Disk File Monitor User Guide

Version 0.15.1

October 7, 2009

Steve Dobbelstein

Table of Contents

Overview

The Disk File Monitor (DFM) monitors I/O requests going to the physical disks and maps the requests to the files for which they were issued. DFM doesn't monitor all the file system operations on a file (read, write, create, delete, etc.). It only monitors the I/O that generates physical disk activity. A file may be read and written a lot but generate no physical disk activity if all of the reads and writes hit the cache. Knowing which file usages generate the most physical I/O can be useful to system administrators. One way they can use this information is in determining which files are good candidates for being moved to faster storage.

DFM uses kprobes to hook into the kernel code paths that generate physical I/O. It captures statistics about the I/O requests and puts them in the debugfs file system where they are gathered by a user-space program.

Root Privileges

As mentioned above, DFM uses kprobes. Kprobes are implemented as kernel modules. Only root is allowed to load kernel modules. Therefore you must have root privileges to run DFM.

Caveat Emptor

Although the utility has been tested in a variety of disk usage scenarios and on several Linux distributions, there is still an inherent risk for bugs. Bugs in kernel modules can cause severe system failure from recoverable oopses to kernel panics and system hangs. Make sure you run DFM on a system that can endure a system failure.

Warning! Opening the kernel may void your warranty. Some Linux distributions will not provide support for a system that has kernel modules loaded that were not approved by the distribution. Only run DFM on a system where you are willing to forfeit support from the distribution.

Building

Untar the source file.

```
woody:~ # tar xzf dfm-0.15.1.tgz
woody:~ #
```

Change to the source directory and run `make` to build the kernel module.

```
woody:~ # cd dfm-0.15.1
woody:~/dfm-0.15.1 # make
make -C /lib/modules/2.6.16.60-0.21-smp/build M=/root/dfm-0.15.1
KCPPFLAGS="-DHAVE_INODE_GENERIC_IP   -DHAVE_KALLSYMS_LOOKUP_NAME" modules
make[1]: Entering directory `/usr/src/linux-2.6.16.60-0.21'
  CC [M]  /root/dfm-0.15.1/kprobes.o
  CC [M]  /root/dfm-0.15.1/log.o
  CC [M]  /root/dfm-0.15.1/debugfs_control.o
  CC [M]  /root/dfm-0.15.1/debugfs_stats.o
  CC [M]  /root/dfm-0.15.1/filetree.o
  CC [M]  /root/dfm-0.15.1/namehash.o
  LD [M]  /root/dfm-0.15.1/dfm_mod.o
```

```
  Building modules, stage 2.
  MODPOST
  CC      /root/dfm-0.15.1/dfm_mod.mod.o
  LD [M]  /root/dfm-0.15.1/dfm_mod.ko
make[1]: Leaving directory `/usr/src/linux-2.6.16.60-0.21'
woody:~/dfm-0.15.1 #
```

The Makefile will detect the version of the currently running kernel and will build the DFM module for that kernel version. If you want to build the DFM module for a different kernel that you have installed on your system, set the `KERNEL_RELEASE` variable on the command line when you run `make`.

```
woody:~/dfm-0.15.1 # make KERNEL_RELEASE=2.6.25-smp
make -C /lib/modules/2.6.25-smp/build M=/root/dfm-0.15.1 KCPPFLAGS="
-DHAVE_INODE_I_PRIVATE -DHAVE_KPROBE_SYMBOL_NAME " modules
make[1]: Entering directory `/usr/src/linux-2.6.25'
Makefile:548: "WARNING: Appending $KCPPFLAGS (-DHAVE_INODE_I_PRIVATE
-DHAVE_KPROBE_SYMBOL_NAME ) from command line to kernel $CPPFLAGS"
  CC [M]  /root/dfm-0.15.1/kprobes.o
  CC [M]  /root/dfm-0.15.1/log.o
  CC [M]  /root/dfm-0.15.1/debugfs_control.o
  CC [M]  /root/dfm-0.15.1/debugfs_stats.o
  CC [M]  /root/dfm-0.15.1/filetree.o
  CC [M]  /root/dfm-0.15.1/namehash.o
  LD [M]  /root/dfm-0.15.1/dfm_mod.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/dfm-0.15.1/dfm_mod.mod.o
  LD [M]  /root/dfm-0.15.1/dfm_mod.ko
make[1]: Leaving directory `/usr/src/linux-2.6.25'
woody:~/dfm-0.15.1 #
```

Kernels before 2.6.19 have a bug in the debugfs code which causes the kernel to panic when the debugfs code faults on a bad pointer when it encounters an error when trying to allocate memory. You may hit this problem if DFM is collecting statistics on a lot of files and the system becomes memory constrained. If possible, apply the `debugfs-fix.patch` to your kernel source and rebuild the kernel.

Installation

Installing DFM is optional. You can run the utility out of the source directory once it has been built. When you run it out of the source directory it will load the kernel module that was built in that directory.

If you want, you can install the utility by running `make install`, which will run `make modules_install` to install the DFM kernel module into `/lib/modules/<KERNEL_VERSION>/` and will copy the `dfm` and `dfm-ctrl` scripts to `/usr/local/sbin`. When the utility is run from `/usr/local/sbin` it will load the DFM kernel module that was installed in `/lib/modules/`uname -r`/` instead of the one in the source directory.

Basic Operation

DFM provides a front end bash script, named `dfm`, that provides a user-friendly interface for running DFM. The script will startup DFM, gather the statistics from the debugfs file system, and shutdown DFM. To get started with DFM simply run the `dfm` script with no parameters. The default behavior is to gather the statistics every five seconds and print them to standard out.

Here is some sample output from one interval.

```
Start:  2009/09/10 20:15:21.882323
Stop:   2009/09/10 20:15:26.885819

/dev/sda2
    I/O           Count        Bytes
    Read              1         4096
    Write             0            0
    Total             1         4096

/var/db2/.fmcd.lock
    I/O           Count        Bytes
    Read              0            0
    Write             3        12288
    Total             3        12288

/usr/local/bin/screen-4.0.2
    I/O           Count        Bytes
    Read              1         4096
    Write             0            0
    Total             1         4096
```

The output begins with the start time and stop time for the interval. Each of the entries for the files begins with the file name on one line followed by lines for each type of I/O done to that file. The I/O type lines list the I/O type, the count of requests of that type, and the total number of bytes for that type.

Parameters

The `dfm` script has three parameters: *interval*, *iterations*, and *exclude*.

interval=<interval>

You can specify the amount of time the script waits between each gathering of the statistics from the debugfs tree. The interval can have optional units of s[econds], m[inutes], h[ours], or d[ays]. The default unit is seconds.

For example, you can run:

`dfm interval=10`	Wait 10 seconds between each sample.
`dfm interval=1m`	Wait 1 minute between each sample.
`dfm interval=3hours`	Wait 3 hours between each sample.

The default interval is 5 seconds.

iterations=<iterations>

You can specify the number of times the script gathers the statistics from the debugfs tree. The default number of iterations is 24 hours divided by the interval. For example, at the default interval of 5 seconds, the default number of iterations is 60*60*24/5 = 17280.

For example, you can run:

`dfm iterations=12`	Take 12 samples with 5 seconds (the default) between each sample.
`dfm interval=1m iterations=5`	Take 5 samples with 1 minute between each sample.

exclude=<device list>

Your system may have numerous physical disks. You may be interested in monitoring the files on only a subset of the physical disks on the system. For example, you may only be interested in monitoring files from a specific file system. Monitoring all the disks can end up dumping a lot of data, most of which you would ignore. DFM provides a way of excluding devices from being monitored.

You can set a list of devices to be excluded with the `exclude=<device list>` parameter. The devices may be specified in either `[/dev/]<name>` or `<major>:<minor>` format. If you specify more than one device, you must enclose the list in quotes so that the list is passed as a single parameter, for example, `exclude="sda sdb 8:96"`.

Options

The `dfm` script has a few options.

Option	Description
-V, --version	Display the DFM version and then exit.
-r, --raw	Display the raw statistics data without labels. This option is useful if you want to process the output with another program. It strips the labels and reduces the whitespace for easier parsing.
-h, -?, --help	Display the help text.

Signals

The `dfm` script handles several signals.

SIGUSR2

Send SIGUSR2 to the running `dfm` process to tell it to suspend its operation. The script will stop the DFM probes and wait for a SIGUSR1 signal to resume.

SIGUSR1

Send SIGUSR1 to the running `dfm` process to tell it to resume its operation. If the script was suspended, it will dump the current statistics, start the DFM probes, and resume its loop of dumping the

statistics after each interval. If the script was already running, SIGUSR1 has the side effect of making the script dump the statistics at that moment instead of waiting for the interval to expire.

Termination

If the script is interrupted (Ctrl-c, or a signal to quit) it gathers the statistics one last time before it unloads the kernel module and exits. This feature is useful if you don't know ahead of time how long you want the interval to be. For example, you may want to gather the disk file statistics during the run of a program, but you don't know how long the program will run. You can start the `dfm` script with a large interval, e.g., 10000, start the program, and then kill the `dfm` script when the program is finished. The script will then dump the disk file statistics from the time it was started.

Advanced Operation

This section provides the details on how DFM operates.

The user interface to the DFM kernel module is through the debugfs file system. The `dfm` script makes sure that the debugfs file system is mounted before it begins its work. The debugfs file system is usually mounted on `/sys/kernel/debug`.

The DebugFS Tree

The DFM kernel module creates a directory named `dfm` in the root of the debugfs tree. All the DFM debugfs activity occurs in the `dfm` directory. DFM maintains several control files in the `dfm` directory.

File name	Function
`running`	Read this file to see if the probes are running. (The DFM kernel module can be loaded but not running.) 0 = probes stopped 1 = probes running Write a "0" to this file to stop the probes. Write a non-zero value to the file to start the probes.
`reset`	Write a non-zero value to this file to reset the statistics. The probes are temporarily stopped, everything in the `dfm/statistics` directory is deleted, and the probes are restarted.
`layout`	Read this file to see the available layouts and which one is currently being used. The one in use is in square brackets. Write a layout name to this file to set a new layout. Setting a new layout causes a reset of the statistics.
`exclude`	Read this file to see the list of devices, indicated by <major>:<minor>, that currently excluded from being monitored. Write a list of devices, <major>:<minor> couples separated by spaces, to this file to set the list of excluded devices. Write and empty string to this file to clear the list of excluded devices.

File name	Function
`verbose`	Read this file to see the verbose level. The verbose level determines which messages get logged. Write a number to this file to set the verbose level. The DFM verbose levels match the levels used for printk() messages in the kernel. Valid values are from 0 to 7.
`start_time`	Read this file to see the time that DFM started collecting statistics. If the probes were never started since the kernel module was loaded, this file will be empty.
`stop_time`	Read this file to see the time that DFM stopped collecting statistics. If the probes were never started since the kernel module was loaded, or if DFM is currently collecting statistics, this file will be empty.
`log`	Read this file to see DFM's in-memory log.

The statistics are kept in a `statistics` subdirectory in the `dfm` directory according to the layout that is currently in use.

Layouts

So exactly how are the statistics laid out in the debugfs tree? The DFM architecture has pluggable layouts. In this release, DFM provides two layouts: *filetree* and *namehash*.

Filetree

The *filetree* layout creates the debugfs entries using the same file name and directory structure of the currently mounted file systems. The name of the file for which the statistics are being collected becomes the name of a directory in the debugfs tree. Within that directory are files with the names of the different types of I/O that have been made to the file. For example, the number of bytes requested for type "Readahead" on file `/usr/src/linux/kernel/sched.c` would be logged in the file `<debugfs-root>/dfm/statistics/usr/src/linux/kernel/sched.c/Readahead`.

Namehash

The filetree layout nicely preserves the file structure of the system. It makes it obvious which files the statistics are for. However, it can take a long time to traverse the directory structure when it comes time to gather the statistics. Gathering the statistics should be fast so that the interruption to collecting the statistics is minimal. (Any program gathering statistics from the DFM debugfs tree should stop the DFM kernel module's statistics collection while gathering the statistics to avoid the danger of trying to traverse a changing directory tree. Stopping the probes for too long can result in a significant loss of statistics collected.)

The *namehash* layout creates a unique hash value for each file and creates a directory that is named with the ASCII string of the hexadecimal value of the hash. As with the filetree layout, the directory contains files with the names of the different types of I/O that have been made to the file. Any program that gathers the statistics needs to traverse only two levels deep to get the statistics. In this layout the name of the file is not apparent. The namehash layout puts a file named `filename` into the directory along with the I/O type files. The `filename` file contains the full path of the file name.

The `dfm-ctrl` Script

The `dfm-ctrl` script provides several commands that make it easier to work with DFM control files and statistics. The structure of the debugfs tree is abstracted away to simple commands.

load

The *load* command attempts to load the DFM kernel module. In an effort to support development of the utility, the script does not require that the module be installed in the `/lib/modules/`uname -r`` tree. The script checks to see if the module exists in the same directory from which the script was run. If it finds the module there, it uses `insmod` to load the module. If it does not find the module there, is uses `modprobe` to load the module. Thus, if the script is run from the build directory, it will find the built module in the directory and will load that version of the module. If the script is invoked from its installed directory, `/usr/local/sbin`, it will not find the DFM module there and will run `modprobe`, which will load the installed version of the module, the one in `/lib/modules/`uname -r`/extra/.`

start

The *start* command will load the DFM kernel module if it is not already loaded and then, if the probes are not already running, echoes "1" to `<debugfs_mount>/dfm/running` to start the probes.

stop

The *stop* command checks to see if the DFM probes are running (i.e., `cat <debugfs_mount>/dfm/running` does not return 0) and, if so, echoes "0" to `<debugfs_mount>/dfm/running` to stop the probes.

reset

The *reset* command echoes a "1" to the `<debugfs_mount>/dfm/reset` file which tells the DFM kernel module to clear the statistics.

stats

The *stats* command dumps the statistics in the debugfs tree. The command has a safety check to make sure that the probes are not running before it gathers the statistics. Gathering the statistics involves walking the `<debugfs_mount>/dfm/statistics/` tree. If the probes are running, the contents of the tree can be changing, making a tree walk dangerous. The user can override the safety check with the `--force` option. (The force option can also be specified as `-f` or `force=yes`.)

status

The *status* command displays whether the DFM kernel module is loaded or not, what layouts are available, the current layout being used, the current verbose level, whether the probes are running or not, what time the probes started running, and what time they stopped, if the probes are stopped.

layout

Without any parameters, the *layout* command displays the layout that DFM is currently using. If the *layout* command is given a parameter, it will attempt to set the DFM layout to the parameter given. The *layout* command checks to make sure that the layout given appears in the list of layouts supported in `<debugfs_mount>/dfm/layout`.

exclude

Without any parameters, the *exclude* command displays the list of devices that are currently excluded from being monitored. The devices are listed as <major>:<minor> pairs. As a help to the user, the script finds the device name associated with the <major>:<minor> and displays the name in parenthesis next to the <major>:<minor>, for example: `8:0 (/dev/sda)` If the *exclude* command is given parameters it will use them to set the list of excluded devices. Devices can be specified in the <major>:<minor> format or by name with the optional `/dev/` prefix, e.g., `sda`.

The *exclude* command has several subcommands to aid in setting the list of devices. The debugfs interface to DFM only supports the setting of the entire list. Users, however, may want to add devices to the list or remove a subset of devices from the list. Also, since the exclude command interprets the lack of parameters to be a query of the excluded devices list, there is no way to set an empty lists, i.e., clear the list. The exclude command supports the following subcommands.

add

The *add* subcommand adds devices to the excluded devices list. As with the native *exclude* command, devices can be specified in the <major>:<minor> format or by name. The *add* subcommand reads the current list of excluded devices from `<debugfs_mount>/dfm/exclude`, appends the new devices, and writes the new list to `<debugfs_mount>/dfm/exclude`.

remove

The *remove* subcommand removes devices from the excluded devices list. As with the native *exclude* command, devices can be specified in the <major>:<minor> format or by name. The *remove* subcommand reads the current list of excluded devices from `<debugfs_mount>/dfm/exclude`. Then for each device given to the command, it searches the list of excluded devices for the given device. If it finds it, it removes it from the list. It then write the new list back to `<debugfs_mount>/dfm/exclude`.

clear

The *clear* subcommand writes an empty list to `<debugfs_mount>/dfm/exclude`, which effectively removes all devices from the exclude list.

verbose

Without any parameters, the *verbose* command displays the current DFM verbose level. It simply does a `cat <debugfs_mount>/dfm/verbose`. If the *verbose* command is given a parameter, it will attempt to set the DFM verbose level to the parameter given by echoing the value into

`<debugfs_mount>/dfm/verbose`. The *verbose* command checks to make sure that the verbose level given is a positive integer. It will also accept the keyword names for each level: emerg[ency] (0), alert (1), crit[ical] (2), err[or] (3), warn[ing] (4), notice (5), info (6), and debug (7).

log

The *log* command dumps out the DFM internal log. It simply does a `cat` of the `<debugfs_mount>/dfm/log` file. For convenience, the *log* command checks to see if standard out is a tty and the environment has the PAGER variable set. If so, it pipes the output of the log file into the pager for easy browsing.

unload

The *unload* command unloads the DFM kernel module. It simply does an `rmmod dfm_mod` if the module is loaded.

Formatting Output

The output of the DFM statistics is not be sorted in any order. This package has several Perl scripts that will sort the statistics into an order that is meaningful to the user. The scripts can be called with the name of a file that contains DFM output, or they can be used as filters by taking input from standard in. The scripts send their output to standard out.

dfm-by-file.pl

For each file given in the DFM output, dfm-by-file.pl sums up the counts and bytes for each interval. It sums all the read I/O types into one statistic for reads and all the write I/O types into one statistic for writes. It then sorts the data by file name and prints it out. Here is some sample output from a build of DFM. (The entries for the numerous kernel header files have been removed.)

```
  I/O        Count        Bytes File
 Read            1        16384 /bin/cat
Write            0            0 /bin/cat
Total            1        16384 /bin/cat

 Read            1        16384 /bin/mv
Write            0            0 /bin/mv
Total            1        16384 /bin/mv

 Read            1        16384 /bin/pwd
Write            0            0 /bin/pwd
Total            1        16384 /bin/pwd

 Read            1        16384 /bin/sleep
Write            0            0 /bin/sleep
Total            1        16384 /bin/sleep

 Read            1        16384 /bin/uname
Write            0            0 /bin/uname
Total            1        16384 /bin/uname

 Read          529      2166784 /dev/sda2
Write         1263      5173248 /dev/sda2
Total         1792      7340032 /dev/sda2

 Read            1         4096 /root/dfm-0.15.0
Write            0            0 /root/dfm-0.15.0
Total            1         4096 /root/dfm-0.15.0

 Read            1         4096 /root/dfm-0.15.0/.tmp_debugfs_stats.o
Write            0            0 /root/dfm-0.15.0/.tmp_debugfs_stats.o
Total            1         4096 /root/dfm-0.15.0/.tmp_debugfs_stats.o

 Read            1         4096 /root/dfm-0.15.0/.tmp_versions/dfm_mod.mod
Write            0            0 /root/dfm-0.15.0/.tmp_versions/dfm_mod.mod
Total            1         4096 /root/dfm-0.15.0/.tmp_versions/dfm_mod.mod

 Read            1         4096 /root/dfm-0.15.0/Makefile
Write            0            0 /root/dfm-0.15.0/Makefile
Total            1         4096 /root/dfm-0.15.0/Makefile

 Read            1        20480 /root/dfm-0.15.0/debugfs_control.c
Write            0            0 /root/dfm-0.15.0/debugfs_control.c
Total            1        20480 /root/dfm-0.15.0/debugfs_control.c

 Read            1         4096 /root/dfm-0.15.0/debugfs_control.h
Write            0            0 /root/dfm-0.15.0/debugfs_control.h
Total            1         4096 /root/dfm-0.15.0/debugfs_control.h

 Read            1        16384 /root/dfm-0.15.0/debugfs_stats.c
Write            0            0 /root/dfm-0.15.0/debugfs_stats.c
Total            1        16384 /root/dfm-0.15.0/debugfs_stats.c

 Read            1         4096 /root/dfm-0.15.0/debugfs_stats.h
Write            0            0 /root/dfm-0.15.0/debugfs_stats.h
Total            1         4096 /root/dfm-0.15.0/debugfs_stats.h

 Read            1         4096 /root/dfm-0.15.0/dfm.h
Write            0            0 /root/dfm-0.15.0/dfm.h
Total            1         4096 /root/dfm-0.15.0/dfm.h

 Read            1         4096 /root/dfm-0.15.0/dfm_mod.ko
Write            0            0 /root/dfm-0.15.0/dfm_mod.ko
Total            1         4096 /root/dfm-0.15.0/dfm_mod.ko

 Read            1         4096 /root/dfm-0.15.0/field-in-struct.pl
Write            0            0 /root/dfm-0.15.0/field-in-struct.pl
Total            1         4096 /root/dfm-0.15.0/field-in-struct.pl
```

```
 Read            1         8192 /root/dfm-0.15.0/filetree.c
Write            0            0 /root/dfm-0.15.0/filetree.c
Total            1         8192 /root/dfm-0.15.0/filetree.c

 Read            0            0 /root/dfm-0.15.0/filetree.o
Write            2         8192 /root/dfm-0.15.0/filetree.o
Total            2         8192 /root/dfm-0.15.0/filetree.o

 Read            1        49152 /root/dfm-0.15.0/kprobes.c
Write            0            0 /root/dfm-0.15.0/kprobes.c
Total            1        49152 /root/dfm-0.15.0/kprobes.c

 Read            1         4096 /root/dfm-0.15.0/kprobes.h
Write            0            0 /root/dfm-0.15.0/kprobes.h
Total            1         4096 /root/dfm-0.15.0/kprobes.h

 Read            0            0 /root/dfm-0.15.0/kprobes.o
Write            4        16384 /root/dfm-0.15.0/kprobes.o
Total            4        16384 /root/dfm-0.15.0/kprobes.o

 Read            1         8192 /root/dfm-0.15.0/log.c
Write            0            0 /root/dfm-0.15.0/log.c
Total            1         8192 /root/dfm-0.15.0/log.c

 Read            1         4096 /root/dfm-0.15.0/log.h
Write            0            0 /root/dfm-0.15.0/log.h
Total            1         4096 /root/dfm-0.15.0/log.h

 Read            1        12288 /root/dfm-0.15.0/namehash.c
Write            0            0 /root/dfm-0.15.0/namehash.c
Total            1        12288 /root/dfm-0.15.0/namehash.c

 Read            0            0 /root/dfm-0.15.0/namehash.o
Write            2         8192 /root/dfm-0.15.0/namehash.o
Total            2         8192 /root/dfm-0.15.0/namehash.o

 Read            1        16384 /usr/bin/as
Write            0            0 /usr/bin/as
Total            1        16384 /usr/bin/as

 Read            1        16384 /usr/bin/gcc
Write            0            0 /usr/bin/gcc
Total            1        16384 /usr/bin/gcc

 Read            1        16384 /usr/bin/ld
Write            0            0 /usr/bin/ld
Total            1        16384 /usr/bin/ld

 Read            1        16384 /usr/bin/make
Write            0            0 /usr/bin/make
Total            1        16384 /usr/bin/make

 Read            1        16384 /usr/bin/objdump
Write            0            0 /usr/bin/objdump
Total            1        16384 /usr/bin/objdump

 Read            1        16384 /usr/bin/perl
Write            0            0 /usr/bin/perl
Total            1        16384 /usr/bin/perl

 Read            1        16384 /usr/lib64/gcc/x86_64-suse-linux/4.1.2/cc1
Write            0            0 /usr/lib64/gcc/x86_64-suse-linux/4.1.2/cc1
Total            1        16384 /usr/lib64/gcc/x86_64-suse-linux/4.1.2/cc1

 Read            1         8192 /usr/lib64/gcc/x86_64-suse-linux/4.1.2/include/stdarg.h
Write            0            0 /usr/lib64/gcc/x86_64-suse-linux/4.1.2/include/stdarg.h
Total            1         8192 /usr/lib64/gcc/x86_64-suse-linux/4.1.2/include/stdarg.h

 Read            1        16384 /usr/lib64/libbfd-2.16.91.0.5.so
Write            0            0 /usr/lib64/libbfd-2.16.91.0.5.so
Total            1        16384 /usr/lib64/libbfd-2.16.91.0.5.so

 Read            1        16384 /usr/lib64/libopcodes-2.16.91.0.5.so
Write            0            0 /usr/lib64/libopcodes-2.16.91.0.5.so
Total            1        16384 /usr/lib64/libopcodes-2.16.91.0.5.so
```

dfm-by-count.pl

For each file given in the DFM output, dfm-by-count.pl sums up the counts and bytes for each interval. It sums all the I/O types into one statistic for the file. It then sorts the data by count and prints it out. Here is the same output from the build of DFM, this time processed by dfm-by-count.pl.

```
Count        Bytes File
 1792      7340032 /dev/sda2
    4        16384 /root/dfm-0.15.0/kprobes.o
    3        12288 /var/db2/.fmcd.lock
    2         8192 /root/dfm-0.15.0/filetree.o
    2         8192 /root/dfm-0.15.0/namehash.o
    1        12288 /root/dfm-0.15.0/namehash.c
    1        16384 /usr/bin/make
    1        16384 /root/dfm-0.15.0/debugfs_stats.c
    1         8192 /root/dfm-0.15.0/log.c
    1         4096 /root/dfm-0.15.0/kprobes.h
    1        16384 /usr/bin/as
    1        49152 /root/dfm-0.15.0/kprobes.c
    1         4096 /root/dfm-0.15.0/log.h
    1        16384 /usr/bin/ld
    1         8192 /root/dfm-0.15.0/filetree.c
    1        16384 /usr/bin/objdump
    1        16384 /usr/bin/gcc
    1        16384 /usr/lib64/libopcodes-2.16.91.0.5.so
    1        16384 /usr/lib64/libbfd-2.16.91.0.5.so
    1         4096 /root/dfm-0.15.0/debugfs_stats.h
    1         8192 /usr/lib64/gcc/x86_64-suse-linux/4.1.2/include/stdarg.h
    1         4096 /root/dfm-0.15.0/.tmp_debugfs_stats.o
    1         4096 /root/dfm-0.15.0/.tmp_versions/dfm_mod.mod
    1         4096 /root/dfm-0.15.0/debugfs_control.h
    1        20480 /root/dfm-0.15.0/debugfs_control.c
    1         4096 /var/spool/postfix/incoming
    1        16384 /bin/uname
    1        16384 /bin/pwd
    1        16384 /bin/mv
    1        16384 /usr/lib64/gcc/x86_64-suse-linux/4.1.2/cc1
    1         4096 /root/dfm-0.15.0/field-in-struct.pl
    1         4096 /root/dfm-0.15.0
    1         4096 /root/dfm-0.15.0/dfm.h
    1        16384 /bin/sleep
    1        16384 /usr/bin/perl
    1         4096 /root/dfm-0.15.0/Makefile
    1         4096 /root/dfm-0.15.0/dfm_mod.ko
    1        16384 /bin/cat
```

dfm-by-bytes.pl

For each file given in the raw output, dfm-by-bytes.pl sums up the counts and bytes for each interval. It sums all the I/O types into one statistic for the file. It then sorts the data by bytes and prints it out. Here is the same output from the build of DFM, this time processed by dfm-by-bytes.pl.

```
    Bytes        Count File
  7340032         1792 /dev/sda2
    49152            1 /root/dfm-0.15.0/kprobes.c
    20480            1 /root/dfm-0.15.0/debugfs_control.c
    16384            1 /usr/bin/make
    16384            1 /root/dfm-0.15.0/debugfs_stats.c
    16384            1 /usr/bin/as
    16384            1 /usr/bin/ld
    16384            1 /usr/bin/objdump
    16384            1 /usr/bin/gcc
    16384            1 /usr/lib64/libopcodes-2.16.91.0.5.so
    16384            1 /usr/lib64/libbfd-2.16.91.0.5.so
    16384            1 /bin/uname
    16384            1 /bin/pwd
    16384            4 /root/dfm-0.15.0/kprobes.o
    16384            1 /bin/mv
    16384            1 /usr/lib64/gcc/x86_64-suse-linux/4.1.2/cc1
    16384            1 /bin/sleep
    16384            1 /usr/bin/perl
    16384            1 /bin/cat
    12288            1 /root/dfm-0.15.0/namehash.c
    12288            3 /var/db2/.fmcd.lock
     8192            1 /root/dfm-0.15.0/log.c
     8192            1 /root/dfm-0.15.0/filetree.c
     8192            2 /root/dfm-0.15.0/filetree.o
     8192            1 /usr/src/linux-2.6.16.60-0.21-fix/include/asm-x86_64/page.h
     8192            2 /root/dfm-0.15.0/namehash.o
     4096            1 /root/dfm-0.15.0/kprobes.h
     4096            1 /root/dfm-0.15.0/log.h
     4096            1 /root/dfm-0.15.0/debugfs_stats.h
     4096            1 /root/dfm-0.15.0/.tmp_debugfs_stats.o
     4096            1 /root/dfm-0.15.0/.tmp_versions/dfm_mod.mod
     4096            1 /root/dfm-0.15.0/debugfs_control.h
     4096            1 /var/spool/postfix/incoming
     4096            1 /root/dfm-0.15.0/field-in-struct.pl
     4096            1 /root/dfm-0.15.0
     4096            1 /root/dfm-0.15.0/dfm.h
     4096            1 /root/dfm-0.15.0/Makefile
     4096            1 /root/dfm-0.15.0/dfm_mod.ko
```