        file.ReadString(str);
        anchoPalabra = _wtoi(str);
        if(anchoPalabra != 8 && anchoPalabra != 16 && anchoPalabra != 32
&& anchoPalabra != 64 )
            varControl.add(_T("Error en la configuracion del ancho de
palabra!"));
        file.ReadString(str);
        file.ReadString(str);
        palabrasBloque = _wtoi(str);
        if(palabrasBloque < 1 || ((int) log2(palabrasBloque)) !=
log2(palabrasBloque) || log2(palabrasBloque) > 10 )
            varControl.add(_T("Error en la configuracion de palabras
por bloque!"));
        file.ReadString(str);
        file.ReadString(str);
        bloqMemoria = _wtoi(str);
        if(bloqMemoria < 1 || ((int) log2(bloqMemoria)) !=
log2(bloqMemoria) || log2(bloqMemoria) > 15 )
            varControl.add(_T("Error en la configuracion del numero de
bloques en memoria!"));
        file.ReadString(str);
        file.ReadString(str);
        bloqCache = _wtoi(str);
        if(bloqCache < 1 || ((int) log2(bloqCache)) != log2(bloqCache)
|| log2(bloqCache) > 9 )
            varControl.add(_T("Error en la configuracion del numero de
bloques en cache!"));
        file.ReadString(str);
        file.ReadString(str);
        fcorrespondencia = int2Correspondencia(_wtoi(str),varControl);
        file.ReadString(str);
        file.ReadString(str);
        nConjuntos = _wtoi(str);
        if(nConjuntos < 0 || ((int) log2(nConjuntos)) != log2(nConjuntos)
|| log2(nConjuntos) > 9 )
            varControl.add(_T("Error en la configuracion del numero de
conjuntos en cache!"));
        file.ReadString(str);
        file.ReadString(str);
        aremplazamiento = int2AREmplazamiento(_wtoi(str),varControl);
        file.ReadString(str);
        file.ReadString(str);
        nCache = _wtoi(str);
        if(nCache != 1)
            varControl.add(_T("Error en la configuracion del numero de
niveles de cache!"));
        file.ReadString(str);
        file.ReadString(str);
        pescritura = _wtoi(str);
        if(pescritura != 2)
            varControl.add(_T("Error en la configuracion del protocolo
de escritura!"));
        file.ReadString(str);
        //ficheros de traza
        for(int i = 0; i < 8;i++)
        {
            file.ReadString(str);
            fTrazas[i] = str.Right(str.GetLength()-3);
        }

        //obtencion de las estadisticas totales

```

```

int TAccesos = 0, TInst = 0, TLec = 0, TEsc = 0;
CStdioFile file2;
CString str2;
for(int i = 0; i<procn;i++)
{
    if(!fTrazas[i].IsEmpty())
    {
        if(!file2.Open(fTrazas[i], CFile::modeRead))
        {
            varControl.add(_T("Error: archivo de
traza inexistente"));
        }
        while(file2.ReadString(str2))
        {
            TAccesos++;
            int i = 0;

            switch((TINSTRUCCION)_wtoi(str2.Left(1)))
            {
                case 0: TInst++;break;
                case 2: TLec++;break;
                case 3: TEsc++;break;
            }
        }
        file2.Close();
    }
}

bIniciar.EnableWindow(true);

char cadena[20];
_itoa_s(TInst,cadena,10);
TotalInst.SetWindowTextW((CString)cadena);
_itoa_s(TAccesos,cadena,10);
TotalAccesos.SetWindowTextW((CString)cadena);
_itoa_s(TLec,cadena,10);
TotalLeidas.SetWindowTextW((CString)cadena);
_itoa_s(TEsc,cadena,10);
TotalEscritos.SetWindowTextW((CString)cadena);

bool alguna = false;
for(int i =0; i< 8; i++)
{
    if(!fTrazas[i].IsEmpty()) alguna = true;
}
if(!alguna)
    varControl.add(_T("Error en la configuracion de los
ficheros de traza, no se ha indicado ninguno!"));

if(fcorrespondencia == F_DIRECTA && (nConjuntos != 0 ||
areemplazamiento != R_NO))
    varControl.add(_T("Error, falta de coherencia en los
datos"));
if(fcorrespondencia == F_ACONJUNTOS && nConjuntos == 0)
    varControl.add(_T("Error, falta de coherencia en los
datos"));
if(fcorrespondencia == F_TASOCIATIVA && nConjuntos != 0)
    varControl.add(_T("Error, falta de coherencia en los
datos"));

```

```

        try{
            varControl.check();
            actualizarVista();
        }catch (ErrorConfig *ex)
        {
            ex->what();
        }
    }

void setInt(CStatic &control, int &i)
{
    char cadena[20];
    _itoa_s(i, cadena, 10);
    control.SetWindowTextW((CString) cadena);
}

void CSMPCCacheView::actualizarControles()
{
    char cadena[20];
    _itoa_s(Bus::getInstancia()->memAccesos, cadena, 10);
    Caccesos.SetWindowTextW((CString) cadena);
    _itoa_s(Bus::getInstancia()->instLeidas, cadena, 10);
    Cinstrucciones.SetWindowTextW((CString) cadena);
    _itoa_s(Bus::getInstancia()->lecturas, cadena, 10);
    Cleidos.SetWindowTextW((CString) cadena);
    _itoa_s(Bus::getInstancia()->escrituras, cadena, 10);
    Cescritos.SetWindowTextW((CString) cadena);
    _itoa_s(Bus::getInstancia()->numTransBus, cadena, 10);
    CtransBus.SetWindowTextW((CString) cadena);
    sprintf_s(cadena, "%f", Bus::getInstancia()->numBloqTrans);
    Cbloqtrans.SetWindowTextW((CString) cadena);
    _itoa_s(Bus::getInstancia()->numTranEstado, cadena, 10);
    Ctranestado.SetWindowTextW((CString) cadena);
    _itoa_s(Bus::getInstancia()->aciertos, cadena, 10);
    Caciertos.SetWindowTextW((CString) cadena);
    _itoa_s(Bus::getInstancia()->fallos, cadena, 10);
    Cfallos.SetWindowTextW((CString) cadena);
    _itoa_s(Bus::getInstancia()->aciertos*100/(Bus::getInstancia()-
>fallos+Bus::getInstancia()->aciertos), cadena, 10);
    CPorAciertos.SetWindowTextW((CString) cadena);
    _itoa_s(Bus::getInstancia()->fallos*100/(Bus::getInstancia()-
>fallos+Bus::getInstancia()->aciertos), cadena, 10);
    CPorFallos.SetWindowTextW((CString) cadena);
    if(!enPausa&&enEjecucion) return;
    int o = 0;
    for(int i = 0; i< procn; i++)
    {
        if(!fTrazas[i].IsEmpty())
        {
            CString ccadena(Bus::getInstancia()->caches[o++]-
>textoCache);
            cambiarTextoProc(i, ccadena);
        }
    }
    ocultarInteraccionesCache();
    if(Bus::getInstancia()->flechaMensajeCache.id != -1)
    {
        flechasCI[Bus::getInstancia()-
>flechaMensajeCache.id].ShowWindow(true);
        flechasCD[Bus::getInstancia()-
>flechaMensajeCache.id].ShowWindow(true);
    }
}

```

```

        flechasCD[Bus::getInstancia() -
>flechaMensajeCache.id].SetBitmap(LoadBitmap(theApp.m_hInstance,MAKEINTRESOURCE(IDB_FArriba)));
        textoCI[Bus::getInstancia() -
>flechaMensajeCache.id].ShowWindow(true);
        textoCD[Bus::getInstancia() -
>flechaMensajeCache.id].ShowWindow(true);
        switch(Bus::getInstancia()->flechaMensajeCache.mensaje.tipo)
        {
            case M_BUSRD: strcpy_s(cadena,"BUSRD");break;
            case M_BUSRDX: strcpy_s(cadena,"BUSRDX");break;
            case M_BUSUPD: strcpy_s(cadena,"BUSUPD");break;
            case M_BUSRDUPT: strcpy_s(cadena,"RD/UPD");break;
            default:  strcpy_s(cadena,"ERROR");break;
        }
        textoCI[Bus::getInstancia() -
>flechaMensajeCache.id].SetWindowTextW((CString)cadena);
        _itoa_s(dirMemToBloq(Bus::getInstancia() -
>flechaMensajeCache.mensaje.dirmem,palabrasBloque,bloqMemoria),cadena,
10);
        textoCD[Bus::getInstancia() -
>flechaMensajeCache.id].SetWindowTextW((CString)cadena);
        if(Bus::getInstancia()->origDat == -1)
        {
            flechaMem.ShowWindow(true);
        }
        else
        {
            flechasCI[Bus::getInstancia() -
>origDat].ShowWindow(true);
        }
    }
    //actualizacion de los estados
    switch(protocolo)
    {
        case P_MSI:
            setInt(cambios[0][0],Bus::getInstancia() -
>cambios_estado[C_M][C_M]);
            setInt(cambios[0][1],Bus::getInstancia() -
>cambios_estado[C_M][C_S]);
            setInt(cambios[0][2],Bus::getInstancia() -
>cambios_estado[C_M][C_I]);
            setInt(cambios[1][0],Bus::getInstancia() -
>cambios_estado[C_S][C_M]);
            setInt(cambios[1][1],Bus::getInstancia() -
>cambios_estado[C_S][C_S]);
            setInt(cambios[1][2],Bus::getInstancia() -
>cambios_estado[C_S][C_I]);
            setInt(cambios[2][0],Bus::getInstancia() -
>cambios_estado[C_I][C_M]);
            setInt(cambios[2][1],Bus::getInstancia() -
>cambios_estado[C_I][C_S]);
            setInt(cambios[2][2],Bus::getInstancia() -
>cambios_estado[C_I][C_I]);
            break;
        case P_MESI:
            setInt(cambios[0][0],Bus::getInstancia() -
>cambios_estado[C_M][C_M]);
            setInt(cambios[0][1],Bus::getInstancia() -
>cambios_estado[C_M][C_E]);

```

```

        setInt(cambios[0][2], Bus::getInstancia() -
>cambios_estado[C_M][C_S]);
        setInt(cambios[0][3], Bus::getInstancia() -
>cambios_estado[C_M][C_I]);
        setInt(cambios[1][0], Bus::getInstancia() -
>cambios_estado[C_E][C_M]);
        setInt(cambios[1][1], Bus::getInstancia() -
>cambios_estado[C_E][C_E]);
        setInt(cambios[1][2], Bus::getInstancia() -
>cambios_estado[C_E][C_S]);
        setInt(cambios[1][3], Bus::getInstancia() -
>cambios_estado[C_E][C_I]);
        setInt(cambios[2][0], Bus::getInstancia() -
>cambios_estado[C_S][C_M]);
        setInt(cambios[2][1], Bus::getInstancia() -
>cambios_estado[C_S][C_E]);
        setInt(cambios[2][2], Bus::getInstancia() -
>cambios_estado[C_S][C_S]);
        setInt(cambios[2][3], Bus::getInstancia() -
>cambios_estado[C_S][C_I]);
        setInt(cambios[3][0], Bus::getInstancia() -
>cambios_estado[C_I][C_M]);
        setInt(cambios[3][1], Bus::getInstancia() -
>cambios_estado[C_I][C_E]);
        setInt(cambios[3][2], Bus::getInstancia() -
>cambios_estado[C_I][C_S]);
        setInt(cambios[3][3], Bus::getInstancia() -
>cambios_estado[C_I][C_I]);
        break;
    case P_DRAGON:
        setInt(cambios[0][0], Bus::getInstancia() -
>cambios_estado[C_E][C_E]);
        setInt(cambios[0][1], Bus::getInstancia() -
>cambios_estado[C_E][C_SC]);
        setInt(cambios[0][2], Bus::getInstancia() -
>cambios_estado[C_E][C_SM]);
        setInt(cambios[0][3], Bus::getInstancia() -
>cambios_estado[C_E][C_M]);
        setInt(cambios[0][4], Bus::getInstancia() -
>cambios_estado[C_E][C_I]);
        setInt(cambios[1][0], Bus::getInstancia() -
>cambios_estado[C_SC][C_E]);
        setInt(cambios[1][1], Bus::getInstancia() -
>cambios_estado[C_SC][C_SC]);
        setInt(cambios[1][2], Bus::getInstancia() -
>cambios_estado[C_SC][C_SM]);
        setInt(cambios[1][3], Bus::getInstancia() -
>cambios_estado[C_SC][C_M]);
        setInt(cambios[1][4], Bus::getInstancia() -
>cambios_estado[C_SC][C_I]);
        setInt(cambios[2][0], Bus::getInstancia() -
>cambios_estado[C_SM][C_E]);
        setInt(cambios[2][1], Bus::getInstancia() -
>cambios_estado[C_SM][C_SC]);
        setInt(cambios[2][2], Bus::getInstancia() -
>cambios_estado[C_SM][C_SM]);
        setInt(cambios[2][3], Bus::getInstancia() -
>cambios_estado[C_SM][C_M]);
        setInt(cambios[2][4], Bus::getInstancia() -
>cambios_estado[C_SM][C_I]);

```

```

        setInt(cambios[3][0], Bus::getInstancia() -
>cambios_estado[C_M][C_E]);
        setInt(cambios[3][1], Bus::getInstancia() -
>cambios_estado[C_M][C_SC]);
        setInt(cambios[3][2], Bus::getInstancia() -
>cambios_estado[C_M][C_SM]);
        setInt(cambios[3][3], Bus::getInstancia() -
>cambios_estado[C_M][C_M]);
        setInt(cambios[3][4], Bus::getInstancia() -
>cambios_estado[C_M][C_I]);
        setInt(cambios[4][0], Bus::getInstancia() -
>cambios_estado[C_I][C_E]);
        setInt(cambios[4][1], Bus::getInstancia() -
>cambios_estado[C_I][C_SC]);
        setInt(cambios[4][2], Bus::getInstancia() -
>cambios_estado[C_I][C_SM]);
        setInt(cambios[4][3], Bus::getInstancia() -
>cambios_estado[C_I][C_M]);
        setInt(cambios[4][4], Bus::getInstancia() -
>cambios_estado[C_I][C_I]);
        break;
    }
}

```

```

void CSMPCCacheView::ocultarInteraccionesCache()
{

```

```

    for(int i = 0; i < procn; i++)
    {
        flechasCI[i].ShowWindow(false);
        flechasCD[i].ShowWindow(false);
        textoCI[i].ShowWindow(false);
        flechaMem.ShowWindow(false);
        textoCD[i].ShowWindow(false);
    }
}

```

```

void CSMPCCacheView::actualizarVista()
{

```

```

    for (int i = 0; i < 8 ; i++)
    {
        if (i < procn)
        {
            proc[i].ShowWindow(true);
            flechasproc[i].ShowWindow(true);
            flechasCI[i].ShowWindow(false);
            flechaMem.ShowWindow(false);
            flechasCD[i].ShowWindow(false);
            textoCI[i].ShowWindow(false);
            textoCD[i].ShowWindow(false);
            textoproc[i].ShowWindow(true);
            if (fTrazas[i].IsEmpty())

```

```

                proc[i].SetBitmap(LoadBitmap(theApp.m_hInstance, MAKEINTRESOURCE(
IDB_PsinT)));
            else

```

```

                proc[i].SetBitmap(LoadBitmap(theApp.m_hInstance, MAKEINTRESOURCE(
IDB_PconT)));
            } else
            {
                proc[i].ShowWindow(false);

```

```

        flechasproc[i].ShowWindow(false);
        flechasCD[i].ShowWindow(false);
        flechasCI[i].ShowWindow(false);
        textoproc[i].ShowWindow(false);
        textoCI[i].ShowWindow(false);
        textoCD[i].ShowWindow(false);
    }
}
switch(protocolo)
{
case P_MSI:
    for(int i = 0; i < 5;i++)
    {
        cambios[i][3].ShowWindow(false);
        cambios[3][i].ShowWindow(false);
        cambios[i][4].ShowWindow(false);
        cambios[4][i].ShowWindow(false);
    }
    labelCambios[3][0].ShowWindow(false);
    labelCambios[4][0].ShowWindow(false);
    labelCambios[3][1].ShowWindow(false);
    labelCambios[4][1].ShowWindow(false);
    labelCambios[0][0].SetWindowTextW(_T("M"));
    labelCambios[1][0].SetWindowTextW(_T("S"));
    labelCambios[2][0].SetWindowTextW(_T("I"));
    labelCambios[0][1].SetWindowTextW(_T("M"));
    labelCambios[1][1].SetWindowTextW(_T("S"));
    labelCambios[2][1].SetWindowTextW(_T("I"));

    break;
case P_MESI:
    for(int i = 0; i < 5;i++)
    {
        if(i<=3)
        {
            cambios[i][3].ShowWindow(true);
            cambios[3][i].ShowWindow(true);
        }
        else
        {
            cambios[i][3].ShowWindow(false);
            cambios[3][i].ShowWindow(false);
        }
        cambios[i][4].ShowWindow(false);
        cambios[4][i].ShowWindow(false);
    }
    labelCambios[3][0].ShowWindow(true);
    labelCambios[4][0].ShowWindow(false);
    labelCambios[3][1].ShowWindow(true);
    labelCambios[4][1].ShowWindow(false);
    labelCambios[0][0].SetWindowTextW(_T("M"));
    labelCambios[1][0].SetWindowTextW(_T("E"));
    labelCambios[2][0].SetWindowTextW(_T("S"));
    labelCambios[3][0].SetWindowTextW(_T("I"));
    labelCambios[0][1].SetWindowTextW(_T("M"));
    labelCambios[1][1].SetWindowTextW(_T("E"));
    labelCambios[2][1].SetWindowTextW(_T("S"));
    labelCambios[3][1].SetWindowTextW(_T("I"));

    break;
case P_DRAGON:

```

```

        for(int i = 0; i < 5;i++)
        {
            cambios[i][3].ShowWindow(true);
            cambios[3][i].ShowWindow(true);
            cambios[i][4].ShowWindow(true);
            cambios[4][i].ShowWindow(true);
        }
        labelCambios[3][0].ShowWindow(true);
        labelCambios[4][0].ShowWindow(true);
        labelCambios[3][1].ShowWindow(true);
        labelCambios[4][1].ShowWindow(true);

        labelCambios[0][0].SetWindowTextW(_T("E"));
        labelCambios[1][0].SetWindowTextW(_T("SC"));
        labelCambios[2][0].SetWindowTextW(_T("SM"));
        labelCambios[3][0].SetWindowTextW(_T("M"));
        labelCambios[4][0].SetWindowTextW(_T("F"));
        labelCambios[0][1].SetWindowTextW(_T("E"));
        labelCambios[1][1].SetWindowTextW(_T("SC"));
        labelCambios[2][1].SetWindowTextW(_T("SM"));
        labelCambios[3][1].SetWindowTextW(_T("M"));
        labelCambios[4][1].SetWindowTextW(_T("F"));
        break;
    }
}

void CSMPCCacheView::mostrarProc()
{
    int i;
    for (i =0;i < procn ; i++)
    {
        proc[i].ShowWindow(true);
        flechasproc[i].ShowWindow(true);
        flechasCI[i].ShowWindow(true);
        flechasCD[i].ShowWindow(true);
        textoproc[i].ShowWindow(true);
    }
}

void CSMPCCacheView::ocultarProc()
{
    int i;
    for (i =0;i < procn ; i++)
    {
        proc[i].ShowWindow(false);
        flechasproc[i].ShowWindow(false);
        flechasCD[i].ShowWindow(false);
        flechasCI[i].ShowWindow(false);
        textoproc[i].ShowWindow(false);
    }
}

void CSMPCCacheView::cambiarFlechaProc(int pos)
{
    if(fParriba[pos])

        flechasproc[pos].SetBitmap(LoadBitmap(theApp.m_hInstance,MAKEINTRESOURCE(IDB_FAbajo)));
    else

```



```

        flechasproc[pos].SetBitmap(LoadBitmap(theApp.m_hInstance,MAKEINTRESOURCE(IDB_FArriba)));
        fParriba[pos] = !fParriba[pos];
    }

void CSMPCCacheView::cambiarTextoProc(int pos, CString texto)
{
    textoproc[pos].SetWindowTextW(texto);
}

void CSMPCCacheView::cambiarFlechaCI(int pos)
{
    if(fCIarriba[pos])

        flechasCI[pos].SetBitmap(LoadBitmap(theApp.m_hInstance,MAKEINTRESOURCE(IDB_FAbajo)));
    else

        flechasCI[pos].SetBitmap(LoadBitmap(theApp.m_hInstance,MAKEINTRESOURCE(IDB_FArriba)));
    fCIarriba[pos] = !fCIarriba[pos];
}

void CSMPCCacheView::cambiarTextoCI(int pos, CString texto)
{
    textoCI[pos].SetWindowTextW(texto);
}

void CSMPCCacheView::cambiarFlechaCD(int pos)
{
    if(fCDarriba[pos])

        flechasCD[pos].SetBitmap(LoadBitmap(theApp.m_hInstance,MAKEINTRESOURCE(IDB_FAbajo)));
    else

        flechasCD[pos].SetBitmap(LoadBitmap(theApp.m_hInstance,MAKEINTRESOURCE(IDB_FArriba)));
    fCDarriba[pos] = !fCDarriba[pos];
}

void CSMPCCacheView::OnBnClickedBiniciar()
{
    iniciarSimulacion();
    enEjecucion = true;
    bPausa.EnableWindow(true);
    bCompleta.EnableWindow(false);
    bIniciar.EnableWindow(false);
    enPausa = false;
}

void CSMPCCacheView::OnBnClickedBpausa()
{
    CString texto;
    if(enPausa)
    {
        texto = "Pausar";
        bPausa.SetWindowTextW(texto);
        enPausa = false;
    }
}

```

```

        else
        {
            texto = "Continuar";
            bPausa.SetWindowTextW(texto);
            bCompleta.EnableWindow(true);
            enPausa = true;
        }
    }

void CSMPCCacheView::OnBnClickedBCompleta()
{
    enEjecucion = true;
    bPausa.EnableWindow(true);
    bCompleta.EnableWindow(false);
    enPausa = false;
    CString aux;
    TotalAccesos.GetWindowTextW(aux);
    steps = _wtoi(aux)+1;
    char cadena[20];
    _itoa_s(steps, cadena, 10);
    TSteps.SetWindowTextW((CString)cadena);
    CString texto;
    texto = "Pausar";
    bPausa.SetWindowTextW(texto);
}

void CSMPCCacheView::OnBnClickedBsalir()
{
    this->CloseWindow();
    exit(0);
}

bool CSMPCCacheView::realizarIteracion()
{
    if(steps == 0) {OnBnClickedBpausa(); return true;}
    bool res = true;
    if(steps > 0)
    {
        res = Bus::getInstancia()->realizarIteracion();
    }
    if(steps <= Bus::getInstancia()->memAccesos)
    {
        OnBnClickedBpausa();
        steps = 1;
        char cadena[20];
        _itoa_s(steps, cadena, 10);
        TSteps.SetWindowTextW((CString)cadena);
    }
    if(!res)//fin ejecucion
    {
        bPausa.EnableWindow(false);
        bCompleta.EnableWindow(false);
        bIniciar.EnableWindow(true);
        enEjecucion = false;
        AfxMessageBox(_T("Simulacion terminada con exito!"));
    }
    actualizarControles();
    return res;
}

void CSMPCCacheView::OnEnChangeSteps()
{

```

```
CString texto;  
TSteps.GetWindowTextW(texto);  
steps = _wtoi(texto);  
}
```

Manual del programador

Introducción

A continuación se describe el proceso de desarrollo del proyecto SMPCache, 2º proyecto de Arquitectura e Ingeniería de computadores para el curso académico 2010/2011. Este software realiza una simulación de un multiprocesador simétrico, al cual se le pueden configurar diversas opciones.

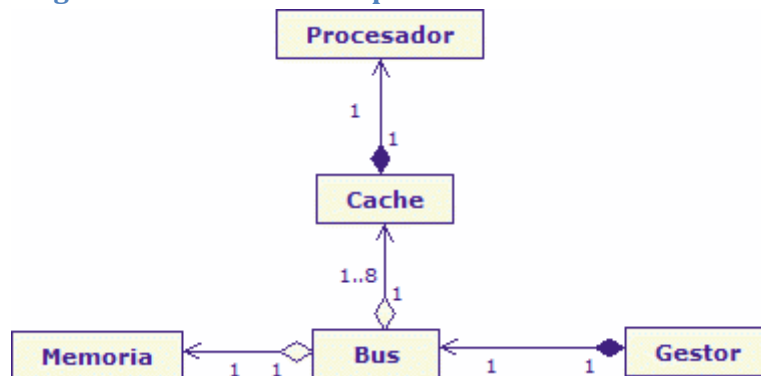
Análisis

Del análisis de los objetivos, hemos obtenido la siguiente información.

Identificación de clases potenciales

Bus, Cache, Procesador, Memoria y gestor.

Diagrama de modelo conceptual



Descripción de las clases Conceptuales

- **Bus**: Comunica las caches y realiza una iteración, contiene las estadísticas y otros valores.
- **Memoria**: Contiene los datos “virtuales”. [Desechada en la fase de diseño]
- **Cache**: A través de los protocolos indicados, hace la simulación según los datos/mensajes recibidos. Accede al procesador para obtener las instrucciones.
- **Procesador**: Carga las instrucciones desde fichero y las pasa a la cache.
- **Gestor**: Gestiona la parte grafica y de configuración de la aplicación, así como su ejecución.

Diseño

Diagrama de clases



Algoritmos de especial interés

- Control de la coherencia cache. [Cache.cpp]
- Arbitración del bus. [Bus.cpp]

Definición de entradas/salidas

La entrada del programa se realiza a través de un fichero de configuración y varios ficheros de traza. Toda la salida es vía pantalla. Se explica con más detenimiento el funcionamiento en el manual de usuario.

Variables o instancias más significativas

Cabe destacar la instancia de la clase Bus, ya que es un singleton con el objetivo de que sea accedida desde cualquier parte del programa, para que las caches puedan comunicar sus mensajes u otra información a este o para actualizar información estadística como ejemplo.

Lista de errores que el programa controla

El software desarrollado responde correctamente a una configuración errónea del fichero de configuración, pero este, al igual que el de traza debe estar bien construido.

Historial de desarrollo y Valoración personal

El proceso de desarrollo de este software de simulación ha sido largo y difícil a la vez que entretenido, al realizarse en un entorno de programación nuevo para nosotros. El lenguaje en sí no ha supuesto demasiada complicación aunque sí las librerías con las que hemos trabajado, ya que estábamos acostumbrados a trabajar con las librerías estándares de C++ y el cambio a las librerías de MS ha sido quizás un poco tedioso, ya que creemos, se complica demasiado por el propósito de mantener compatibilidad con versiones anteriores.

Al comienzo, nos centramos en la interfaz, desarrollándola totalmente antes de empezar a construir el código relativo a la simulación. Mencionar que una vez comenzado a construir el código fue necesario reestructurar completamente todo el proyecto debido a que nuestra primera concepción era que las instancias de los procesadores llevarían la ejecución.

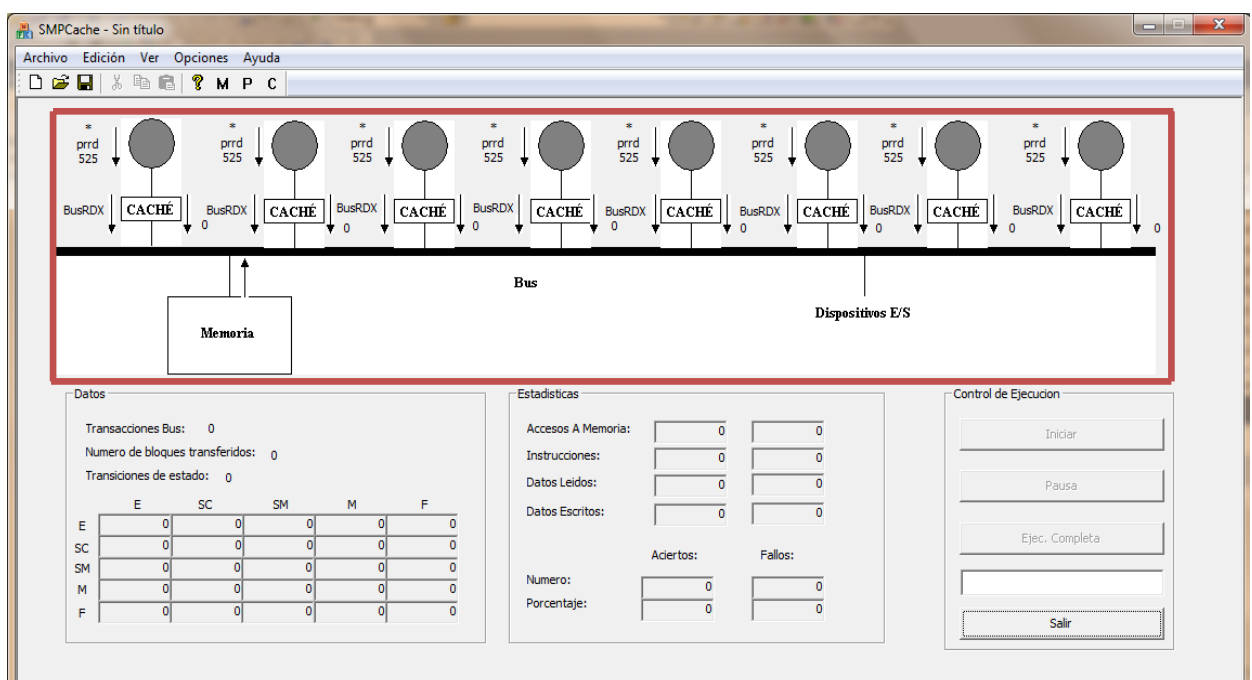
Tras terminar de construir el código y comenzar a realizar algunas trazas para comprobar el correcto funcionamiento de la aplicación fue necesario corregir algunos errores.

Manual de Usuario

Se nos ha pedido la realización de una práctica sobre memorias cachés en multiprocesadores, a continuación explicaremos como funciona nuestro entorno desarrollado .

Debemos indicar que se ha desarrollado en visual estudio, con lo cual para que podamos ejecutar correctamente nuestra aplicación será necesario tener un sistema operativo de Microsoft compatible con Microsoft framework 3.5.

A continuación mostraremos la pantalla principal desde la cual podremos realizar todas las opciones propuestas en la práctica:



Rectángulo rojo: En esta imagen observamos la cantidad de procesadores activos, así como toda la interacción entre los procesadores, caches y memoria principal, tanto si se hacen lecturas y escrituras a memoria o a otras cachés.

Datos: en este apartado se muestra la información relativa a las distintas transacciones de bus, número de bloques transferidos, transacciones de estado, y dependiendo del protocolo que utilizemos (MSI, MESI, o DRAGON), la cantidad de saltos entre los distintos estados.

Estadísticas: En este apartado podemos observar cómo transcurre la ejecución de nuestras trazas, existen los siguientes puntos:

Accesos a memoria: En el recuadro de la derecha se indican el número de accesos a memoria que se harán en total, teniendo en cuenta todos los procesadores con sus distintas

trazas, mientras que en el recuadro izquierdo aparecen el acceso a memoria actual, es decir el acceso que se está ejecutando en ese momento en relación al total.

Instrucciones: En el recuadro derecho se indica el número de instrucciones total que se van a ejecutar, mientras que en el recuadro izquierdo se indica la instrucción actual que se está ejecutando.

Datos leídos: Indica el número de datos leídos ejecutados en relación a los totales que realizará nuestra aplicación dependiendo de las trazas y procesadores que utilicemos.

Datos escritos: Indica el número de datos escritos ejecutados en relación a los totales que realizará nuestra aplicación dependiendo de las trazas y procesadores que utilicemos.

Aciertos y fallos: Finalmente en este apartado indicamos el número de aciertos y fallos en caché así como su porcentaje.

Control de ejecución:

En este apartado es donde interactuamos con nuestra aplicación. Está formada por los siguientes botones:

Iniciar: Cuando pulsemos este botón empezará a ejecutarse nuestra aplicación, previamente deberemos de cargar la configuración necesaria, es decir las distintas trazas que usaremos en los procesadores elegidos.

Pausa: Botón que nos permite pausar nuestra aplicación. Si pausamos la aplicación, este botón cambiará a continuar, pudiendo salir del estado de pausa al volver a pulsar el botón (esta vez llamado continuar) y continuaremos con la aplicación.

Ejecución completa: Si pulsamos este botón se realizará la ejecución completa de nuestra práctica, a no ser que pulsemos el botón pausar. Debemos de tener cargada previamente las trazas a utilizar antes de pulsar este botón, además también deberá de haber sido pulsado con anterioridad el botón iniciar.

Seguido de este botón hay un recuadro en el cual podemos indicar que la ejecución avance hasta el acceso a memoria indicado.

Salir: Si pulsamos este botón saldremos de nuestra aplicación.

Ejemplo de ejecución:

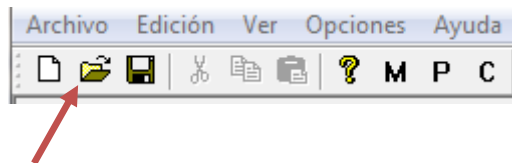
A continuación explicaremos un pequeño ejemplo de cómo realizar una simulación.

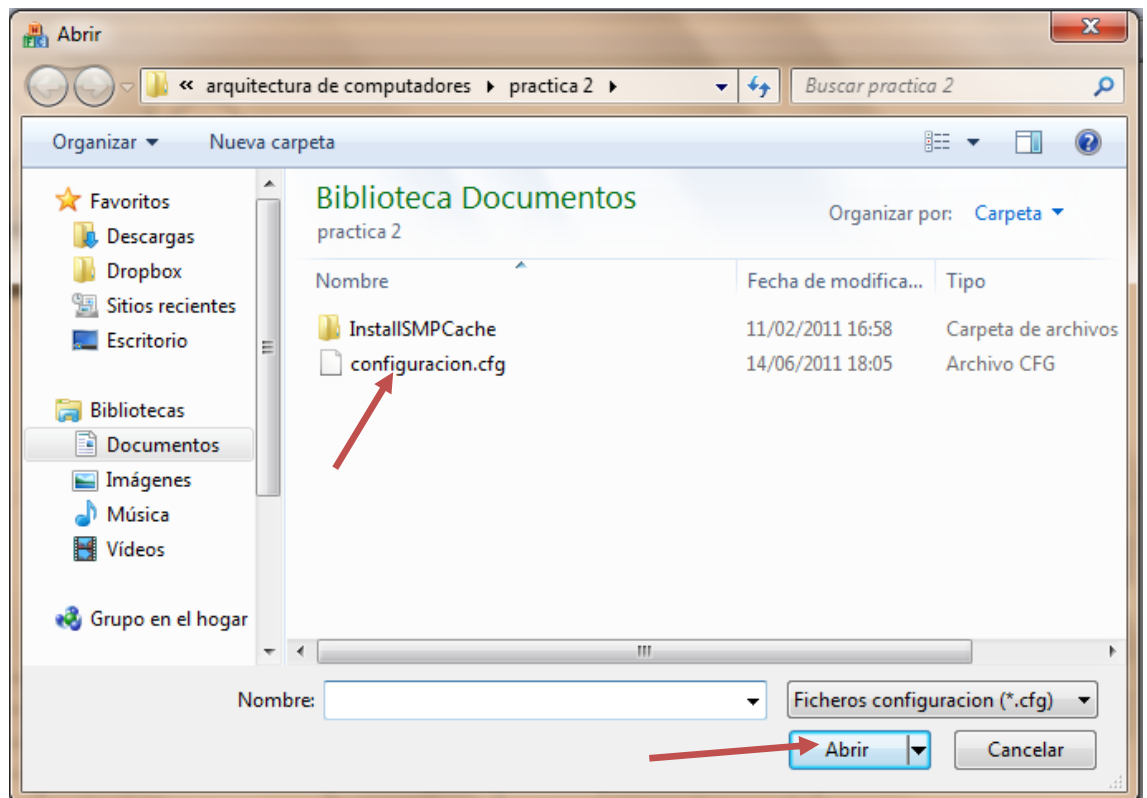
- 1) En primer lugar deberemos de tener un fichero .cfg en el que indicar la configuración que le daremos a nuestro simulador, con los procesadores y trazas de cada procesador utilizados. A continuación mostraremos un ejemplo de configuración con el cual realizaremos nuestro ejemplo propuesto.

```
N procesadores:
4
Protocolo coherencia cach 麁
3
```

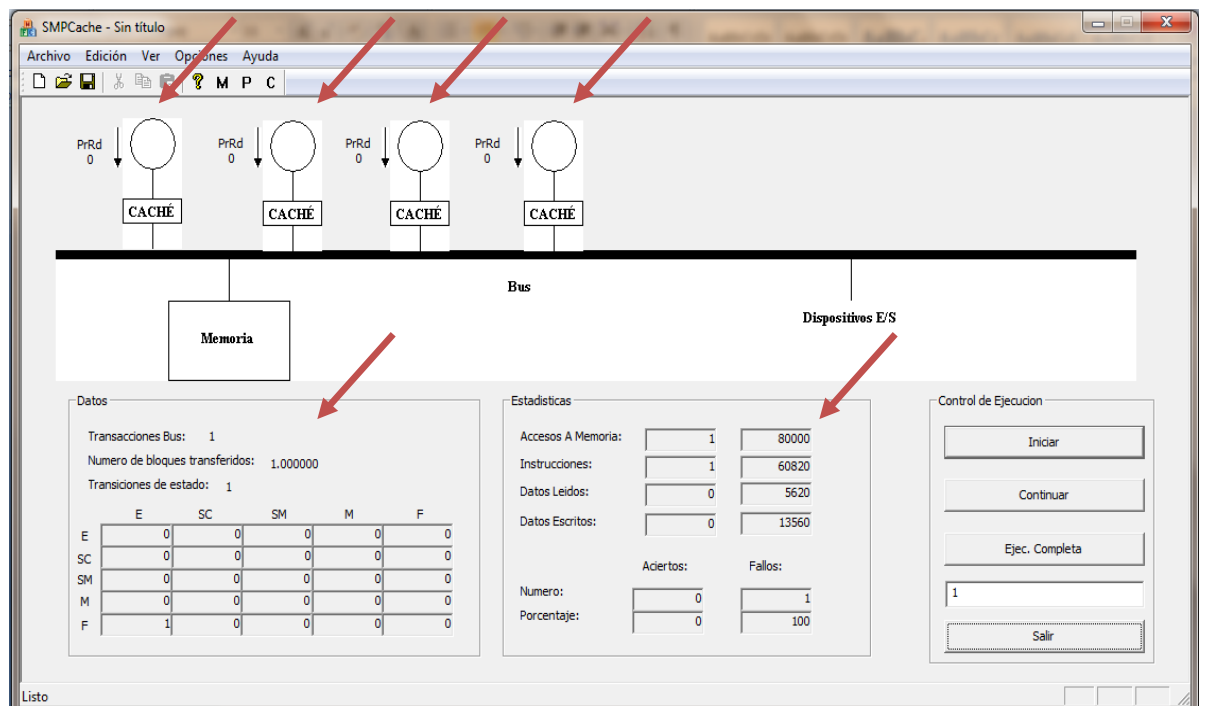

Arbitracion bus:
2
Ancho palabra(bits):
64
Palabras en un bloque:
1024
Bloquees en memoria:
1024
Bloques en cache:
64
Funcion correspondencia:
2
N conjuntos:
1
Algoritmo reemplazamiento:
2
Niveles de cache:
1
Politica de escritura:
2
Trazas de memoria:
P1="RUTA"\traza.prg
P2="RUTA"\traza.prg
P3="RUTA"\traza.prg
P4="RUTA"\traza.prg
P5=
P6=
P7=
P8=

- 2) Una vez que tenemos el fichero .cfg, debemos de indicárselo a nuestro simulador, para ello hacemos click en el botón abrir e indicamos la ruta en la que se encuentra nuestro fichero de configuración:

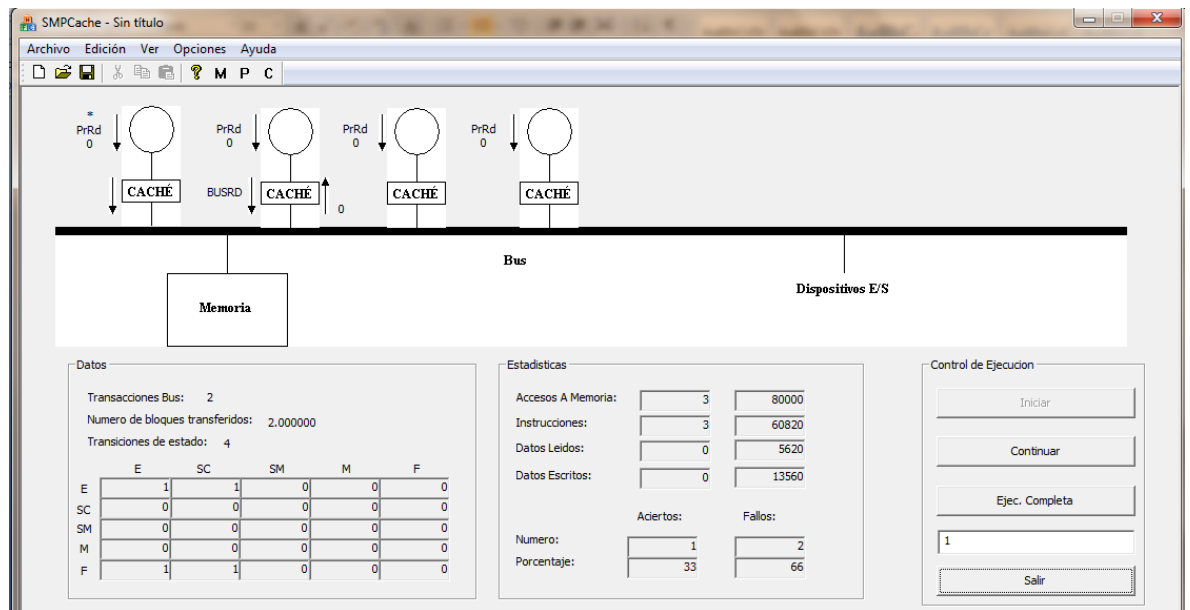




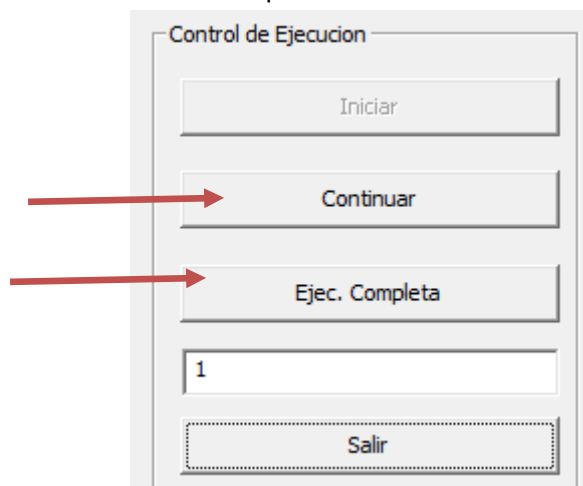
- 3) Una vez indicado el fichero de configuración que vamos utilizar, podemos observar cómo al indicar el fichero que vamos a usar se rellenan algunos datos en nuestra interfaz, así como los procesadores que vamos a utilizar:



Una vez mostrada esta información podemos pulsar el botón iniciar para ejecutar nuestra aplicación, y se ejecutará la primera instrucción.

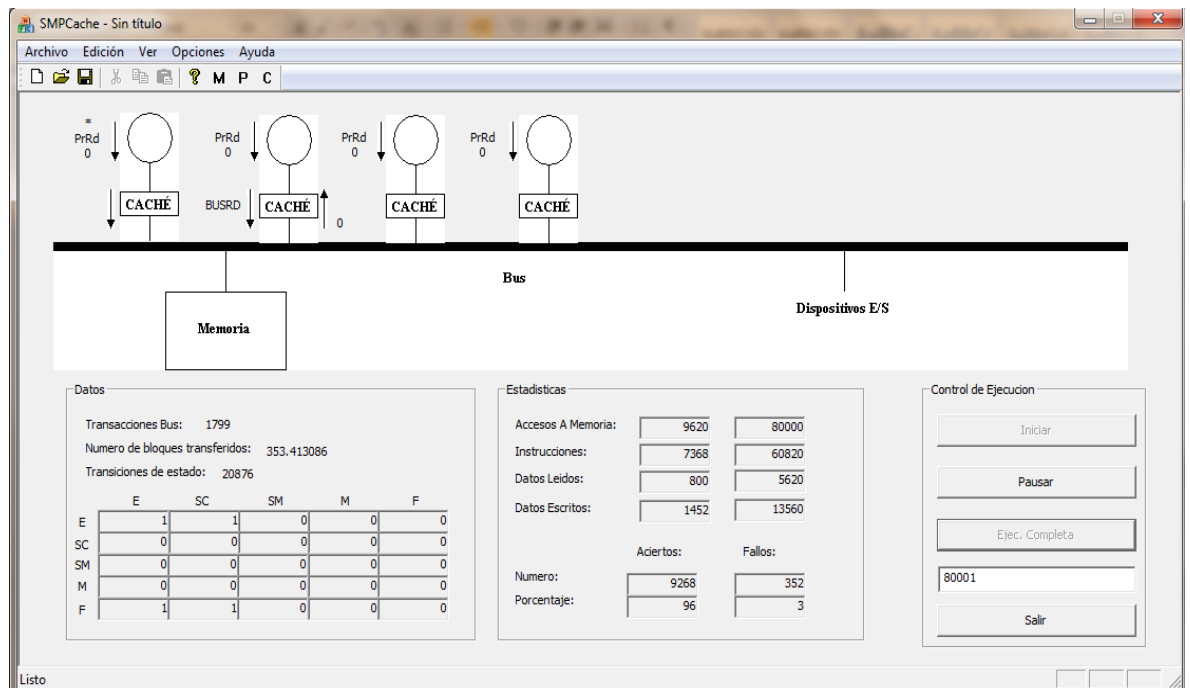


- 4) Ahora, podemos o bien, pulsar el botón de ejecución completa o ir instrucción a instrucción pulsando el botón continuar.

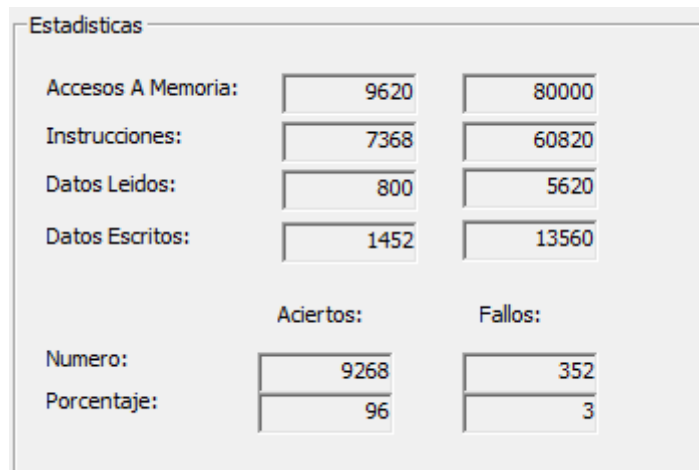


En este caso vamos a realizar la ejecución completa.

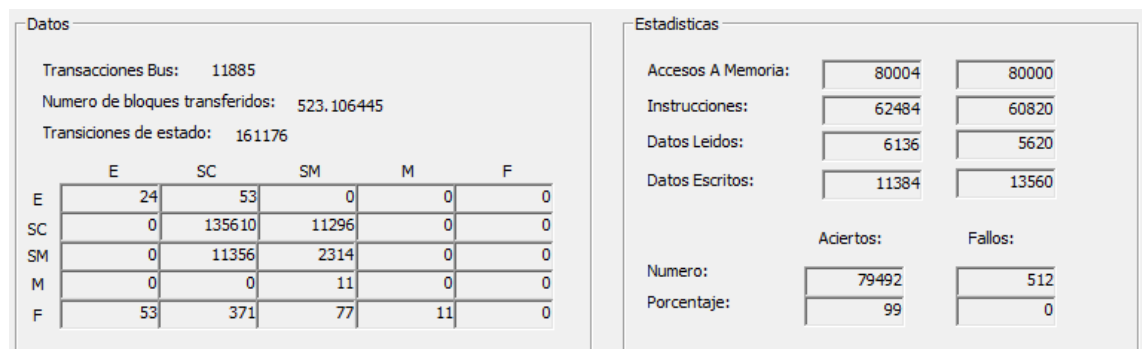
- 5) Para realizar la ejecución completa, debemos de pulsar el botón Ejec. Completa. A continuación mostraremos una imagen mientras se realiza este tipo de ejecución.



Como podemos observar la simulación se va realizando, y aumentan la información de las estadísticas, hasta que lleguemos a 80000 accesos a memoria.



- 6) La simulación terminará al llegar a los 80000 accesos a memoria. Una vez que haya terminado podremos ver más información como las distintas transacciones entre los estados.



- 7) De esta manera terminaría nuestra simulación.


Para terminar decir que hemos incluido en la aplicación otros tipos de ejecución como sería la ejecución paso a paso, pulsando únicamente el botón “continuar”

Continuar

O incluso podemos acceder al acceso de memoria deseado, indicándolo en el rectángulo de control de ejecución y pulsando el botón continuar.

Estadísticas			
<u>Accesos A Memoria:</u>	<input type="text" value="1"/>	<input type="text" value="80000"/>	
Instrucciones:	<input type="text" value="1"/>	<input type="text" value="60820"/>	
Datos Leídos:	<input type="text" value="0"/>	<input type="text" value="5620"/>	
Datos Escritos:	<input type="text" value="0"/>	<input type="text" value="13560"/>	
Aciertos:		Fallos:	
Numero:	<input type="text" value="0"/>	<input type="text" value="1"/>	
Porcentaje:	<input type="text" value="0"/>	<input type="text" value="100"/>	


Control de Ejecucion	
<input type="button" value="Iniciar"/>	
<input type="button" value="Continuar"/>	
<input type="button" value="Ejec. Completa"/>	
<input type="text" value="300"/>	
<input type="button" value="Salir"/>	



De esta manera llegaremos al acceso 300 de memoria:

Estadísticas			
<u>Accesos A Memoria:</u>	<input type="text" value="300"/>	<input type="text" value="80000"/>	
Instrucciones:	<input type="text" value="219"/>	<input type="text" value="60820"/>	
Datos Leídos:	<input type="text" value="16"/>	<input type="text" value="5620"/>	
Datos Escritos:	<input type="text" value="65"/>	<input type="text" value="13560"/>	
Aciertos:		Fallos:	
Numero:	<input type="text" value="264"/>	<input type="text" value="36"/>	
Porcentaje:	<input type="text" value="88"/>	<input type="text" value="12"/>	

Control de Ejecucion	
<input type="button" value="Iniciar"/>	
<input type="button" value="Continuar"/>	
<input type="button" value="Ejec. Completa"/>	
<input type="text" value="1"/>	
<input type="button" value="Salir"/>	



De esta manera daremos por finalizado nuestro manual de usuario.

Conclusión:

Tras realizar la práctica nos hemos dado cuenta del potencial que nos proporciona visual studio, aunque lo peor de todo es que lleguemos a estas alturas de la carrera sin saber nada sobre esta herramienta. Con respecto a la práctica decir que es una práctica bastante dura, en el sentido de que hay que invertir mucho tiempo en ella y es difícil, ya que es necesario tener bastantes conocimientos adquiridos antes de empezar a realizarla.

Comentar que la información propuestas por el profesor a lo largo de las prácticas es importante, aunque en nuestro caso la entrega del último documento hubiera sido mejor recibirla con mayor antelación, ya que nuestro código era bastante diferente al propuesto por el profesor. De esta manera nuestro código no sigue la misma dinámica al propuesto, aunque funciona correctamente. Suponiendo un esfuerzo adicional en comparación al invertido por otros compañeros.

Finalmente comentar que ha sido una práctica entretenida, e importante, ya que solo el simple hecho de utilizar visual estudio y ganar algo de soltura con él nos servirá en el día de mañana.

La satisfacción obtenida al terminar la práctica es alta, y hemos estado muy contentos con la realización de la misma.