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:	http://www.nilfs.org/en/
Download page:	http://www.nilfs.org/en/download.html
Git tree web page:	http://www.nilfs.org/git/
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
- resize
- defragmentation

Mount options

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

nobarrier	Disables barriers.
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 Issue discard/TRIM commands to the underlying block device when blocks are freed. This is useful for SSD devices and sparse/thinly-provisioned LUNs.

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

```
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):

```
  ______________________________________________________
 | |SB| | Segment | Segment | Segment | ... | Segment | |
 |_|__|_|____0____|____1____|____2____|_____|____N____|_|
 0 +1K +4K       +8M       +16M      +24M  +(8MB x N)
      .             .            (Typical offsets for 4KB-block)
   .                  .
 .______________________.
 | log  | log  |... | log |
 |__1__|__2__|____|__m__|
       .       .
     .               .
   .                       .
 .______________________________.
 | Summary | Payload blocks  |SR|
 |_blocks__|_________________|__|
```

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

```
    |<---       File-A        --->|<---       File-B        --->|
   _______________________________________________________________
    | Data blocks | B-tree blocks | Data blocks | B-tree 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):

```
  _________________________________________________________________________
 | Summary | finfo | binfo | ... | binfo | finfo | binfo | ... | binfo |...
 |_blocks__|___A___|_(A,1)_|_____|(A,Na)_|___B___|_(B,1)_|_____|(B,Nb)_|___
```

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:

```
  ______________________________________________________________________
 | Summary | regular file | file  | ... | ifile | cpfile | sufile | DAT |SR|
 |_blocks__|_or_directory_|_______|_____|_______|________|________|_____|__|
```

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 include/linux/nilfs2_fs.h.