# Async

Michael Zucchi

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>Async | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | Michael Zucchi | July 22, 2024 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# Chapter 1

# Async

## 1.1 Async.guide

```
Module afficheur de fichier asynchrone pour AmigaE2.5+

© 1993 Michael Zucchi
Tous Droits Réservés

Ce document décrit l'utilisation d'une suite de routines de lecture de
fichier de façon asynchrone fait pour le langage AmigaE. L'interface est
faite pour suivre les appels dela dos.library V36 le plus fidèlement que
possible.

Les sections suivantes sont disponibles:

        SurVol          quelques idées derrière le module

       General functions

       as_Open()           pour ouvrir un fichier
       as_Close()         pour fermer un fichier
       as_Read()           pour lire un fichier

       Fonctions de haut niveau

       as_FGetS()         pour lire les lignes de texte
       as_FGetC()         pour lire caractère par caractère

       Fonctions de bas niveau

       as_NextBuffer()       pour accèder directement aux tampons


        Exemples

NOTE: Ce module a besoin du Workbench 2.0 (V36) ou plus!  SVP
assurez vous que cette version du système est présente avant l'utilisation
de ces fonctions.
```

## 1.2 Survol du Module

Qu'entend-on par 'entrée/sortie asynchrone' ?

Lorsque la plupart des programmes utilise la dos.library pour écrire/lire des fichiers, ils appellent simplement Read() ou Write(). Que ce passe-t-il éxamine le gestionnaire de fichier qui leur est passé pour information sur le gestionnaire qui gère le fichier, et créé un dos 'packet' de ces informations (voir dos/dosextens.m pour voir à quoi ressemble un 'packet') ? Ce 'packet' est alors envoyé via le système d'envoie standard de message au gestionnaire qui gère le fichier. Les appels de fonction dos alors attend une réponse (reply) via le port message du processus pr_MessagePort - ie ils attendent jusqu'à ce que le système de fichier et gestionnaire aient récupéré l'information avant de revenir (et par conséquent le fait que les réponses parvienne au travers de pr_MessagePort est la raison pour laquelle le dos ne peut pas être appelé à partir d'une tâche 'standard'). Avec un device io lent (par exemple, un lecteur de disquette) toute cette attente peut signifier que le cpu est en attente la plupart du temps des données arrivant.

Comment résolvez-vous cette situation moins qu'idéale ? C'est simple. Vous pouvez créer vos propres 'packets' et les envoyer directement au dos. De cette façon, une port particulier de réponse peut être mis pour ces 'packets', et des demandes en lecture (ou écriture) peuvent être envoyés immédiatement, et les données lues quand les 'packets' sont retournés. Si quelque chose a besoin d'être fait pendant que le système de fichier s'occupe de ces données, alors votre programme peut le faire, sans avoir à attendre.

C'est à la base ce que async.m fait. Pour le moment, seul la lecture est supportée, mais l'écriture sera ajoutée dansle futur, avec des fonctions utiles comme Seek() etc. J'ai eu l'idée de le faire en voyant des codes que j'ai eu de la BBS locale, quelque chose d'un des gars sympas de Commodore, je crois.

## 1.3 Celui qui l'a écrit

J'ai écrit ce code il y a quelques temps, surtout pour un utilitaire de répertoire 'multi-threaded' sur lequel je travaille de temps en temps. Je l'ai trouvé tellement pratique pour ajouter un la puissance à quasiment tout ce que j'ai écrit, que je pense que d'autres personnes le trouveront utile aussi.

En ce moment, j'étudie 'de temps en temps' (:-) pour obtenir le diplome d'ingénieur système ordinateur (Computer Systems Engineering degree) à l'Univerity d'Australie du Sud.
Je suis 'Zed' de FRONTIER à mes heures anti-os.

Je peux être joint comme suit:

Internet email:

9107047w@lux.levels.unisa.edu.au
jusqu'à la fin '94 au plus - fiable

```
lettre 'Real Mode' (tm):

Michael Zucchi
PO BOX 824
Waikerie
South Australia 5330
       lent, mai très fiable - jusqu'à ce que ma mère vende la maison :)

Michael Zucchi
110 Dunrobin Rd
Warradale
South Australia 5046
       à ma porte - jusqu'à ce que je déménage (?)
```

## 1.4 as_open

```
async.m/as_Open                                                async.m/as_Open

 Syntaxe

       file := as_Open( name:PTR TO CHAR,
                        mode:LONG,
                        count:LONG,
                        size:LONG );

 Description
       Opens an asynchronous file, and returns a pointer to a (private) file
       handle.  When called, packets will be sent to the appropriate handler
       to fill all buffers, and the return will call immediately.

 Entrée
       name    A string, describing the name of the file to open
       mode    Same as mode in dos.library/Open.  Must be MODE_OLDFILE
               for now.
       count   Number of buffers to allocate.  3 works very well.
       size    The size of each buffer to allocate.  Above 5000 works
               well, must be a multiple of 4.

 Sorties
       file    A pointer to a filehandle that may be passed to the other
               async functions.

 Notes
       No sanity checking is done on any of the input values.  Use
       reasonable values for everything.
       The filehandle returned by as_Open() is NOT compatible with normal
       dos filehandles, and system calls!

 Voir aussi
       as_Close(), as_NextBuffer(), as_Read(), as_FGetS(), as_FGetC()
```

## 1.5 as_close

async.m/as_Close async.m/as_Close

Syntaxe

```
as_Close( file:LONG );
```

Description

Closes the file, free's all memory buffers and cleans up all outstanding packets. This call may be made as any time on a valid async filehandle.

Entrée

file valid filehandle from as_Open(), or NIL in which case nothing happens.

Sorties

Notes

Voir aussi

as_Open()

## 1.6 as_read

async.m/as_Read async.m/as_Read

Syntaxe

```
bytes := as_Read( file:LONG,
                  buffer:PTR TO CHAR,
                  number:LONG );
```

Description

as_Read reads a number of bytes ('number') into the buffer specified by 'buffer', from the async file 'file'.

The number of bytes actually read in is indicated by the return value. A return of zero indicates end of file, and errors are flagged by a return value of -1.

Entrée

file Only a valid filehandle from as_Open() is allowed.
buffer A pointer to at least 'number' bytes of memory to store the data. May be arbitrarily aligned.
number Specifies the number of bytes to read. number=0 is ignored.

Sorties

bytes The number of bytes actually stored in 'buffer'. A value of zero indicates end of file, and -1 that a file error has occurred, check IoErr() for detail.

Notes

Voir aussi
as_Open(), as_Close(), as_NextBuffer(), as_FGetS(), as_FGetC()

## 1.7 as_fgets

async.m/as_FGetS async.m/as_FGetS

Syntaxe

```
buffer := as_FGetS( file:LONG,
                    buffer:PTR TO CHAR,
                    number:LONG );
```

Description
Reads upto 'size' bytes from the file 'file' into the buffer pointed to by the buffer parameter. Stops reading at end of file or once a NEWLINE ($0a) character is encountered. Returns a pointer to that buffer or NIL on end of file or error.

The string stored in the buffer is NULL terminated.

Entrée
file A valid filehandle from as_Open().
buffer A pointer to at least 'number' bytes of memory to store the data. May be arbitrarily aligned.
number Specifies the number of bytes to read, at maximum. This MUST be >2.

Sorties
buffer Same as 'buffer' passed as an input, or NIL on end of file or file error.

Notes
If the line is too long to fit, the input stream is not skipped till the next linefeed.

Voir aussi
as_Open(), as_Close(), as_Read(), as_NextBuffer(), as_FGetC()

## 1.8 as_fgetc

async.m/as_FGetC async.m/as_FGetC

Syntaxe

```
char := as_FGetC( file:LONG );
```

Description
Reads the next character from the input file. Returns -1 on error or end of file. The character is an unsigned 32 bit quantity.

```
 Entrée
        file    A valid filehandle from as_Open().

 Sorties
        char    The next available byte from the input stream, or -1 on
                error.

 Notes
        This call is about as efficient as possible.

 Voir aussi
        as_Open(), as_Close(), as_Read(), as_NextBuffer(), as_FGetS()
```

## 1.9 as_nextbuffer

```
async.m/as_NextBuffer                                   async.m/as_NextBuffer

 Syntaxe

        buffer,valid := as_NextBuffer( file:LONG );

 Description
        Returns the next available data buffer in the file.  If end of file
        has not yet been reached, and a buffer has now been made free, then
        another read request is sent to the filesystem, in an asynchronous
        manner.

 Entrée
        file    A valid filehandle from as_Open().

 Sorties
        buffer  The address of the internal data buffer, 0 for end of file,
                or -1 on a file error.
        valid   Number of valid bytes in the buffer.  This will be the same
                as the size of each buffer as specified when the file was
                opened, unless it is the last buffer being read.

 Notes
        The call is NOT compatible with any of the other reading functions.
        If you call those functions, you must NOT call this function, and
        visa-versa.  It is a low level function which both of the other read
        functions make use of directly, and should only be used (esclusively)
        where extra performance/lower memory use is required.

 Voir aussi
        as_Open(), as_Close(), as_Read(), as_FGetS(), as_FGetC()
```

## 1.10 Informations sur les exemples

```
Inclue dans ce répertoire, quelques exemples d'utilisation de ce module.

 typef
```

C’est un exemple simple démontrant l’utilisation de as_FGetS().
Il affiche simplement un texte dansle shell actuel - un peu plus
rapidement que ce que c:type fait.

usage:
        typef [Name] <filename>

PlaySamp

C’est un exemple non trivial de la fonction as_Read(). C’est un
player complet de ’raw sample’ qui peut être utilisé pour jouer
n’importe des samples de n’importe quelle taille à partir du disque.

usage:
        PlaySamp [Name] <file1> [<file2> ... ] RATE <rate>

histogram

Un exemple simple d’utilisation de la commande as_NextBuffer(). Il
compte les occurance de chqua octets dans le fichier, et donne un
rapport à la fin.

usage:
        histogram [Name] <filename>

Coder un exemple plus utile pour as_NextBuffer() demande un peu plus
de travail que je n’avais de temps pour le faire :)