NILFS2

NILFS2 is a log-structured file system (LFS) supporting continuous snapshotting. In addition to versioning capability of the entire file system, users can even restore files mistakenly overwritten or destroyed just a few seconds ago. Since NILFS2 can keep consistency like conventional LFS, it achieves quick recovery after system crashes.

NILFS2 creates a number of checkpoints every few seconds or per synchronous write basis (unless there is no change). Users can select significant versions among continuously created checkpoints, and can change them into snapshots which will be preserved until they are changed back to checkpoints.

There is no limit on the number of snapshots until the volume gets full. Each snapshot is mountable as a read-only file system concurrently with its writable mount, and this feature is convenient for online backup.

The userland tools are included in nilfs-utils package, which is available from the following download page. At least "mkfs.nilfs2", "mount.nilfs2", "umount.nilfs2", and "nilfs_cleanerd" (so called cleaner or garbage collector) are required. Details on the tools are described in the man pages included in the package.

Project web page: https://nilfs.sourceforge.io/
Download page: https://nilfs.sourceforge.io/en/download.html
List info: http://vger.kernel.org/vger-lists.html#linux-nilfs

Caveats

Features which NILFS2 does not support yet:

- atime
- extended attributes
- POSIX ACLs
- quotas
- fsck
- defragmentation

Mount options

NILFS2 supports the following mount options: (*) == default

barrier(*)	This enables/disables the use of write barriers. This
nobarrier	requires an IO stack which can support barriers, and if nilfs gets an error on a barrier write, it will disable again with a warning.
errors=continue	Keep going on a filesystem error.
errors=remount-ro(*)	Remount the filesystem read-only on an error.
errors=panic	Panic and halt the machine if an error occurs.
cp=n	Specify the checkpoint-number of the snapshot to be mounted. Checkpoints and snapshots are listed by lscp user command. Only the checkpoints marked as snapshot are mountable with this option. Snapshot is read-only, so a read-only mount option must be specified together.
order=relaxed(*)	Apply relaxed order semantics that allows modified data blocks to be written to disk without making a checkpoint if no metadata update is going. This mode is equivalent to the ordered data mode of the ext3 filesystem except for the updates on data blocks still conserve atomicity. This will improve synchronous write performance for overwriting.
order=strict	Apply strict in-order semantics that preserves sequence of all file operations including overwriting of data blocks. That means, it is guaranteed that no overtaking of events occurs in the recovered file system after a crash.
norecovery	Disable recovery of the filesystem on mount. This disables every write access on the device for read-only mounts or snapshots. This option will fail for r/w mounts on an unclean volume.
discard	This enables/disables the use of discard/TRIM commands.
nodiscard(*)	The discard/TRIM commands are sent to the underlying block device when blocks are freed. This is useful for SSD devices and sparse/thinly-provisioned LUNs.

Ioctls

There is some NILFS2 specific functionality which can be accessed by applications through the system call interfaces. The list of all NILFS2 specific ioctls are shown in the table below.

Table of NILFS2 specific ioctls:

Ioctl	Description
NILFS_IOCTL_CHANGE_CPMODE	Change mode of given checkpoint between checkpoint and snapshot state. This ioctl is used in chcp and mkcp utilities.
NILFS_IOCTL_DELETE_CHECKPOINT	Remove checkpoint from NILFS2 file system. This ioctl is used in rmcp utility.
NILFS_IOCTL_GET_CPINFO	Return info about requested checkpoints. This ioctl is used in lscp utility and by nilfs_cleanerd daemon.
NILFS_IOCTL_GET_CPSTAT	Return checkpoints statistics. This ioctl is used by lscp, rmcp utilities and by nilfs_cleanerd daemon.
NILFS_IOCTL_GET_SUINFO	Return segment usage info about requested segments. This ioctl is used in lssu, nilfs_resize utilities and by nilfs_cleanerd daemon.
NILFS_IOCTL_SET_SUINFO	Modify segment usage info of requested segments. This ioctl is used by nilfs_cleanerd daemon to skip unnecessary cleaning operation of segments and reduce performance penalty or wear of flash device due to redundant move of in-use blocks.
NILFS_IOCTL_GET_SUSTAT	Return segment usage statistics. This ioctl is used in lssu, nilfs_resize utilities and by nilfs_cleanerd daemon.
NILFS_IOCTL_GET_VINFO	Return information on virtual block addresses. This ioctl is used by nilfs_cleanerd daemon.
NILFS_IOCTL_GET_BDESCS	Return information about descriptors of disk block numbers. This ioctl is used by nilfs_cleanerd daemon.
NILFS_IOCTL_CLEAN_SEGMENTS	Do garbage collection operation in the environment of requested parameters from userspace. This ioctl is used by nilfs_cleanerd daemon.
NILFS_IOCTL_SYNC	Make a checkpoint. This ioctl is used in mkcp utility.
NILFS_IOCTL_RESIZE	Resize NILFS2 volume. This ioctl is used by nilfs_resize utility.
NILFS_IOCTL_SET_ALLOC_RANGE	Define lower limit of segments in bytes and upper limit of segments in bytes. This ioctl is used by nilfs_resize utility.

NILFS2 usage

To use nilfs2 as a local file system, simply:

```
# mkfs -t nilfs2 /dev/block_device
# mount -t nilfs2 /dev/block_device /dir
```

This will also invoke the cleaner through the mount helper program (mount.nilfs2).

Checkpoints and snapshots are managed by the following commands. Their manpages are included in the nilfs-utils package above.

lscp	list checkpoints or snapshots.
mkcp	make a checkpoint or a snapshot.
chcp	change an existing checkpoint to a snapshot or vice versa.
rmcp	invalidate specified checkpoint(s).

To mount a snapshot:

```
# mount -t nilfs2 -r -o cp=<cno> /dev/block_device /snap_dir
```

where <cno> is the checkpoint number of the snapshot.

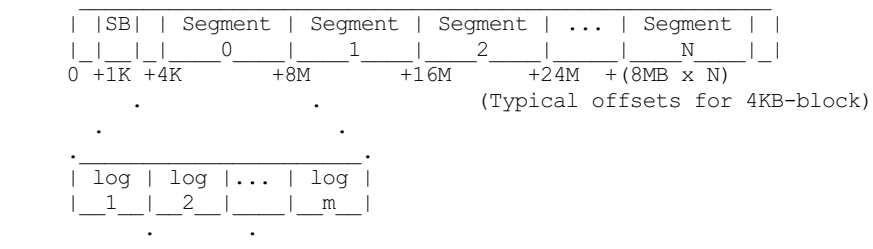
To unmount the NILFS2 mount point or snapshot, simply:

```
# umount /dir
```

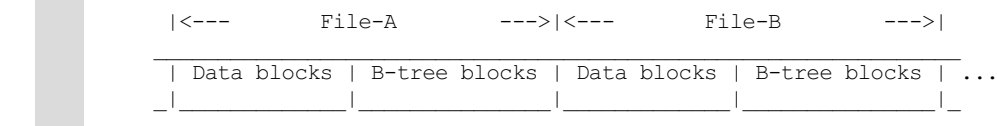
Then, the cleaner daemon is automatically shut down by the umount helper program (umount.nilfs2).

Disk format

A nilfs2 volume is equally divided into a number of segments except for the super block (SB) and segment #0. A segment is the container of logs. Each log is composed of summary information blocks, payload blocks, and an optional super root block (SR):

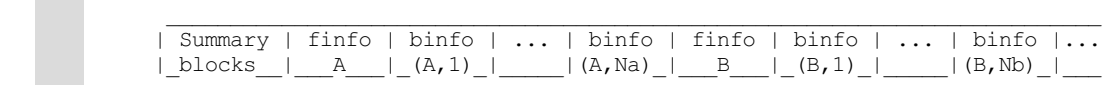


The payload blocks are organized per file, and each file consists of data blocks and B-tree node blocks:



Since only the modified blocks are written in the log, it may have files without data blocks or B-tree node blocks.

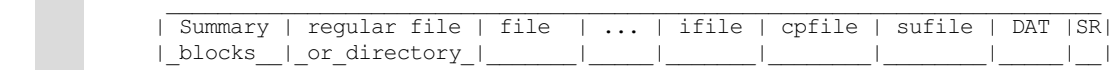
The organization of the blocks is recorded in the summary information blocks, which contains a header structure (nilfs_segment_summary), per file structures (nilfs_finfo), and per block structures (nilfs_binfo):



The logs include regular files, directory files, symbolic link files and several meta data files. The mata data files are the files used to maintain file system meta data. The current version of NILFS2 uses the following meta data files:

```
1) Inode file (ifile)             -- Stores on-disk inodes
2) Checkpoint file (cpfile)       -- Stores checkpoints
3) Segment usage file (sufile)    -- Stores allocation state of segments
4) Data address translation file  -- Maps virtual block numbers to usual
   (DAT)                             block numbers.  This file serves to
                                     make on-disk blocks relocatable.
```

The following figure shows a typical organization of the logs:



To stride over segment boundaries, this sequence of files may be split into multiple logs. The sequence of logs that should be treated as logically one log, is delimited with flags marked in the segment summary. The recovery code of nilfs2 looks this boundary information to ensure atomicity of updates.

The super root block is inserted for every checkpoints. It includes three special inodes, inodes for the DAT, cpfile, and sufile. Inodes of regular files, directories, symlinks and other special files, are included in the ifile. The inode of ifile itself is included in the corresponding checkpoint entry in the cpfile. Thus, the hierarchy among NILFS2 files can be depicted as follows:

```
Super block (SB)
     |
     v
Super root block (the latest cno=xx)
     |-- DAT
     |-- sufile
     `-- cpfile
            |-- ifile (cno=c1)
            |-- ifile (cno=c2) ---- file (ino=i1)
            :        :          |-- file (ino=i2)
            `-- ifile (cno=xx)  |-- file (ino=i3)
                                :        :
                                `-- file (ino=yy)
                                  ( regular file, directory, or symlink )
```

For detail on the format of each file, please see nilfs2_ondisk.h located at include/uapi/linux directory.

There are no patents or other intellectual property that we protect with regard to the design of NILFS2. It is allowed to replicate the design in hopes that other operating systems could share (mount, read, write, etc.) data stored in this format.