# AsyncIO

Magnus Holmgren

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>AsyncIO | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | Magnus Holmgren | December 31, 2022 | |

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

# Contents

# Chapter 1

# AsyncIO

## 1.1 AsyncIO.guide

```
                 AsyncIO library

        Version 39.2

  Original code by Martin Taillefer
        updates by Magnus Holmgren
```

AsyncIO is a library providing double buffered asyncronous IO to files in a DOS-like manner.



## 1.2 AsyncIO.guide/Background

Background

Reading and writing data is crucial to most applications and is in many cases a major bottleneck. Using AmigaDOS' sophisticated file system architecture can help reduce, and sometimes eliminate, the time spent waiting for IO to complete. This package offers a few small routines that can greatly improve an application's IO performance.

Normally, an application processes a file in a manner similar to the following:

1 - Open the file

2 - Read some data

3 - Process data just read

4 - Repeat steps 2 and 3 until all data is processed

5 - Close file

Although the above sequence works fine, it doesn't make full use of the Amiga's multitasking abilities. Step 2 in the above can become a serious bottleneck. Whenever the application needs some data by using the DOS Read() function, AmigaDOS has to put that task to sleep, and initiate a request to the file system to have it fetch the data. The file system then starts up the disk hardware and reads the data. Once the data is read, the application is woken up and can start processing the data just read.

The point to note in the above paragraph is that when the file system is reading data from disk, the application is asleep. Wouldn't it be nice if the application could keep running while data is being fetched for it?

Most Amiga hard drives make use of DMA (Direct Memory Access). DMA enables a hard drive to transfer data to memory at the same time as the CPU does some work. This parallelism is what makes the set of accompanying routines so efficient. They exploit the fact that data can be transfered to memory while the application is busy processing other data.

Using the asynchronous IO routines, an application's IO happens like this:

1 - Open the file, ask the file system to start reading ahead

2 - Read some data, ask the file system to read more data

3 - Process data

4 - Repeat steps 2 and 3 until all data is processed

5 - Close file

Immediately after opening the file, a request is sent to the file system to get it reading data in the background. By the time the application gets around to reading the first byte of data, it is likely already in memory. That means the application doesn't need to wait and can start processing the data. As soon as the application starts processing data from the file, a second request is sent out to the file system to fill up a second buffer. Once the application is done processing the first buffer, it starts processing the second one. When this happens, the file system starts filling up the first buffer again with new data. This process continues until all data has been read.

The whole technique is known as "double-buffered asynchronous IO" since it uses two buffers, and happens in the background (asynchronously).

The set of functions presented below offers high-performance IO using the technique described above. The interface is very similar to standard AmigaDOS files. These routines enable full asynchronous read/write of any file.

## 1.3 AsyncIO.guide/Contents

Contents

The archive contains the following drawers:

```
dlib      Link libraries for DICE.
emodules    "Include" files for E. These are currently not up to date.
include     Include files for C:
  clib    Function prototypes.
  diceclib  Special header for DICE 3.0 or higher, and is used
      when the "-mi" option is used. Similar to the
      proto file for SAS. Place them in DInclude:clib.
  fd    Function descriptor, e.g. used by various
      programming language tools (not really a C include
      file).
  libraries General include file(s), for needed structures
      and constants.
  pragmas   Pragma file for DICE and SAS/C (and perhaps
      StormC).
  proto   Proto file for SAS/C.
lib     Link libraries for SAS/C.
libs      The shared library.
src     Source code for AsyncIO.
```

## 1.4 AsyncIO.guide/Usage

Usage

AsyncIO is available both as a C link library, and as a shared library. Their usage should be fairly straight forward. There are a couple of details though, since some of the include files are used for both the shared and the link library. I'll only adress the use of AsyncIO from C here, as this is more or less the only language I use (in addition to some assembler).

Using the shared library

When using the shared library, make sure ASIO_SHARED_LIB is defined before you include any header file for AsyncIO.

```
DICE: Include <clib/asyncio_protos.h>. Then link with
  dlib/asynciolib#?.lib, if you want the autoinit code, or you didn't
  use the "-mi" option. If "-mi" is used, remeber to copy the file
  "include/diceclib/asyncio_protos.h" to "dinclude:clib".
```

SAS/C: Include the file <proto/asyncio.h>, and link with lib/asyncio.lib
if you need the autoinit code. The proto file defines
ASIO_SHARED_LIB for you, so this should be all you need to do.
Currently there is no glue code available.

Using the link library

DICE: Simply define ASIO_REGARGS when you want the regards
version, and link with the proper library. If you use dcc for
linking, DICE should link with the right library. Otherwise you
will get linker errors.

SAS/C: Do not include <proto/asyncio.h>. Include <clib/asyncio_protos>
instead. It is very important that you link with the right library,
and define ASIO_REGARGS and link with "lib/asyncior.lib" when you
want the regargs version.

Warning: If you link with the wrong library, the program will link
properly, but will show strange errors when run! The reason for
this "problem" is that I wanted to be able to use the regargs
version from assembler, without assuming which register contained
which argument, and then SAS/C does it this way (i.e. no difference
in function names, as would normally happen for register argument
functions).

To use the "no externals" version of the link library, you first need to
compile it. ;) Then define ASIO_NOEXTERNALS, link with the new library, and
otherwise follow the instructions for the normal link libraries. ("No
externals" means that the code does not require any external data symbols,
like DOSBase or ExecBase, making the code purer. To do this, certain
function calls takes DOSBase and ExecBase as arguments.)

To create the link libraries:

DICE: Type e.g. "lbmake asyncio s r e" and have dtmp: set to some
temporary space. This creates a regargs, smalldata version.

SAS/C: Type e.g. "dmake lib/asyncioer.lib" to get the regargs version.

Notes

Remember that all ASIO_#? symbols must be set before any AsyncIO header is
read. There is one exception: ASIO_SHARED_LIB does not need to be defined
before you include <proto/asyncio.h>.

I haven't personally tested all library versions in the makefile/lib.def.

SAS/C users still need DMake (from DICE) to compile things. I'm used to
DMake, and haven't really used SMake. ;)

The code was originally compiled with DICE. Current version is compiled
with SAS/C (as well).

Getting the code to compile with other compilers (GCC, StormC, ... ) should
be fairly easy, since I've tried to keep all compiler specific things
within #defines in a header file, and generally tried to only use normal
ANSI/C.

The DICE link libraries are created with the help of the LbMake utility in
the DICE package. The file Lib.Def contains the definitions for them.

## 1.5 AsyncIO.guide/Technical notes

Technical notes

The purpose of this chapter is to describe the inner workings of the code.
It is partly for my benefit, so that I don't need to remember all details,
and partly for your benefit, if you want to try to find bugs, or maybe use
asyncronous code in places were AsyncIO as is does not provide the needed
functionality.

The comments in the code do help, but here is the place to put everything
together, and place additional comments.

Please note that the information described herein may change, and can not
be used in normal applications that use AsyncIO! It is intended for persons
dealing with the AsyncIO code itself (or derivates thereof).

The AsyncFile structure
(More to be added...)

## 1.6 AsyncIO.guide/Technical notes/The AsyncFile structure

The AsyncFile structure

af_File (BPTR)
The DOS file handle.

af_BlockSize (ULONG)
Blocksize of the device the file resides on.

af_Handler (struct MsgPort *)
Handler port of the file system responsible for the file.

af_BytesLeft (LONG)
Number of bytes left in the buffer we are currently reading data from.

af_BufferSize (ULONG)
The size of one of the two IO buffers.

af_Buffers[ 2 ] (UBYTE *)
Pointers to the two IO buffers.

af_Packet (struct StandardPacket)
Packet structure used when communicating with the file system.

af_PacketPort (struct MsgPort)

Message port used to receive the packets sent to the file system. No
signal bit is allocated of this port; SIGB_SINGLE is used when needed.

af_CurrentBuf (ULONG)
Number of the buffer that is currently being filled (by the file system),
or, if AS_WaitPacket() just has been called, the newly arrived buffer. It
does not refer to the buffer we are currently copying data from. Can be
used to index the af_Buffers[] array.

af_SeekOffset (ULONG)
In order to keep the buffers block aligned while seeking in a read mode
file, this offset keeps track of the distance from the start of the buffer
to the position we should start reading the new data from.

af_SysBase (struct ExecBase *)
Pointer to SysBase. Only available if ASIO_NOEXTERNALS is defined.

af_DOSBase (struct DosLibrary *)
Pointer to DOSBase. Only available if ASIO_NOEXTERNALS is defined.

af_PacketPending (UBYTE)
If true, then the file system is currently filling a packet for us, and it
will eventually be waiting for us on the message port.

af_ReadMode (UBYTE)
If true, then the file is used for reading only. Otherwise the file is used
for writing only.

af_CloseFH (UBYTE)
If true, then CloseAsync() will close the file in af_File. If
OpenAsyncFromFH() is used to open the file, then this field will be false.

af_SeekPastEOF (UBYTE)
If true, then the last SeekAsync() tried to seek past EOF. SeekAsync() uses
this to allow the application to seek back to a valid position, and
continue reading from there.

af_LastRes1 (ULONG)
af_LastBytesLeft (ULONG)
These two stores the last af_BytesLeft and af_Packet.sp_Pkt.dp_Res1. This
is needed in order to be able to resume seeking after a seek past EOF.

## 1.7 AsyncIO.guide/History

History

Release 1 - 23-Mar-94

* When seeking within the current read buffer, the wrong packet would
be sent out to be filled asynchronously. Depending on the data read
from the buffer, and how fast it was read, you could end up getting
incorrect data on subsequent ReadAsync() calls.

* There was actually bufferSize*2 bytes allocated for IO buffers
instead of just bufferSize. This is fixed. So if you want the same

effective buffer size as before, you must double the value of the bufferSize argument supplied to OpenAsync().

* MEMF_PUBLIC is now specified for the IO buffers. This is in support of VM hacks such as GigaMem.

* A Seek() call had the mode and offset parameters reversed. The code worked, because both values were luckily always 0, but it was not very clean.

* Now uses a typedef for the AsyncFile structure, and enums for the open modes and seek modes.

Release 2 - 16-Feb-94

* SeekAsync() now consistently works. It was getting confused when called multiple times in a row with no intervening IO.

* WriteAsync() would produce garbage in the destination file if it had to bring up a "Disk is full" requester, and the user freed some room on the disk and selected "Retry".

Release 3 - 12-Aug-95 - Changes from now on by Magnus Holmgren

* SeekAsync() is now not unecessarely slow when doing some "kinds" of read-mode seeks (typically when seeking after some small amount of initial read). The problem was that it usually only considered the newly arrived buffer, not both buffers. This could make it discard both buffers, and restart reading, although one of the buffers already had the needed data.

Note that the kind of seeks that caused the above problem may still seem to be somewhat slow, since the code must wait for both buffers to be loaded. This cannot (easily) be avoided.

* SeekAsync() doesn't cause the read buffer to contain garbage after certain seeks any more. The problem was that ReadAsync() would read from the wrong buffer (the one currently being loaded). This made the files seem to contain garbage. This happened when the seek location was within the newly arrived buffer.

* The code package is now supplied as a link library, rather than a single source module. The internal functions labeled "AS_#?" are private and should not be called directly by the application.

* A few minor "cosmetic" changes were done, to either make the code more readable, or to make it slightly smaller.

* Include file restructured a little (a public and a private part).

* OpenAsync() now offers some new "options" (all from the AIFF code by Olaf Barthel):

* Opening a file from an already open filehandle is now possible.

* A "no externals" version may be compiled, that doesn't require any external variables to be available.

```
    * Each of the buffers will now be roughly bufferSize / 2 bytes large,
  rather than bufferSize bytes (rel 6 note: Really from Taillefers
  third release).

    * If there isn't enough memory for the requested buffer size, the
  code will try with smaller buffers (still properly "aligned")
  before giving up.

Release 4 - 13-Sep-95

    * Oops. Forgot to include the include/asyncio.h file. Maybe as well,
  since I anyway had forgot to mention a few things in it.. ;)

    * Asyncio is now also available as a shared library (to be placed in
  libs:). This means that a couple of new include files were added,
  and a one include file was split up.

  The main reason for doing this was to simplify the use of it in
  other languages (only need to "port" a few includes). I have no
  other include files than those for (DICE) C at the moment, but feel
  free to send me headers for other languages as well.

  Note: I have not yet verified that the SeekAsync function works,
  but since read and write works just fine, I see no reason why
  SeekAsync wouldn't work. ;)

  Also, I haven't really used it in programs much either (only some
  testing), so the different defines and similar to use different
  versions may be a bit clumsy to use (see above for more
  information). Feel free to send me comments on how to improve this.

    * Changed so that a non-regargs version of the link library can be
  created. To use the regargs version now, simply make sure that
  ASIO_REGARGS is defined, and link with the proper link library.

    * Changed the name of the NOEXTERNALS define to ASIO_NOEXTERNALS.
  Define this before you include <clib/asyncio_protos.h> if you want
  to use that feature. Note that you need to create a proper link
  library yourself! (With DICE, all you need to do is "LbMake asyncio
  s r e" in the Src drawer.)

    * Modified OpenAsync() a little, to work around a problem when having
  SnoopDos (with SendARexx active), MungWall and Enforcer running.
  Not an asyncio bug as such, but.. ;)

Release 6 - 10-Nov-95

    * Bumped to version 6, since there was an "official" release 3 that I
  was unaware of. However, most of the changes were in the AIFF
  source. Thus, not much was "lost" due to this. (In fact, the
  SeekAsync bugfix in the real release 3 was not correct. ;)

    * Sigh. The SeekAsync fix (to the performance problem) was buggy. :/
  I think I've got it right now.

    * asyncio.library bumed to version 37.1.
```

```
      * Cleaned up this documentation.

      * Fixed some bugs in the include files.

      * Made some fixes to the shared library.

Release 7 - 7-Jan-96

      * Files to use asyncio.library from E included.

      * Fixed yet some SeekAsync() problems. It could (still) get confused
  when called multiple times in a row with no intervening IO. I
  wonder if there are any holes left. ;)

      * Some more fixes to the shared library.

      * asyncio.library bumped to version 37.2.

      * Recompiled asyncio.library using SAS/C 6.56, and added general
  SAS/C support.

      * Cleaned up parts of this doc.

Release 8 - 28-Jan-97

      * Yet another hole in SeekAsync() (hopefully) plugged. ;) Seeking in
  a double-buffered file is tricky business it seems. Thanks to David
  Eaves for finding it, and sending me a good report about it
  (including source to reproduce it).

      * The problem was that when SeekAsync() really seeked past the end of
  the file, it didn't notice, and would try to reload the "last"
  buffer.

      * Finally finished up the ReadLineAsync() function. Only difference
  from dos.library/FGets() should be the return value. I hope it
  works properly, since I haven't tested it that much. ;)

      * Bumped the library version to 38, because of the new functions.

Release 9 - 27-Apr-97

      * Oops. The include/clib/asyncio_protos.h file wasn't quite correct,
  causing problems when compiling with SAS/C and ASIO_REGARGS
  defined. Also updated includes for StormC. Thanks to Alexander
  Kazik who sent me the needed changes for StormC.

      * Michael B. Smith pointed out that ReadLineAsync() wasn't quite
  FGets()-compatible. So he sent me his version, which was. I've now
  added his version (with some minor changes by me), and changed the
  implementation for ReadLineAsync() to first call the new function
  FGetsLenAsync(), and then skip up to the LF char, for backward
  compatibility.

      * Added PeekAsync(), to allow you to look ahead in the current buffer
  without actually "reading" anything. Useful when reading from pipes
```

to e.g. find out the file type (needed in Visage ;).

* Bumped the library version to 39, because of the new functions.

* The autoopen code for SAS/C now uses __asiolibversion rather than __oslibversion (in case of failure, __oslibversion is set to the value of __asiolibversion, and the standard __autoopenfail is called, in order to save code). The default value is 39, and any external reference to __asiolibversion will cause the autoopen code to be linked.

* Ouch. The DMakeFile/lib.def files contained some laws. As a result, the DICE linker libraries (in dlib/) were not updated at all. Now there should be fresh libraries for both DICE and SAS/C.

* Fixed a couple of typos in the autodocs, including the problem with the enum arguments specified as (U)BYTE, when they need to be LONG (some compilers assumes enums to be longs, unless told otherwise).

Version 39.1 - 27-Jul-97

* Ouch. The "David Eaves" bug fix in SeekAsync() was wrong. Not only did it not work as intended, the intention was all wrong. Should be fixed now, I hope. ;)

* Cleaned up the messy asyncio.doc file, and moved all non-autodoc stuff to this AmigaGuide file. Also added some technical notes (not complete yet).

* Changed to "real" version numbers for the release archive.

* Referencing __asiolibversion alone isn't enough to drag in autoinit code any more. Should also make it easier for user programs to set their own version now. ;)

* Actually compiled a SAS/C version with ASIO_NOEXTERNALS defined. PeekAsync() and FGetsLenAsync() needed a tweak because of this. The generated link library seems to work as well.

* Tweaked the DMakeFile (and fixed a few quirks), to create the object files in various sub-drawers.

* FGets(Len)Async() would never fill the entire read buffer, even when it could (we're talking about one byte not used here! ;).

* Tried to make the usage instructions a bit clearer.

Version 39.2 - 2-Sep-97

* ReadLineAsync() didn't detect end-of-file if the last line wasn't terminated by a line feed.

* ReadLineAsync() could write one byte too much to the read buffer.

* Tried to make SeekAsync() allow seeks after a seek past EOF (in order to be more DOS-compatible). I hope I didn't break anything. ;)

```
     * Old bug: ReadCharAsync() and WriteCharAsync() could sometimes
  manage to do their job successfully when they really should fail
  (and return -1).
```

## 1.8 AsyncIO.guide/Authors

```
Authors

Martin Taillefer wrote the original code.

Magnus Holmgren (that's me ;) "took over" the development in 1995, since I
had found a couple of bugs in SeekAsync(). I contacted Martin Taillefer
about it, and asked if I could release new versions of AsyncIO, and he
didn't complain.

Olaf Barthel provided inspiration for some of the changes in release 3
(one or two of them most likely came from Martin Taillefers third release,
which I was unaware of at the time).

Michael B. Smith wrote the FGetsAsync() and FGetsLenAsync() functions,
which I added after some (really) minor changes.

Most of the changes I have done are marked in the sources with a comment
that starts with "MH:".

If you want to contact me about something, feel free to send a message to:

  cmh@lls.se        (internet)
  "Magnus Holmgren", 2:203/512.10   (fidonet)

Please note that I don't poll the fidonet address that often. ;)
```