```
//=======================================
// Compiler: MSL
// Project:  TestIO
// Bestand:  FileCopyAsyncBehind.h
// Implemen: FileCopyAsyncBehind.cp
//=======================================

#pragma once

int main();

bool Test();

bool Test(z2 invoer_refNum,
          z4 invoer_lengte);

bool Test(z2 invoer_refNum,
          z4 invoer_lengte,
          z2 uitvoer_refNum);

bool Test(z2 invoer_refNum,
          z4 invoer_lengte,
          z2 uitvoer_refNum,
          p_char buffer);

bool Test(z2 invoer_refNum,
          z4 invoer_lengte,
          z2 uitvoer_refNum,
          p_char buffer,
          IOCompletionUPP leesCompletion);

bool Test(z2 invoer_refNum,
          z4 invoer_lengte,
          z2 uitvoer_refNum,
          p_char buffer,
          IOCompletionUPP leesCompletion,
          IOCompletionUPP schrijfCompletion);

bool VraagOmEenBlokTeLezen(z4 lengte,
                           r_ParamBlockRec pb);

bool VraagOmEenBlokTeSchrijven(z4 lengte,
                               r_ParamBlockRec pb);

pascal void HetIsGelezen(p_ParamBlockRec paramBlock);
```

```
pascal void HetIsGeschreven(p_ParamBlockRec paramBlock);
```

```
//=========================================
// Compiler: MSL
// Project:  TestIO
// Bestand:  FileCopyAsyncBehind.cp
//=========================================

#include "FileCopyAsyncBehind.h"

int main()
{
    cout << "Start van het programma.";
    cout << endl;

    cout << "Typ Command-Q om te stoppen.";
    cout << endl;

    c_n8 start=Tijd::Geef_microseconden();
    c_bool ok=Test();
    c_n8 einde=Tijd::Geef_microseconden();
    c_n8 duur=einde-start;
    cout << "Duur=";
    cout << duur;
    cout << endl;

    if (!ok)
        {
        cout << "Het programma mislukt.";
        cout << endl;
        }

    cout << "Einde van het programma.";
    cout << endl;

    return 0;
}

bool Test()
{
    // 1. Maak invoer spec.
    FSSpec invoerSpec;
    OSErr err=Files::OSERR_FSMakeFSSpec(0,
                                        0,
                                        "\pKangaroo:Test.bin",
                                        invoerSpec);
    if (err!=osErr_noErr)
```

```
        {
        cout << "Pad van invoer is niet gevonden.";
        cout << endl;
        return false;
        }

    // 2. Open invoer.
    z2 invoer_refNum;
    err=Files::OSERR_FSpOpenDF(invoerSpec,
                               permission_rd,
                               invoer_refNum);
    if (err!=osErr_noErr)
        {
        cout << "Open data fork mislukt.";
        cout << endl;
        return false;
        }

    // 3. Zoek invoer lengte.
    z4 invoer_lengte;
    err=Files::OSERR_GetEOF(invoer_refNum,
                            invoer_lengte);
    if (err!=osErr_noErr)
        {
        cout << "Lees data fork lengte mislukt.";
        cout << endl;
        return false;
        }

    // 4. Test.
    Test(invoer_refNum,
         invoer_lengte);

    // 5. Sluit invoer.
    err=Files::OSERR_FSClose(invoer_refNum);
    if (err!=osErr_noErr)
        {
        cout << "Sluit data fork mislukt.";
        cout << endl;
        return false;
        }

    return true;
}
```

```
bool Test(c_z2 invoer_refNum,
          c_z4 invoer_lengte)
{
    // 1. Maak uitvoer spec.
    FSSpec uitvoerSpec;
    OSErr err=Files::OSERR_FSMakeFSSpec(0,
                                        0,
                                        "\pTemp:Uitvoer:Test.bin",
                                        uitvoerSpec);
    if (err!=osErr_noErr)
        {
        if (err!=osErr_fnfErr)
            {
            cout << "Maak pad van uitvoer mislukt.";
            cout << endl;
            return false;
            }
        }

    // 2. Maak uitvoer.
    err=Files::OSERR_FSpCreate(uitvoerSpec,
                               'TEMP',
                               'TEMP',
                               scriptCode_system);
    if (err!=osErr_noErr)
        {
        cout << "Maak nieuw bestand mislukt.";
        cout << endl;
        return false;
        }

    // 3. Open uitvoer.
    z2 uitvoer_refNum;
    err=Files::OSERR_FSpOpenDF(uitvoerSpec,
                               permission_wr,
                               uitvoer_refNum);
    if (err!=osErr_noErr)
        {
        cout << "Open nieuwe data fork mislukt.";
        cout << endl;
        return false;
        }

    // 4. Test.
    if (invoer_lengte)
```

```
        {
        Test(invoer_refNum,
             invoer_lengte,
             uitvoer_refNum);
        }

    // 5. Sluit uitvoer.
    err=Files::OSERR_FSClose(uitvoer_refNum);
    if (err!=osErr_noErr)
        {
        cout << "Sluit nieuwe data fork mislukt.";
        cout << endl;
        return false;
        }

    return true;
}

bool Test(c_z2 invoer_refNum,
          c_z4 invoer_lengte,
          c_z2 uitvoer_refNum)
{
    // 1. Maak buffer.
    p_char buffer;
    OSErr err=Memory::OSERR_NewPtr(invoer_lengte,
                                   buffer);
    if (err!=osErr_noErr)
        {
        cout << "Te weinig geheugen voor buffer.";
        cout << endl;
        return false;
        }

    // 2. Test.
    Test(invoer_refNum,
         invoer_lengte,
         uitvoer_refNum,
         buffer);

    // 3. Wis buffer.
    err=Memory::OSERR_DisposePtr(buffer);
    if (err!=osErr_noErr)
        {
        cout << "Wis buffer pointer mislukt.";
        cout << endl;
```

```
        return false;
        }

    return true;
}

bool Test(c_z2 invoer_refNum,
          c_z4 invoer_lengte,
          c_z2 uitvoer_refNum,
          cp_char buffer)
{
    // 1. Maak UPP.
    c_IOCompletionUPP leesCompletion=NewIOCompletionUPP(HetIsGelezen);
    if (!leesCompletion)
        {
        cout << "Maak lees completion UPP mislukt.";
        cout << endl;
        return false;
        }

    // 2. Test.
    Test(invoer_refNum,
         invoer_lengte,
         uitvoer_refNum,
         buffer,
         leesCompletion);

    // 3. Wis UPP.
    DisposeIOCompletionUPP(leesCompletion);

    return true;
}

bool Test(c_z2 invoer_refNum,
          c_z4 invoer_lengte,
          c_z2 uitvoer_refNum,
          cp_char buffer,
          c_IOCompletionUPP leesCompletion)
{
    // 1. Maak UPP.
    c_IOCompletionUPP schrijfCompletion=NewIOCompletionUPP(HetIsGeschreven);
    if (!schrijfCompletion)
        {
        cout << "Maak schrijf completion UPP mislukt.";
        cout << endl;
```

```
        return false;
        }

    // 2. Test.
    Test(invoer_refNum,
         invoer_lengte,
         uitvoer_refNum,
         buffer,
         leesCompletion,
         schrijfCompletion);

    // 3. Wis UPP.
    DisposeIOCompletionUPP(schrijfCompletion);

    return true;
}

enum Staat
{
    start,
    gevraagd,
    gelukt,
    mislukt,
    einde
};

typedef volatile Staat v_Staat;

v_Staat lezen_staat=start;
v_z4 lezen_teDoen;

v_Staat schrijven_staat=start;
v_z4 schrijven_teDoen;

bool Test(c_z2 invoer_refNum,
          c_z4 invoer_lengte,
          c_z2 uitvoer_refNum,
          p_char buffer,
          c_IOCompletionUPP leesCompletion,
          c_IOCompletionUPP schrijfCompletion)
{
    lezen_teDoen=invoer_lengte;
    schrijven_teDoen=invoer_lengte;

    ParamBlockRec invoer_paramBlock;
```

```
    IOParam &in=invoer_paramBlock.ioParam;
    Blok::Zero(sizeof(IOParam),
               &in);
    in.ioCompletion=leesCompletion;
    in.ioRefNum=invoer_refNum;
    in.ioBuffer=buffer;
    in.ioPosMode=0;
    in.ioPosOffset=0;

    ParamBlockRec uitvoer_paramBlock;
    IOParam &uit=uitvoer_paramBlock.ioParam;
    Blok::Zero(sizeof(IOParam),
               &uit);
    uit.ioCompletion=schrijfCompletion;
    uit.ioRefNum=uitvoer_refNum;
    uit.ioBuffer=buffer;
    uit.ioPosMode=0;
    uit.ioPosOffset=0;

    c_z4 blok_grootte=8*1024;

    cout << "Blokgrootte = ";
    cout << blok_grootte;
    cout << endl;

    z4 aGelukt=0;

    z4 aLeesNu;
    z4 aSchrijfNu;

    while ((lezen_staat!=einde) or (schrijven_staat!=einde))
        {
        switch (lezen_staat)
            {
            case gevraagd:
            case einde:
                {
                break;
                }
            case start:
                {
                aLeesNu=(lezen_teDoen<blok_grootte)
                        ? lezen_teDoen
                        : blok_grootte;
                if (!VraagOmEenBlokTeLezen(aLeesNu,
```

```
                                        invoer_paramBlock))
                {
                cout << "Vraag om een blok te lezen mislukt.";
                cout << endl;
                return false;
                }
            break;
            }
        case gelukt:
            {
            aGelukt++;
            lezen_staat=lezen_teDoen ? start : einde;
            break;
            }
        case mislukt:
            {
            cout << "Lezen mislukt.";
            cout << endl;
            return false;
            }
        }

    switch (schrijven_staat)
        {
        case gevraagd:
        case einde:
            {
            break;
            }
        case start:
            {
            if (!aGelukt)
                {
                break;
                }
            aSchrijfNu=(schrijven_teDoen<blok_grootte)
                       ? schrijven_teDoen
                       : blok_grootte;
            if (!VraagOmEenBlokTeSchrijven(aSchrijfNu,
                                           uitvoer_paramBlock))
                {
                cout << "Vraag om een blok te schrijven mislukt.";
                cout << endl;
                return false;
                }
```

```
                aGelukt--;
                break;
                }
            case gelukt:
                {
                schrijven_staat=schrijven_teDoen ? start : einde;
                break;
                }
            case mislukt:
                {
                cout << "Schrijven mislukt.";
                cout << endl;
                return false;
                }
            }
        }

    return true;
}

bool VraagOmEenBlokTeLezen(c_z4 lengte,
                           r_ParamBlockRec pb)
{
    //cout << "+";
    pb.ioParam.ioReqCount=lengte;
    lezen_staat=gevraagd;
    c_OSErr err=Files::OSERR_PBReadAsync(pb);
    if (err!=osErr_noErr)
        {
        cout << "Vraag om te lezen mislukt.";
        cout << endl;
        return false;
        }
    return true;
}

bool VraagOmEenBlokTeSchrijven(c_z4 lengte,
                               r_ParamBlockRec pb)
{
    //cout << "-";
    pb.ioParam.ioReqCount=lengte;
    schrijven_staat=gevraagd;
    c_OSErr err=Files::OSERR_PBWriteAsync(pb);
    if (err!=osErr_noErr)
        {
```

```
        cout << "Vraag om te schrijven mislukt.";
        cout << endl;
        return false;
        }
    return true;
}

pascal void HetIsGelezen(cp_ParamBlockRec paramBlock)
{
    c_OSErr err_lezen=paramBlock->ioParam.ioResult;
    if (err_lezen)
        {
        lezen_staat=mislukt;
        return;
        }
    c_z4 aGevraagd=paramBlock->ioParam.ioReqCount;
    c_z4 aGelezen=paramBlock->ioParam.ioActCount;
    if (aGelezen<aGevraagd)
        {
        lezen_staat=mislukt;
        return;
        }
    lezen_staat=gelukt;
    lezen_teDoen-=aGelezen;
}

pascal void HetIsGeschreven(cp_ParamBlockRec paramBlock)
{
    c_OSErr err_schrijven=paramBlock->ioParam.ioResult;
    if (err_schrijven)
        {
        schrijven_staat=mislukt;
        return;
        }
    c_z4 aGevraagd=paramBlock->ioParam.ioReqCount;
    c_z4 aGeschreven=paramBlock->ioParam.ioActCount;
    if (aGeschreven<aGevraagd)
        {
        schrijven_staat=mislukt;
        return;
        }
    schrijven_staat=gelukt;
    schrijven_teDoen-=aGeschreven;
}
```