= DragonFly BSD =

DragonFly BSD is a free and open source Unix @-@ like operating system created as a fork of FreeBSD 4 @.@ 8 . Matthew Dillon , an Amiga developer in the late 1980s and early 1990s and a FreeBSD developer between 1994 and 2003 , began work on DragonFly BSD in June 2003 and announced it on the FreeBSD mailing lists on 16 July 2003 .

Dillon started DragonFly in the belief that the methods and techniques being adopted for threading and symmetric multiprocessing in FreeBSD 5 would lead to poor system performance and cause maintenance difficulties . He sought to correct these suspected problems within the FreeBSD project . Due to ongoing conflicts with other FreeBSD developers over the implementation of his ideas , his ability to directly change the FreeBSD codebase was eventually revoked . Despite this , the DragonFly BSD and FreeBSD projects still work together contributing bug fixes , driver updates , and other system improvements to each other .

Intended to be the logical continuation of the FreeBSD 4.x series , DragonFly 's development has diverged significantly from FreeBSD 's , including a new Light Weight Kernel Threads (LWKT) implementation , a lightweight ports / messaging system , and feature @-@ rich HAMMER file system . Many concepts planned for DragonFly were inspired by the AmigaOS operating system .

= = System design = =

= = = Kernel = = =

DragonFly 's kernel is a hybrid , containing features of both monolithic and microkernels , such as the message passing capability of microkernels enabling larger portions of the OS to benefit from protected memory , as well as retaining the speed of monolithic kernels for certain critical tasks . The messaging subsystem being developed is similar to those found in microkernels such as Mach , though it is less complex by design . DragonFly 's messaging subsystem has the ability to act in either a synchronous or asynchronous fashion , and attempts to use this capability to achieve the best performance possible in any given situation .

According to developer Matthew Dillon , progress is being made to provide both device input / output (I / O) and virtual file system (VFS) messaging capabilities that will enable the remainder of the project goals to be met . The new infrastructure will allow many parts of the kernel to be migrated out into userspace ; here they will be more easily debugged as they will be smaller , isolated programs , instead of being small parts entwined in a larger chunk of code . Additionally , the migration of select kernel code into userspace has the benefit of making the system more robust ; if a userspace driver crashes , it will not crash the kernel .

System calls are being split into userland and kernel versions and being encapsulated into messages . This will help reduce the size and complexity of the kernel by moving variants of standard system calls into a userland compatibility layer , and help maintain forwards and backwards compatibility between DragonFly versions . Linux and other Unix @-@ like OS compatibility code is being migrated out similarly .

= = = = Threading = = = =

As support for multiple processor architectures complicates symmetric multiprocessing (SMP) support , DragonFly BSD now limits its support to the x86 @-@ 64 platform . DragonFly originally ran on the x86 architecture , however as of version 4 @.@ 0 it is no longer supported . Since version 1 @.@ 10 , DragonFly supports 1 : 1 userland threading (one kernel thread per userland thread) , which is regarded as a relatively simple solution that is also easy to maintain . Inherited from FreeBSD , DragonFly also supports multi @-@ threading .

In DragonFly , each CPU has its own thread scheduler . Upon creation , threads are assigned to processors and are never preemptively switched from one processor to another ; they are only

migrated by the passing of an inter @-@ processor interrupt (IPI) message between the CPUs involved . Inter @-@ processor thread scheduling is also accomplished by sending asynchronous IPI messages . One advantage to this clean compartmentalization of the threading subsystem is that the processors ' on @-@ board caches in Symmetric Multiprocessor Systems do not contain duplicated data , allowing for higher performance by giving each processor in the system the ability to use its own cache to store different things to work on .

The LWKT subsystem is being employed to partition work among multiple kernel threads (for example in the networking code there is one thread per protocol per processor) , reducing competition by removing the need to share certain resources among various kernel tasks .

= = = = Shared resources protection = = = =

In order to run safely on multiprocessor machines , access to shared resources (like files , data structures) must be serialized so that threads or processes do not attempt to modify the same resource at the same time . In order to prevent multiple threads from accessing or modifying a shared resource simultaneously , DragonFly employs critical sections , and serializing tokens to prevent concurrent access . While both Linux and FreeBSD 5 employ fine @-@ grained mutex models to achieve higher performance on multiprocessor systems , DragonFly does not . Until recently , DragonFly also employed spls , but these were replaced with critical sections .

Much of the system 's core , including the LWKT subsystem , the IPI messaging subsystem and the new kernel memory allocator , are lockless , meaning that they work without using mutexes , with each process operating on a single CPU . Critical sections are used to protect against local interrupts , individually for each CPU , guaranteeing that a thread currently being executed will not be preempted .

Serializing tokens are used to prevent concurrent accesses from other CPUs and may be held simultaneously by multiple threads , ensuring that only one of those threads is running at any given time . Blocked or sleeping threads therefore do not prevent other threads from accessing the shared resource unlike a thread that is holding a mutex . Among other things , the use of serializing tokens prevents many of the situations that could result in deadlocks and priority inversions when using mutexes , as well as greatly simplifying the design and implementation of a many @-@ step procedure that would require a resource to be shared among multiple threads . The serializing token code is evolving into something quite similar to the " Read @-@ copy @-@ update " feature now available in Linux . Unlike Linux 's current RCU implementation , DragonFly 's is being implemented such that only processors competing for the same token are affected rather than all processors in the computer .

DragonFly switched to multiprocessor safe slab allocator , which requires neither mutexes nor blocking operations for memory assignment tasks . It was eventually ported into standard C library in the userland , where it replaced FreeBSD 's malloc implementation .

= = = = Virtual kernel = = = =

Since release 1 @.@ 8 DragonFly has a virtualization mechanism similar to UML , allowing a user to run another kernel in the userland . The virtual kernel (vkernel) is run in completely isolated environment with emulated network and storage interfaces , thus simplifying testing kernel subsystems and clustering features .

The vkernel has two important differences from the real kernel : it lacks many routines for dealing with the low @-@ level hardware management and it uses C standard library (libc) functions in place of in @-@ kernel implementations wherever possible . As both real and virtual kernel are compiled from the same code base , this effectively means that platform @-@ dependent routines and re @-@ implementations of libc functions are clearly separated in a source tree .

The virtualized platform vkernel runs on is built on top of high @-@ level abstractions provided by the real kernel . These abstractions include the kqueue @-@ based timer , the console (mapped to the virtual terminal where vkernel is executed) , the disk image and virtual kernel Ethernet device (

VKE) , tunneling all packets to the host 's tap interface .

= = = Package management = = =

Third @-@ party software is available on DragonFly as binary packages via pkgng or from a native ports collection ? DPorts .
DragonFly originally used the FreeBSD Ports collection as its official package management system , but starting with the 1 @.@ 4 release switched to NetBSD 's pkgsrc system , which was perceived as a way of lessening the amount of work needed for third @-@ party software availability . Eventually , maintaining compatibility with pkgsrc proved to require more effort than was initially anticipated , so the project created DPorts , an overlay on top of the FreeBSD Ports collection .

= = = CARP support = = =

The initial implementation of Common Address Redundancy Protocol (commonly referred to as CARP) was finished in March 2007 . As of 2011 , CARP support is integrated into DragonFly BSD .

= = = HAMMER file system = = =

Alongside the Unix File System , which is typically the default file system on BSDs , DragonFly BSD supports HAMMER file system . It was developed specifically for DragonFly BSD to provide a feature @-@ rich yet better designed analogue of the increasingly popular ZFS . HAMMER supports configurable file system history , snapshots , checksumming , data deduplication and other features typical for file systems of its kind .
The next generation of HAMMER file system (HAMMER2) is being developed by Dillon . DragonFly BSD 3 @.@ 8 @.@ 0 was the first released to include support for HAMMER2 , though it is declared as not ready for general use in release notes .

= = = devfs = = =

In 2007 DragonFly BSD received a new device file system (devfs) , which dynamically adds and removes device nodes , allows accessing devices by connection paths , recognises drives by serial numbers and removes the need for pre @-@ populated / dev file system hierarchy . It was implemented as a Google Summer of Code 2009 project .

= = = Application snapshots = = =

DragonFly BSD supports Amiga @-@ style resident applications feature : it takes a snapshot of a large , dynamically linked program 's virtual memory space after loading , allowing future instances of the program to start much more quickly than it otherwise would have . This replaces the prelinking capability that was being worked on earlier in the project 's history , as the resident support is much more efficient . Large programs like those found in KDE Software Compilation with many shared libraries will benefit the most from this support .

= = Development and distribution = =

As with FreeBSD and OpenBSD , the developers of DragonFly BSD are slowly replacing K & R style C code with more modern , ANSI equivalents . Similar to other operating systems , DragonFly 's version of the GNU Compiler Collection has an enhancement called the Stack @-@ Smashing Protector (ProPolice) enabled by default , providing some additional protection against buffer overflow based attacks . It should be noted that as of 23 July 2005 , the kernel is no longer built with this protection by default .
Being a derivative of FreeBSD , DragonFly has inherited an easy @-@ to @-@ use integrated build

system that can rebuild the entire base system from source with only a few commands . The DragonFly developers use the Git version control system to manage changes to the DragonFly source code . Unlike its parent FreeBSD , DragonFly has both stable and unstable releases in a single source tree , due to a smaller developer base .

Like the other BSD kernels (and those of most modern operating systems) , DragonFly employs a built @-@ in kernel debugger to help the developers find kernel bugs . Furthermore , as of October 2004 , a debug kernel , which makes bug reports more useful for tracking down kernel @-@ related problems , is installed by default , at the expense of a relatively small quantity of disk space . When a new kernel is installed , the backup copy of the previous kernel and its modules are stripped of their debugging symbols to further minimize disk space usage .

= = = Distribution media = = =

The operating system is distributed as a Live CD and Live USB (full X11 flavour available) that boots into a complete DragonFly system . It includes the base system and a complete set of manual pages , and may include source code and useful packages in future versions . The advantage of this is that with a single CD you can install the software onto a computer , use a full set of tools to repair a damaged installation , or demonstrate the capabilities of the system without installing it . Daily snapshots are available from the master site for those who want to install the most recent versions of DragonFly without building from source .

Like the other free open source BSDs , DragonFly is distributed under the terms of the modern version of the BSD license .

= = = Release history = = =