osFree Whitepaper

About osFree

osFree is a free and open-source operating system project based on the L4 microkernel aiming to be binary compatible with OS/2 (ia32). Also, a parallel coexistence of several “alien” OS API's is possible.

The osFree project is aimed at producing an operating system with support of OS/2 compatible personality as the base personality. We have used the following rules in our research and development:

- Usable
- Extensible
- Open
- Easy

As a result, we have the following considerations:

- CUA for usability (see IBM SAA CUA)
- OOP principles and modularity in both Microkernel interfaces and SOM, together with binary compatability, for extensibility
- Documentation and Open-source for openness
- All of the above for ease of use

Lightweight

OS/2 is one of most lightweight 32-bit OS. osFree also will be as lightweight as possible. We don't want to request 1 Gb minimum for work. We want to make it work on as minimal hardware as it possible. This will allow to use osFree in embeded area.

Open

osFree components come under open licenses like BSD and (L)GPL licenses. And we try to document interfaces as much as possible. So you are free to extend it as you wish.

Object-oriented

osFree tries to extend object-oriented design of desktop to other parts of the system, using OOP principles. We use SOM as a base object model, as designed by IBM for OS/2 desktop and other parts of the system. Also, we want to introduce CPI+, GPI+, PM+, i.e., we plan to make the object-oriented wrappers for current CPI [1], GPI [2] and PM services. Access to low-level services of the

kernel also planned to be represented as SOM objects.

Why re-implement OS/2?

OS/2 has one of the most stable, robust and high-performance kernels. Written in assembly language, it is highly-optimized and uses all i386 architecture features very extensively. It's modular design allows to easy replace components with more featured/less resource-eating/cut off GUI, or customize system to fit user preferences. It is highly configurable. We like its compact and clean API, it's easy to use and intuitive powerful true object-oriented user interface. It's uses one of the best general purpose scripting languages named REXX as operating system scripting service with API available to any application. OS/2 was advertised by IBM as “DOS better than DOS and Windows better than Windows”. It is true – it's VDM was the best ever existing. And not only DOS/Windows. It had Java and XFree86 support very powerful too. So, we started loving OS/2 as powerful integration platform on top of single desktop. It has been used by marginals and non-conformists for years and always had its own way. We want to continue following this way :) We can sleep peacefully knowing that it is not popular between hackers and virus writers, they like mainstream.. But we can't stay this way – starting at December 2006, IBM management decided to kill OS/2 and OS/2 community left with aging system without kernel sources and with fading driver support. The driver support becomes more bad each time, and almost all new drivers are ports from Linux. Petitions to IBM were failed as well. Nevertheless, we want to continue our experience with OS/2. The most necessary task is kernel rewriting. Here, it must be noted that OS/2 still has 32-bit kernel. Existing kernel, even if having sources, is not portable to other architectures, the main of which are ARM and x86_64. Modern software is bloating quickly, so we will encounter the 32-bits OS limitations, the main of which is impossibility to use more than 4 GB of RAM. Now common web browsers and offices eat up gigabytes of RAM easily. Also. one of the main RAM consumers are VM's. And we should not forget about positioning of OS/2 as Integration platform (TM), including VM's too. Because of that, OS/2 needs a new kernel. We've been always curious regarding IBM's experiments with OS/2 on top of microkernel. We read a redbook about it. So, we meet with enthusiasm the suggestion to use L4 as a base. So this project was started.

Why not migrate to other modern OS?

IBM, Netlabs and other companies tries to move users to another OS like GNU/Linux, BSD, Windows and others. We agree that TCO of current OS/2 become bigger and bigger for home users. In servers area OS/2 also become obsolete (but still stable and mature). But we like approaches of IBM according design of OS. We like it **designed**, not only developed. It is easy to use. It's API is small and clean. We want to continue work and program with OS/2.

Compatible

osFree is planned to be compatible with most of current OS/2 API. But not drivers. OS/2 drivers become more and more obsolete, but we want to have modern hardware support. In current design osFree can be hosted on most of actual kernels like L4, Linux, Windows, etc. As result we can reuse existent drivers as is.

API compatibility allow us to have still clean and small API and reuse existent applications. We have no plan for full support of 16-bit part of OS/2 because not so many applications which true 16-bit. For

most mixed 16/32 applications we will provide on-the-fly patching of 16-bit calls to true 32-bit calls. As result we will have true 32-bit applications (after many years of mixed 16/32 applications).

It is possible to embed the mixed 16/32 to pure 32-bit apps converter in the executable file loader. So, the pure 32-bit application will always be in memory. It is even possible to save the conversion result to disk, which could be started again without the need to convert it again (just like Just-in-time-compilation in Java).

We have a plan for limited support of DOS and Win16 personalities (for historic reasons). But we don't limit you to add more personalities.

General design of osFree

We propose to reuse some or most of OS/2 PPC design with some reformulation. Since OS/2 support is our primary target, of course, we propose to reuse most of OS/2 technologies.

In general, we'll use L4 microkernel as a core of the system. On top of L4 we plan to implement the following personalities:

- Neutral personality
- OS/2 personality
- Linux personality
- MVM personality
    - DOS personality
    - Win16 personality
- Win32 personality

Neutral personality or **Personality-neutral services** is the real OS API. It is a set of servers and libraries giving away various services. All other personalities need to work via Neutral personality. Most probably, many Neutral services API's will be reused almost as is (in the form of simple wrappers). The Neutral Personality API can be compared with Native NT API – they are almost functionally equivalent.

OS/2 personality aimed to provide partial or full set of OS/2 API. At the first stage we want to implement core 32-bit API. In most cases OS/2 personality calls are planned to be just forwarders of calls to Neutral personality.

Linux personality aimed to provide full functionality of Linux. Linux is one of the current mainstream OSes. It provides lots of development tools, libraries and applications, and we want to use Linux as one of the main development platforms. We plan to reuse the L4Linux project for this. If everything will work fine, we'll just recompile the L4Linux project and reuse it.

MVM personality aimed to provide the functionality of an environment for running multiple Virtual Machines with unmodified OS'es. DOS and Win16 personalities will be based on MVM personality with support of Virtual Device Drivers

DOS personality aimed to provide the functionality of DOS. DOS was supported by the original OS/2 and is still used by many people. For us, this direction of development is very perspective, though it has less priority than OS/2 personality development. So, we don't want to lose too much forces on this goal, so we'll most probably reuse other projects here, like QEMU, VBOX, DosBox, DosEmu, FreeDos etc.

Win16 personality aimed to provide partial or full functionality of Win16 up to version Windows ME. Win16 personality actually work under MVM prsonality. Seamless desktop integration are planned.

Win32 personality aimed to provide partial or full functionality of Win32. Windows is also one of the mainstream OSes and we can't ignore its existence. **If** it will be possible to para-virtualize ReactOS then we'll also provide Win32 functionality (there is an idea of implementing a HAL, working above l4env/l4re). Otherwise Win32 support will come via WINE project.

Of course you are free to add another personality.

At present time we have closed view about file systems support and boot process. We are reusing OS/2 concept of IFS'es. Most notable differences from OS/2 PC are the absence of MiniFSD (like it is in OS/2 PPC) and 32-bit IFS main driver. For more information about the boot process look at Boot process guides and references.

For general development guidelines see Developer Reference.

Why use a microkernel

- microkernel can serve as a base for different OS API's, implemented on top of it. These API's can be executed concurrently, having their common part as small as the microkernel is. This allows for parallel (non-layered) implementation of concurrent API's.
- OS API over a microkernel is implemented in user level, leaving only microkernel in privileged kernel mode. Such OS components as thread schedulers, memory managers, swapper task, even direct hardware access and interrupt handling, are moved to userlevel too.
- this allows implementing stable system with rock stable small kernel and less stable user level components, which (as it shows Minix design, for example) can be restarted (even automatically) after it has failed. The microkernel itself can be well-debugged, and even, formally verified with mathematical methods as error-proof. So that, the errors in microkernel are very unlikely, and other OS components could be not so critical.
- this also allows to use ordinary development techniques for drivers as driver in microkernel system does not radically differ from other applications.
- microkernel architecture improves dependability of the system, which means that dependencies between system components are well-defined. Servers executing over a microkernel, interact only through well-defined interfaces and incapsulate their internals, which closely resembles the object-oriented approach.
- microkernel system improves isolation of errors inside system components, because servers are executing in separate address spaces.
- microkernel abstracts hardware from usermode servers, which allows implementation of portable operating systems, where all usermode system part remains unchanged in its source form, requiring only its recompilation.

Why L4?

- L4 is a second-generation microkernel, which improves overall system performance significantly. This can be demonstrated by L4Linux as an example. L4Linux is essentially a port of Linux to a new 'L4' architecture. Its hardware access code was changed, so it is accessed not directly, but through L4 mechanisms. The Linux kernel performance can be tested by benchmarks, which shows a loss of about 2% of performance of native Linux kernel. We were tried to start L4Linux on a real machine and did not noticed the difference. This shows that L4

demonstrated excellent performance with very small loss.
- It's minimality and moving all policies outside the kernel, leaving inside it only minimal set of mechanisms, makes it almost universal and makes it possible to implement almost any desired API.
- We're must not reinvent the wheel: it has all required mechanisms, which we need
- A set of general-purpose services is already implemented for it, so we're not left with bare kernel
- L4Linux can be used as Linux personality base. It is yet at development stage but nevertheless, almost all is working. On our laptop, only PCMCIA modem was not working (it could be fixed, though, by doing ioremap to other addresses – the original addresses used by the driver, are used in L4-based system). Wi-fi, bluetooth stack, USB stack, filesystems, CD writing – all these were working! And one more inconvenience: as the video is working over the native graphical L4 console, which at present is working via VESA mode (only selected video cards are supported with acceleration), so the video support is way limited.
- Device Driver Environment (DDE) could be used in future as a drivers framework. It ports Linux (DDE/Linux) and FreeBSD (DDE/FreeBSD) drivers to L4 userlevel. So, the big codebase of Linux drivers can be reused in future. In contrast with windows ones, they are available in source form and could be ported to user-level.

Contributing

We have a lot of things still to do, so any help is welcome. Not only for code development, but also documentation writing, web page maintenance, distribution maintenance, and much more. See the Project Roadmap for more information about where we intend to go.

We are also looking for developers. For newbies, we have a number of small tasks. If you're an experienced developer, there are plenty of complex tasks awaiting your talents! Check out the development page for more information about developing for us, and look at our licensing.

osFree IRC channel is #osFree at EFnet and eCSnet.

1)
OS/2 kernel Control Program Interface
2)
Graphics Programming Interface