

Operating System: High Level Design Document

Luke A. Guest

27th November 2007

0.1 Introduction

Everybody has their own favourite OS that they've come across in the past and that has now for some reason or another died out¹ and they want an OS that brings this ease that they found in their particular OS back. For some it might be Apollo, but for me, it's AmigaOS. I have waited a long time for a replacement that has never come, I even did a small bit of work on AROS, but I want something new now, something that takes ideas from the AmigaOS and builds on it to provide something modern with no legacy baggage.

0.2 Requirements

I'm not going to go into the whys or wherefores of developing a new OS in this document, only describe how *I think* an OS should behave. I will also document areas/concepts that we can pilfer from other existing/past OSes. Let us start with the basics of what an OS should be capable of handling:

- It must be portable across architectures.
- It must provide memory protection to tasks.
- It must utilise multiple CPU's if they exist.
- It must provide a programming language neutral API.
- It must be scalable.
- It must be fault tolerant.
- It must be consistent.

Ok, so there's nothing new in this list yet there's no OS that does this the way I want it to, yet.

0.2.1 Portability

With wearable and portable computers (digital music players, phones, handheld consoles, etc.) becoming more prevalent in the world, there is a need to provide an OS that is portable across platforms. A phone may use an ARM or a MIPS CPU, for example; these are completely different to the CPU's we use on our desktop machines. Having an OS that can be moved with the minimum of fuss from 1 platform to the next is vitally important.

¹Usually due to being bought out or the company went bankrupt due to severely bad management!

0.2.2 Memory protection

Old OSes from the 80's didn't have memory protection, what this means is that if a task wanted to it could write to another task's memory and this in turn could crash the whole system. We have OSes that protect each task from each other as standard on desktop machines, so we should have this too².

0.2.3 Multiple CPU's

If a particular hardware platform has multiple CPU's, they should be utilised transparently by the OS. Many desktop machines now have more than 1 CPU inside, most portable machines (phones, etc.) have only 1. By using these resources the OS will become more efficient³.

0.2.4 Programming language neutral

Every programmer has their own favourite programming language, an OS should strive to be programmable in any of these languages.

0.2.5 Scalable

We have already mentioned portable computers and desktops, these are 2 extremes. Why should I need to install a different OS on 1 device only to use a different OS on another? They should use the same OS!

It would be possible to provide some sort of profile which would allow an OS to be built up in layers. For example, the base layer could be everything needed that is common to all platforms, then build upon this to add functionality for portable devices, build on this for a server, again another layer could be used for desktops, etc⁴.

Obviously, there are some components that aren't needed by some profiles, i.e. why would a server need a component that can make a phone call or store email addresses? A desktop might make use of these features though, e.g. Skype, Personal Information Management (PIM), etc.

Other certain OSes force the user to buy another OS completely for a portable device, another for a desktop device and another still for a server! Why? It's not necessary!

0.2.6 Fault Tolerance

I don't want my whole OS to crash if 1 part of it crashes. I want the crashed component to start back up again (if it needs to) without causing me any problems.

²I'm not too sure about this paragraph as it would be possible to abstract out the memory protection hardware such that it would be possible to have it or not (see AROS' mmu.resource).

³Within the constraints of Ahmdal's Law of course.

⁴These layers need to be clearly defined.

Say the UI component crashes, this should restart but it shouldn't force me to log back in again, this also applies to a video card driver or the network driver.

0.2.7 Consistency

This is the biggest problem with the various Unix derived OSes out there. There are so many different GUI's, and different applications use different GUI's, what this does is provide an inconsistent feel for the applications being used on an OS. This also can cause slight problems in their usage as well due to the fact that one GUI doesn't work the same way as another.

An example in this area are the differences in KDE and GNOME applications. For example an application from KDE running under GNOME may use key bindings that are considered standard bindings for GNOME and vice versa, this means the user either has to know what key bindings are used in either GUI or have key bindings that don't work for an application. Having a consistent GUI will remove this problem.

0.3 General use

0.3.1 Booting

When I start the machine I want it to boot fast, I don't want to be hanging around waiting, it needs to be as instantaneous as possible. I should then be presented with a GUI login⁵ which will allow me to log into the machine.

0.3.2 GUI

When I've logged in, I will be presented with some sort of standard UI for managing the various components on my machine. I should be able to drag and drop icons around, copy them about to different locations on disk (or ram disk), the usual GUI fare.

Desktops

OSes have always had 1 desktop which gets cluttered up very easily, under Linux I use 4 virtual desktops on a day to day basis. Mac OS X (even Linux with CompViz/Beryl/etc.) has included a method (*Expose*) of uncluttering the display to make switching windows easier, even Vista has this ability now. Apple has now incorporated *Spaces* into Mac OS X which is the equivalent of the virtual desktop.

⁵This could apply to handheld devices as well if the added security is wanted.

Screens

AmigaOS had the concept of screens which you could use for this. I could open an application window on the Workbench public screen or open it on it's own. I could drag the screen down to see what application was running behind, I could also drag down the screen behind that one to see what was behind that, and so on. Each screen *could* be running at a different resolution to the others and they'd still display correctly.

All screens could be either public or private (set by the author of the program) which meant that if an application allowed it's windows to open on a public screen a requester could be displayed to allow the user to select which one to open them on.

I intend to incorporate the concept of screens.

Sending objects to applications

The AmigaOS also had something called AppIcons and AppMenus, these allowed the user to drop an icon onto a menu or an icon on the Workbench screen and basically open that file with the application that was already opened. Would we need something similar?

0.3.3 Menus

Multi-click

Another feature that the AmigaOS had was the ability to click multiple times in a menu before letting go of the menu button⁶; this was patented by Commodore at the time and nobody else could use this, this patent⁷ has since expired. Another reason for using this is RSI, current OSes force you to traverse multiple levels of menu just for 1 mouse click! Try doing that multiple times, your arm/fingers is/are going to start to hurt.

Structure

This brings me onto another feature of the AmigaOS menus, they were limited to the following structure:

1. Menu bar
2. Menu
3. Sub-menu

⁶That was the right button, by the way.

⁷Not that software patents make sense! Or that they apply to me as I'm in the UK.

You had a menu bar in the top of the screen when you pressed the menu button, this showed the menu items available for the current application, each menu item had an associated menu and in the menu you could have 1 (yes, only 1) extra level of menu (the sub-menu). This was an excellent design *feature* as you could never have menus that went from 1 side of the screen to the other only to build up tons of sub-menus, think of your arms (RSI!!).

This structure can be a problem though as other application are ported across. I do remember it being limiting because application developers from other platforms were used to being unlimited in menu use and thus, used many sub-menus.

Spider Menu (Can I copyright/trademark this?)

The menu that is used in Alias|Wavefront's Maya 3D package is interesting, this is a pop-up menu in which the user needs to move the mouse an equal distance to be able to select any item in the menu. This is also something to think about as far as RSI is concerned⁸.

By combining these features, I think this new OS could have something quite unique!

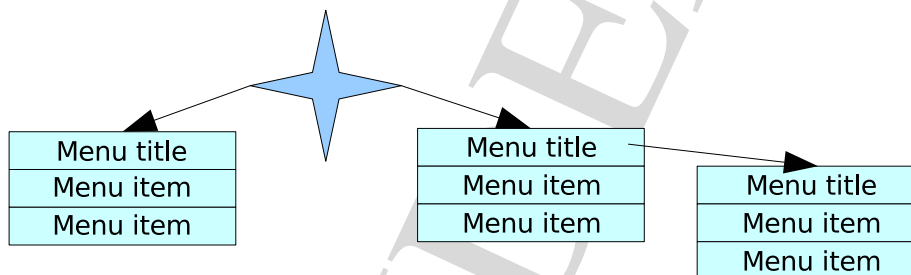


Figure 1: Spider Menu Concept

0.3.4 Objects

Unix made everybody think that everything is a file, which became problematic later on when they tried to incorporate networking, they ended up making network connections (sockets) files as well. This is the wrong way to think, everything is an object and every object has a particular type. As an example, consider that I want to take a movie *object* and burn it to VCD.

I would have the movie object on disk and I'd need to convert it to an MPEG object before burning. The size of the picture may be wrong, so I would have to add in some filter to resize it, and then burn the object to disk. If I was working under Linux, I'd have a ton of different programs each that do similar things but don't all work together or on every type of movie object. These programs all have

⁸An interesting example (although, not a popup menu) can be found at [this site](#).

different command line options, and they have lots of them and they're not easy for the average user to work out.

So, essentially, I would want to take the movie object and connect a decoder to it, then connect a filter to resize it, if the picture was grainy, I could then connect another filter to it to clean it up, then on the back end I'd connect up the VCD encoder and finally connect up the burner object to actually handle the burning of the data to CDR. In a way, it's like the pipelining of Unix shells, but this would be visual in nature.

0.3.5 The Object System

All OSes have filesystems of some sort, as you can probably tell by now, this OS will be object oriented, therefore *files* won't exist! What we will have is a collection of objects on disk, a folder is a storage container object, movies, pictures, audio are media objects.

We can take some ideas from BeOS here⁹ in that we should provide a means to annotate objects on disk with attributes and also provide a means for querying the object system. We could use a limited form of SQL for *power users* and a GUI query interface for people that want life made nice and easy.

Installing applications

Why do OSes have separate applications for installing applications or components? Why do we need one? We don't! Why can't we just have some sort of package object which we can just drag across to the object system? To uninstall this package we just delete it.

Now, what if other applications use this package? We would therefore require some sort of reference counting mechanism which keeps track of which other packages use this one. When all references have been removed, it is then safe to remove this one.

If we drag over a package which requires the installation of another, this should be handled at this point, i.e. download the package to the object system¹⁰.

This is somewhat like resource and data forks in classic Mac OS and also a bit like the filesystem in RiscOS which had containers, you'd hold down a key and double click to get inside it, at which point you could see the files contained within.

Importing & Exporting objects

This is a problem what will affect moving files from other filesystems into the Object System and vice-versa. This also affects downloading files from the internet.

⁹ Practical File System Design with the Be File System

¹⁰I'm beginning to see why Dave Haynie coined the term *Object Sea* to describe his object oriented file system concept. It's nicer than Object System, can we use it? Probably not.

As a file is being brought into the system, it's type needs to be identified and then it needs to be stored as that object type on the Object System. So, we need another type of object an importer. An exporter object would handle the reverse, transforming an object to it's known format as a file.

Persistence

Would it be possible to utilise persistence? I don't really know how I'd go about implementing something like this as it's extremely difficult to visualise how it would work. The idea of never having to save data is a nice one though.

0.4 Distribution

No, this doesn't mean an OS distribution like the different GNU/Linux OSes, but rather distributing the load of a running application. For example, say I'm running an application that does a lot of heavy processing and I'm connected to a network of machines all running this OS, why shouldn't worker threads migrate to other machines to do some work?

This reminds me of a demo that QNX had of Quake III running on one machine and then they dragged the window across the network to another machine¹¹. This idea can be extended to moving from 1 machine to another, if you know you're going to move to the living room from the office, why shut everything down and start again? Why not just shift the session across machines?

0.5 Servers, Libraries, Classes & versions

On GNU/Linux systems, libraries are changed over time, their contents can change and thus, a particular application would need to be built from source again. It's not possible to use an older binary only program with a newer library. You could copy the older libs across (which is what games company's have done before) but that makes your installation bigger. That is, if you can still build the application with the old library using the compilers; you may need to use older compilers which may not compile themselves using newer compilers.

This is another area where the AmigaOS excelled, a library had a number of offsets in a table which were the entry points to the functions in the library. If a call became obsolete the version number was changed and the function removed. The newer library could co-exist on disk alongside the older one (obviously not in the same place due to filename conflicts).

Now, this is a temporary measure while the application is updated to the newer library, but what if it's not updated anymore? With the Linux solution, it's a pain, with the AmigaOS solution, you can just copy in the old library and still use it.

¹¹I've never actually seen this demo, only heard of it.

The OS will be built up out of a collection of servers, libraries (shared and static) & classes (like a library). Servers will be the active parts of the OS, libraries and classes would be the passive parts. All combine together to form a whole.

0.6 Scripting

The OS should provide a means for scripting objects. This allows users to create interpreted applications using the existing objects. Applications could use scripting to enhance itself or to allow applications to work with each other without actually knowing anything about each other¹².

0.7 Shells

This OS should be mainly graphical in nature, but for power users there should exist a command shell, this could be a native shell or BASH or some other shell ported from any other platform if the user so desires. Shells enable the user to get things done quickly without wading through lots of windows.

0.7.1 Command names

We shouldn't have small names for commands, we can have long names, just as long as the name doesn't get too long! Instead of *cp* we should have *Copy*, etc.

The shell should have automatic completion so that the command names can be filled in automatically.

0.8 Devices

The various Unix like OSes create a whole load of device nodes, a lot of which don't even exist! This OS should only list devices that are available to the user. The drivers should register the device names in the device namespace so that they can be accessed if the user has the correct privileges. If a user doesn't have privileges to access the device, then the user shouldn't even see it, right?

0.9 Conclusion

There is nothing new presented in this document, I don't think there is anything new anymore. These concepts just hasn't been put together into 1 OS before; or if they have, it was some research project that has since been shelved.

¹²Much like the **REXX** language.

0.10 Acknowledgements

I got a bit hacked off with typing TM after every trademarked name, so I thought I'd just gather them up and dump them here:

1. AmigaOSTM & WorkbenchTM are registered trademarks of Amiga Inc.
2. BeOSTM is a registered trademark of Palm.
3. Mac OSTM, Mac OS XTM is probably a registered trademark of Apple Inc.
4. UnixTM is a registered trademark of somebody, SCO?.
5. MayaTM is a registered trademark of Alias|Wavefront.
6. RiscOSTM is a registered trademark of RiscOS Ltd.
7. VistaTM is a registered trademark of Microsoft Corp.
8. REXXTM is a registered trademark of IBM.