F2FS in-mem data structures
fs/f2fs/f2fs.h

```
733 enum {
734         CP_TIME,
735         REQ_TIME,
736         MAX_TIME,
737 };
738
739 #ifdef CONFIG_F2FS_FS_ENCRYPTION
740 #define F2FS_KEY_DESC_PREFIX "f2fs:"
741 #define F2FS_KEY_DESC_PREFIX_SIZE 5
742 #endif
743 struct f2fs_sb_info {
744         struct super_block *sb;                 /* pointer to VFS super block */
745         struct proc_dir_entry *s_proc;          /* proc entry */
746         struct f2fs_super_block *raw_super;     /* raw super block pointer */
747         int valid_super_block;                  /* valid super block no */
748         unsigned long s_flag;                           /* flags for sbi */
749
750 #ifdef CONFIG_F2FS_FS_ENCRYPTION
751         u8 key_prefix[F2FS_KEY_DESC_PREFIX_SIZE];
752         u8 key_prefix_size;
753 #endif
754         /* for node-related operations */
755         struct f2fs_nm_info *nm_info;           /* node manager */
756         struct inode *node_inode;               /* cache node blocks */
757
758         /* for segment-related operations */
759         struct f2fs_sm_info *sm_info;           /* segment manager */
760
761         /* for bio operations */
762         struct f2fs_bio_info read_io;                   /* for read bios */
763         struct f2fs_bio_info write_io[NR_PAGE_TYPE];    /* for write bios */
764         struct mutex wio_mutex[NODE + 1];       /* bio ordering for NODE/DATA */
765
766         /* for checkpoint */
767         struct f2fs_checkpoint *ckpt;           /* raw checkpoint pointer */
768         int cur_cp_pack;                        /* remain current cp pack */
769         spinlock_t cp_lock;                     /* for flag in ckpt */
770         struct inode *meta_inode;               /* cache meta blocks */
771         struct mutex cp_mutex;                  /* checkpoint procedure lock */
772         struct rw_semaphore cp_rwsem;           /* blocking FS operations */
773         struct rw_semaphore node_write;         /* locking node writes */
774         wait_queue_head_t cp_wait;
775         unsigned long last_time[MAX_TIME];      /* to store time in jiffies */
776         long interval_time[MAX_TIME];           /* to store thresholds */
777
778         struct inode_management im[MAX_INO_ENTRY];      /* manage inode cache */
779
780         /* for orphan inode, use 0'th array */
781         unsigned int max_orphans;               /* max orphan inodes */
782
783         /* for inode management */
784         struct list_head inode_list[NR_INODE_TYPE];     /* dirty inode list */
785         spinlock_t inode_lock[NR_INODE_TYPE];   /* for dirty inode list lock */
786
787         /* for extent tree cache */
788         struct radix_tree_root extent_tree_root;/* cache extent cache entries */
789         struct rw_semaphore extent_tree_lock;   /* locking extent radix tree */
790         struct list_head extent_list;           /* lru list for shrinker */
791         spinlock_t extent_lock;                 /* locking extent lru list */
792         atomic_t total_ext_tree;                /* extent tree count */
793         struct list_head zombie_list;           /* extent zombie tree list */
794         atomic_t total_zombie_tree;             /* extent zombie tree count */
```

```
795         atomic_t total_ext_node;              /* extent info count */
796
797         /* basic filesystem units */
798         unsigned int log_sectors_per_block;     /* log2 sectors per block */
799         unsigned int log_blocksize;             /* log2 block size */
800         unsigned int blocksize;                 /* block size */
801         unsigned int root_ino_num;              /* root inode number*/
802         unsigned int node_ino_num;              /* node inode number*/
803         unsigned int meta_ino_num;              /* meta inode number*/
804         unsigned int log_blocks_per_seg;        /* log2 blocks per segment */
805         unsigned int blocks_per_seg;            /* blocks per segment */
806         unsigned int segs_per_sec;              /* segments per section */
807         unsigned int secs_per_zone;             /* sections per zone */
808         unsigned int total_sections;            /* total section count */
809         unsigned int total_node_count;          /* total node block count */
810         unsigned int total_valid_node_count;    /* valid node block count */
811         loff_t max_file_blocks;                 /* max block index of file */
812         int active_logs;                        /* # of active logs */
813         int dir_level;                          /* directory level */
814
815         block_t user_block_count;               /* # of user blocks */
816         block_t total_valid_block_count;        /* # of valid blocks */
817         block_t discard_blks;                   /* discard command candidats */
818         block_t last_valid_block_count;         /* for recovery */
819         u32 s_next_generation;                  /* for NFS support */
820         atomic_t nr_wb_bios;                    /* # of writeback bios */
821
822         /* # of pages, see count_type */
823         atomic_t nr_pages[NR_COUNT_TYPE];
824         /* # of allocated blocks */
825         struct percpu_counter alloc_valid_block_count;
826
827         /* valid inode count */
828         struct percpu_counter total_valid_inode_count;
829
830         struct f2fs_mount_info mount_opt;       /* mount options */
831
832         /* for cleaning operations */
833         struct mutex gc_mutex;                  /* mutex for GC */
834         struct f2fs_gc_kthread  *gc_thread;     /* GC thread */
835         unsigned int cur_victim_sec;            /* current victim section num */
836
837         /* threshold for converting bg victims for fg */
838         u64 fggc_threshold;
839
840         /* maximum # of trials to find a victim segment for SSR and GC */
841         unsigned int max_victim_search;
842
843         /*
844          * for stat information.
845          * one is for the LFS mode, and the other is for the SSR mode.
846          */
847 #ifdef CONFIG_F2FS_STAT_FS
848         struct f2fs_stat_info *stat_info;       /* FS status information */
849         unsigned int segment_count[2];          /* # of allocated segments */
850         unsigned int block_count[2];            /* # of allocated blocks */
851         atomic_t inplace_count;         /* # of inplace update */
852         atomic64_t total_hit_ext;               /* # of lookup extent cache */
853         atomic64_t read_hit_rbtree;             /* # of hit rbtree extent node */
854         atomic64_t read_hit_largest;            /* # of hit largest extent node */
855         atomic64_t read_hit_cached;             /* # of hit cached extent node */
856         atomic_t inline_xattr;                  /* # of inline_xattr inodes */
857         atomic_t inline_inode;                  /* # of inline_data inodes */
858         atomic_t inline_dir;                    /* # of inline_dentry inodes */
859         int bg_gc;                              /* background gc calls */
860         unsigned int ndirty_inode[NR_INODE_TYPE];       /* # of dirty inodes */
861 #endif
862         unsigned int last_victim[2];            /* last victim segment # */
863         spinlock_t stat_lock;                   /* lock for stat operations */
```

```
864
865         /* For sysfs suppport */
866         struct kobject s_kobj;
867         struct completion s_kobj_unregister;
868
869         /* For shrinker support */
870         struct list_head s_list;
871         struct mutex umount_mutex;
872         unsigned int shrinker_run_no;
873
874         /* For write statistics */
875         u64 sectors_written_start;
876         u64 kbytes_written;
877
878         /* Reference to checksum algorithm driver via cryptoapi */
879         struct crypto_shash *s_chksum_driver;
880
881         /* For fault injection */
882 #ifdef CONFIG_F2FS_FAULT_INJECTION
883         struct f2fs_fault_info fault_info;
884 #endif
885 };
```

```
606 struct f2fs_sm_info {
607         struct sit_info *sit_info;              /* whole segment information */
608         struct free_segmap_info *free_info;     /* free segment information */
609         struct dirty_seglist_info *dirty_info;  /* dirty segment information */
610         struct curseg_info *curseg_array;       /* active segment information */
611
612         block_t seg0_blkaddr;           /* block address of 0'th segment */
613         block_t main_blkaddr;           /* start block address of main area */
614         block_t ssa_blkaddr;            /* start block address of SSA area */
615
616         unsigned int segment_count;     /* total # of segments */
617         unsigned int main_segments;     /* # of segments in main area */
618         unsigned int reserved_segments; /* # of reserved segments */
619         unsigned int ovp_segments;      /* # of overprovision segments */
620
621         /* a threshold to reclaim prefree segments */
622         unsigned int rec_prefree_segments;
623
624         /* for small discard management */
625         struct list_head discard_list;          /* 4KB discard list */
626         struct list_head wait_list;             /* linked with issued discard bio */
627         int nr_discards;                        /* # of discards in the list */
628         int max_discards;                       /* max. discards to be issued */
629
630         /* for batched trimming */
631         unsigned int trim_sections;             /* # of sections to trim */
632
633         struct list_head sit_entry_set; /* sit entry set list */
634
635         unsigned int ipu_policy;        /* in-place-update policy */
636         unsigned int min_ipu_util;      /* in-place-update threshold */
637         unsigned int min_fsync_blocks;  /* threshold for fsync */
638
639         /* for flush command control */
640         struct flush_cmd_control *cmd_control_info;
641
642 };
```

```
507 struct f2fs_nm_info {
508         block_t nat_blkaddr;            /* base disk address of NAT */
509         nid_t max_nid;                  /* maximum possible node ids */
510         nid_t available_nids;           /* maximum available node ids */
511         nid_t next_scan_nid;            /* the next nid to be scanned */
512         unsigned int ram_thresh;        /* control the memory footprint */
513         unsigned int ra_nid_pages;      /* # of nid pages to be readaheaded */
514         unsigned int dirty_nats_ratio;  /* control dirty nats ratio threshold */
515
516         /* NAT cache management */
517         struct radix_tree_root nat_root;/* root of the nat entry cache */
518         struct radix_tree_root nat_set_root;/* root of the nat set cache */
519         struct rw_semaphore nat_tree_lock;      /* protect nat_tree_lock */
520         struct list_head nat_entries;   /* cached nat entry list (clean) */
521         unsigned int nat_cnt;           /* the # of cached nat entries */
522         unsigned int dirty_nat_cnt;     /* total num of nat entries in set */
523
524         /* free node ids management */
525         struct radix_tree_root free_nid_root;/* root of the free_nid cache */
526         struct list_head free_nid_list; /* a list for free nids */
527         spinlock_t free_nid_list_lock;  /* protect free nid list */
528         unsigned int fcnt;              /* the number of free node id */
529         struct mutex build_lock;        /* lock for build free nids */
530
531         /* for checkpoint */
532         char *nat_bitmap;               /* NAT bitmap pointer */
533         int bitmap_size;                /* bitmap size */
534 };
```

```
419 struct f2fs_inode_info {
420         struct inode vfs_inode;         /* serve a vfs inode */
421         unsigned long i_flags;          /* keep an inode flags for ioctl */
422         unsigned char i_advise;         /* use to give file attribute hints */
423         unsigned char i_dir_level;      /* use for dentry level for large dir */
424         unsigned int i_current_depth;   /* use only in directory structure */
425         unsigned int i_pino;            /* parent inode number */
426         umode_t i_acl_mode;             /* keep file acl mode temporarily */
427
428         /* Use below internally in f2fs*/
429         unsigned long flags;            /* use to pass per-file flags */
430         struct rw_semaphore i_sem;      /* protect fi info */
431         atomic_t dirty_pages;           /* # of dirty pages */
432         f2fs_hash_t chash;              /* hash value of given file name */
433         unsigned int clevel;            /* maximum level of given file name */
434         struct task_struct *task;       /* lookup and create consistency */
435         nid_t i_xattr_nid;              /* node id that contains xattrs */
436         unsigned long long xattr_ver;   /* cp version of xattr modification */
437         loff_t  last_disk_size;         /* lastly written file size */
438
439         struct list_head dirty_list;    /* dirty list for dirs and files */
440         struct list_head gdirty_list;   /* linked in global dirty list */
441         struct list_head inmem_pages;   /* inmemory pages managed by f2fs */
442         struct mutex inmem_lock;        /* lock for inmemory pages */
443         struct extent_tree *extent_tree;        /* cached extent_tree entry */
444         struct rw_semaphore dio_rwsem[2];/* avoid racing between dio and gc */
445 };
```

```
286 /*
287  * For INODE and NODE manager
288  */
289 /* for directory operations */
290 struct f2fs_dentry_ptr {
```

```
291         struct inode *inode;
292         const void *bitmap;
293         struct f2fs_dir_entry *dentry;
294         __u8 (*filename)[F2FS_SLOT_LEN];
295         int max;
296 };
```

```
663 /*
664  * The below are the page types of bios used in submit_bio().
665  * The available types are:
666  * DATA                 User data pages. It operates as async mode.
667  * NODE                 Node pages. It operates as async mode.
668  * META                 FS metadata pages such as SIT, NAT, CP.
669  * NR_PAGE_TYPE         The number of page types.
670  * META_FLUSH           Make sure the previous pages are written
671  *                      with waiting the bio's completion
672  * ...                  Only can be used with META.
673  */
674 #define PAGE_TYPE_OF_BIO(type)  ((type) > META ? META : (type))
675 enum page_type {
676         DATA,
677         NODE,
678         META,
679         NR_PAGE_TYPE,
680         META_FLUSH,
681         INMEM,          /* the below types are used by tracepoints only. */
682         INMEM_DROP,
683         INMEM_REVOKE,
684         IPU,
685         OPU,
686 };
```

```
708 enum inode_type {
709         DIR_INODE,                      /* for dirty dir inode */
710         FILE_INODE,                     /* for dirty regular/symlink inode */
711         DIRTY_META,                     /* for all dirtied inode metadata */
712         NR_INODE_TYPE,
713 };

715 /* for inner inode cache management */
716 struct inode_management {
717         struct radix_tree_root ino_root;        /* ino entry array */
718         spinlock_t ino_lock;                    /* for ino entry lock */
719         struct list_head ino_list;              /* inode list head */
720         unsigned long ino_num;                  /* number of entries */
721 };
722
723 /* For s_flag in struct f2fs_sb_info */
```

```
644 /*
645  * For superblock
646  */
647 /*
648  * COUNT_TYPE for monitoring
649  *
650  * f2fs monitors the number of several block types such as on-writeback,
651  * dirty dentry blocks, dirty node blocks, and dirty meta blocks.
652  */
653 enum count_type {
654         F2FS_DIRTY_DENTS,
655         F2FS_DIRTY_DATA,
656         F2FS_DIRTY_NODES,
657         F2FS_DIRTY_META,
658         F2FS_INMEM_PAGES,
659         F2FS_DIRTY_IMETA,
660         NR_COUNT_TYPE,
661 };
```

```
688 struct f2fs_io_info {
689         struct f2fs_sb_info *sbi;       /* f2fs_sb_info pointer */
690         enum page_type type;    /* contains DATA/NODE/META/META_FLUSH */
691         int op;                 /* contains REQ_OP_ */
692         int op_flags;           /* rq_flag_bits */
693         block_t new_blkaddr;    /* new block address to be written */
694         block_t old_blkaddr;    /* old block address before Cow */
695         struct page *page;      /* page to be written */
696         struct page *encrypted_page;    /* encrypted page */
697 };
698
699 #define is_read_io(rw) (rw == READ)
700 struct f2fs_bio_info {
701         struct f2fs_sb_info *sbi;       /* f2fs superblock */
702         struct bio *bio;                /* bios to merge */
703         sector_t last_block_in_bio;     /* last block number */
704         struct f2fs_io_info fio;        /* store buffered io info. */
705         struct rw_semaphore io_rwsem;   /* blocking op for bio */
706 };
```

```
564 /*
565  * For SIT manager
566  *
567  * By default, there are 6 active log areas across the whole main area.
568  * When considering hot and cold data separation to reduce cleaning overhead,
569  * we split 3 for data logs and 3 for node logs as hot, warm, and cold types,
570  * respectively.
571  * In the current design, you should not change the numbers intentionally.
572  * Instead, as a mount option such as active_logs=x, you can use 2, 4, and 6
573  * logs individually according to the underlying devices. (default: 6)
574  * Just in case, on-disk layout covers maximum 16 logs that consist of 8 for
575  * data and 8 for node logs.
576  */
577 #define NR_CURSEG_DATA_TYPE     (3)
578 #define NR_CURSEG_NODE_TYPE     (3)
579 #define NR_CURSEG_TYPE  (NR_CURSEG_DATA_TYPE +
NR_CURSEG_NODE_TYPE)
580
```

```
581 enum {
582         CURSEG_HOT_DATA = 0,    /* directory entry blocks */
583         CURSEG_WARM_DATA,       /* data blocks */
584         CURSEG_COLD_DATA,       /* multimedia or GCed data blocks */
585         CURSEG_HOT_NODE,        /* direct node blocks of directory files */
586         CURSEG_WARM_NODE,       /* direct node blocks of normal files */
587         CURSEG_COLD_NODE,       /* indirect node blocks */
588         NO_CHECK_TYPE,
589         CURSEG_DIRECT_IO,       /* to use for the direct IO path */
590 };
```

```
592 struct flush_cmd {
593         struct completion wait;
594         struct llist_node llnode;
595         int ret;
596 };
597
598 struct flush_cmd_control {
599         struct task_struct *f2fs_issue_flush;   /* flush thread */
600         wait_queue_head_t flush_wait_queue;     /* waiting queue for wake-up */
601         atomic_t submit_flush;                  /* # of issued flushes */
602         struct llist_head issue_list;           /* list for command issue */
603         struct llist_node *dispatch_list;       /* list for command dispatch */
604 };
```

```
536 /*
537  * this structure is used as one of function parameters.
538  * all the information are dedicated to a given direct node block determined
539  * by the data offset in a file.
540  */
541 struct dnode_of_data {
542         struct inode *inode;            /* vfs inode pointer */
543         struct page *inode_page;        /* its inode page, NULL is possible */
544         struct page *node_page;         /* cached direct node page */
545         nid_t nid;                      /* node id of the direct node block */
546         unsigned int ofs_in_node;       /* data offset in the node page */
547         bool inode_page_locked;         /* inode page is locked or not */
548         bool node_changed;              /* is node block changed */
549         char cur_level;                 /* level of hole node page */
550         char max_level;                 /* level of current page located */
551         block_t data_blkaddr;           /* block address of the node block */
552 };
```

```
370 /*
371  * This structure is taken from ext4_map_blocks.
372  *
373  * Note that, however, f2fs uses NEW and MAPPED flags for f2fs_map_blocks().
374  */
375 #define F2FS_MAP_NEW            (1 << BH_New)
376 #define F2FS_MAP_MAPPED         (1 << BH_Mapped)
377 #define F2FS_MAP_UNWRITTEN      (1 << BH_Unwritten)
378 #define F2FS_MAP_FLAGS          (F2FS_MAP_NEW | F2FS_MAP_MAPPED |\
379                                 F2FS_MAP_UNWRITTEN)
380
381 struct f2fs_map_blocks {
382         block_t m_pblk;
383         block_t m_lblk;
```

```
384         unsigned int m_len;
385         unsigned int m_flags;
386         pgoff_t *m_next_pgofs;          /* point next possible non-hole pgofs */
387 };
388
389 /* for flag in get_data_block */
390 #define F2FS_GET_BLOCK_READ             0
391 #define F2FS_GET_BLOCK_DIO              1
392 #define F2FS_GET_BLOCK_FIEMAP           2
393 #define F2FS_GET_BLOCK_BMAP             3
394 #define F2FS_GET_BLOCK_PRE_DIO          4
395 #define F2FS_GET_BLOCK_PRE_AIO          5
396
397 /*
398  * i_advise uses FADVISE_XXX_BIT. We can add additional hints later.
399  */
400 #define FADVISE_COLD_BIT        0x01
401 #define FADVISE_LOST_PINO_BIT   0x02
402 #define FADVISE_ENCRYPT_BIT     0x04
403 #define FADVISE_ENC_NAME_BIT    0x08
404
405 #define file_is_cold(inode)     is_file(inode, FADVISE_COLD_BIT)
406 #define file_wrong_pino(inode)  is_file(inode, FADVISE_LOST_PINO_BIT)
407 #define file_set_cold(inode)    set_file(inode, FADVISE_COLD_BIT)
408 #define file_lost_pino(inode)   set_file(inode, FADVISE_LOST_PINO_BIT)
409 #define file_clear_cold(inode)  clear_file(inode, FADVISE_COLD_BIT)
410 #define file_got_pino(inode)    clear_file(inode, FADVISE_LOST_PINO_BIT)
411 #define file_is_encrypt(inode)  is_file(inode, FADVISE_ENCRYPT_BIT)
412 #define file_set_encrypt(inode) set_file(inode, FADVISE_ENCRYPT_BIT)
413 #define file_clear_encrypt(inode) clear_file(inode, FADVISE_ENCRYPT_BIT)
414 #define file_enc_name(inode)    is_file(inode, FADVISE_ENC_NAME_BIT)
415 #define file_set_enc_name(inode) set_file(inode, FADVISE_ENC_NAME_BIT)
416
417 #define DEF_DIR_LEVEL           0
```

```
318 /*
319  * XATTR_NODE_OFFSET stores xattrs to one node block per file keeping -1
320  * as its node offset to distinguish from index node blocks.
321  * But some bits are used to mark the node block.
322  */
323 #define XATTR_NODE_OFFSET       ((((unsigned int)-1) << OFFSET_BIT_SHIFT) \
324                                 >> OFFSET_BIT_SHIFT)
325 enum {
326         ALLOC_NODE,                     /* allocate a new node page if needed */
327         LOOKUP_NODE,                    /* look up a node without readahead */
328         LOOKUP_NODE_RA,                 /*
329                                          * look up a node with readahead called
330                                          * by get_data_block.
331                                          */
332 };

334 #define F2FS_LINK_MAX   0xffffffff      /* maximum link count per file */
335
336 #define MAX_DIR_RA_PAGES        4       /* maximum ra pages of dir */
337
338 /* vector size for gang look-up from extent cache that consists of radix tree */
339 #define EXT_TREE_VEC_SIZE       64
340
341 /* for in-memory extent cache entry */
342 #define F2FS_MIN_EXTENT_LEN     64      /* minimum extent length */
343
344 /* number of extent info in extent cache we try to shrink */
345 #define EXTENT_CACHE_SHRINK_NUMBER      128
346
```

```
347 struct extent_info {
348         unsigned int fofs;              /* start offset in a file */
349         u32 blk;                        /* start block address of the extent */
350         unsigned int len;               /* length of the extent */
351 };
352
353 struct extent_node {
354         struct rb_node rb_node;         /* rb node located in rb-tree */
355         struct list_head list;          /* node in global extent list of sbi */
356         struct extent_info ei;          /* extent info */
357         struct extent_tree *et;         /* extent tree pointer */
358 };
359
360 struct extent_tree {
361         nid_t ino;                      /* inode number */
362         struct rb_root root;            /* root of extent info rb-tree */
363         struct extent_node *cached_en;  /* recently accessed extent node */
364         struct extent_info largest;     /* largested extent info */
365         struct list_head list;          /* to be used by sbi->zombie_list */
366         rwlock_t lock;                  /* protect extent info rb-tree */
367         atomic_t node_cnt;              /* # of extent node in rb-tree*/
368 };
```

```
235 #define F2FS_IOC_GETFLAGS               FS_IOC_GETFLAGS
236 #define F2FS_IOC_SETFLAGS               FS_IOC_SETFLAGS
237 #define F2FS_IOC_GETVERSION             FS_IOC_GETVERSION
238
239 #define F2FS_IOCTL_MAGIC                0xf5
240 #define F2FS_IOC_START_ATOMIC_WRITE     _IO(F2FS_IOCTL_MAGIC, 1)
241 #define F2FS_IOC_COMMIT_ATOMIC_WRITE    _IO(F2FS_IOCTL_MAGIC, 2)
242 #define F2FS_IOC_START_VOLATILE_WRITE   _IO(F2FS_IOCTL_MAGIC, 3)
243 #define F2FS_IOC_RELEASE_VOLATILE_WRITE _IO(F2FS_IOCTL_MAGIC, 4)
244 #define F2FS_IOC_ABORT_VOLATILE_WRITE   _IO(F2FS_IOCTL_MAGIC, 5)
245 #define F2FS_IOC_GARBAGE_COLLECT        _IO(F2FS_IOCTL_MAGIC, 6)
246 #define F2FS_IOC_WRITE_CHECKPOINT       _IO(F2FS_IOCTL_MAGIC, 7)
247 #define F2FS_IOC_DEFRAGMENT             _IO(F2FS_IOCTL_MAGIC, 8)
248 #define F2FS_IOC_MOVE_RANGE             _IOWR(F2FS_IOCTL_MAGIC, 9,      \
249                                         struct f2fs_move_range)
250
251 #define F2FS_IOC_SET_ENCRYPTION_POLICY
FS_IOC_SET_ENCRYPTION_POLICY
252 #define F2FS_IOC_GET_ENCRYPTION_POLICY
FS_IOC_GET_ENCRYPTION_POLICY
253 #define F2FS_IOC_GET_ENCRYPTION_PWSALT
FS_IOC_GET_ENCRYPTION_PWSALT
254
255 /*
256  * should be same as XFS_IOC_GOINGDOWN.
257  * Flags for going down operation used by FS_IOC_GOINGDOWN
258  */
259 #define F2FS_IOC_SHUTDOWN       _IOR('X', 125, __u32)   /* Shutdown */
260 #define F2FS_GOING_DOWN_FULLSYNC        0x0     /* going down with full sync */
261 #define F2FS_GOING_DOWN_METASYNC        0x1     /* going down with metadata
*/
262 #define F2FS_GOING_DOWN_NOSYNC          0x2     /* going down */
263 #define F2FS_GOING_DOWN_METAFLUSH       0x3     /* going down with meta flush
*/
264
265 #if defined(__KERNEL__) && defined(CONFIG_COMPAT)
266 /*
267  * ioctl commands in 32 bit emulation
268  */
269 #define F2FS_IOC32_GETFLAGS             FS_IOC32_GETFLAGS
270 #define F2FS_IOC32_SETFLAGS             FS_IOC32_SETFLAGS
271 #define F2FS_IOC32_GETVERSION           FS_IOC32_GETVERSION
272 #endif
```

```
274 struct f2fs_defragment {
275         u64 start;
276         u64 len;
277 };
278
279 struct f2fs_move_range {
280         u32 dst_fd;             /* destination fd */
281         u64 pos_in;             /* start position in src_fd */
282         u64 pos_out;            /* start position in dst_fd */
283         u64 len;                /* size to move */
284 };
```

```
139 struct cp_control {
140         int reason;
141         __u64 trim_start;
142         __u64 trim_end;
143         __u64 trim_minlen;
144         __u64 trimmed;
145 };
146
147 /*
148  * For CP/NAT/SIT/SSA readahead
149  */
150 enum {
151         META_CP,
152         META_NAT,
153         META_SIT,
154         META_SSA,
155         META_POR,
156 };
157
158 /* for the list of ino */
159 enum {
160         ORPHAN_INO,             /* for orphan ino list */
161         APPEND_INO,             /* for append ino list */
162         UPDATE_INO,             /* for update ino list */
163         MAX_INO_ENTRY,          /* max. list */
164 };
165
166 struct ino_entry {
167         struct list_head list;  /* list head */
168         nid_t ino;              /* inode number */
169 };
170
171 /* for the list of inodes to be GCed */
172 struct inode_entry {
173         struct list_head list;  /* list head */
174         struct inode *inode;    /* vfs inode pointer */
175 };
176
177 /* for the list of blockaddresses to be discarded */
178 struct discard_entry {
179         struct list_head list;  /* list head */
180         block_t blkaddr;        /* block address to be discarded */
181         int len;                /* # of consecutive blocks of the discard */
182 };
183
184 struct bio_entry {
185         struct list_head list;
186         struct bio *bio;
187         struct completion event;
188         int error;
189 };
190
191 /* for the list of fsync inodes, used only during recovery */
192 struct fsync_inode_entry {
193         struct list_head list;  /* list head */
```

```
194         struct inode *inode;    /* vfs inode pointer */
195         block_t blkaddr;        /* block address locating the last fsync */
196         block_t last_dentry;    /* block address locating the last dentry */
197 };
```

```
25 struct f2fs_gc_kthread {
26         struct task_struct *f2fs_gc_task;
27         wait_queue_head_t gc_wait_queue_head;
28
29         /* for gc sleep time */
30         unsigned int min_sleep_time;
31         unsigned int max_sleep_time;
32         unsigned int no_gc_sleep_time;
33
34         /* for changing gc mode */
35         unsigned int gc_idle;
36 };
37
38 struct gc_inode_list {
39         struct list_head ilist;
40         struct radix_tree_root iroot;
41 };
```

fs/f2fs/node.h

```
41 /* For flag in struct node_info */
42 enum {
43         IS_CHECKPOINTED,        /* is it checkpointed before? */
44         HAS_FSYNCED_INODE,      /* is the inode fsynced before? */
45         HAS_LAST_FSYNC,         /* has the latest node fsync mark? */
46         IS_DIRTY,               /* this nat entry is dirty? */
47 };
48
49 /*
50  * For node information
51  */
52 struct node_info {
53         nid_t nid;              /* node id */
54         nid_t ino;              /* inode number of the node's owner */
55         block_t blk_addr;       /* block address of the node */
56         unsigned char version;  /* version of the node */
57         unsigned char flag;     /* for node information bits */
58 };
59
60 struct nat_entry {
61         struct list_head list;  /* for clean or dirty nat list */
62         struct node_info ni;    /* in-memory node information */
63 };


137 enum mem_type {
138         FREE_NIDS,      /* indicates the free nid list */
139         NAT_ENTRIES,    /* indicates the cached nat entry */
140         DIRTY_DENTS,    /* indicates dirty dentry pages */
141         INO_ENTRIES,    /* indicates inode entries */
142         EXTENT_CACHE,   /* indicates extent cache */
143         BASE_CHECK,     /* check kernel status */
144 };
145
146 struct nat_entry_set {
147         struct list_head set_list;      /* link with other nat sets */
148         struct list_head entry_list;    /* link with dirty nat entries */
149         nid_t set;                      /* set number*/
150         unsigned int entry_cnt;         /* the # of nat entries in set */
151 };
152
153 /*
154  * For free nid mangement
155  */
156 enum nid_state {
157         NID_NEW,        /* newly added to free nid list */
158         NID_ALLOC       /* it is allocated */
159 };
160
161 struct free_nid {
162         struct list_head list;  /* for free node id list */
163         nid_t nid;              /* node id */
164         int state;              /* in use or not: NID_NEW or NID_ALLOC */
165 };
```

fs/f2fs/segment.h

```
149 /* for a function parameter to select a victim segment */
150 struct victim_sel_policy {
151         int alloc_mode;                 /* LFS or SSR */
152         int gc_mode;                    /* GC_CB or GC_GREEDY */
153         unsigned long *dirty_segmap;    /* dirty segment bitmap */
154         unsigned int max_search;        /* maximum # of segments to search */
155         unsigned int offset;            /* last scanned bitmap offset */
156         unsigned int ofs_unit;          /* bitmap search unit */
157         unsigned int min_cost;          /* minimum cost */
158         unsigned int min_segno;         /* segment # having min. cost */
159 };
160
161 struct seg_entry {
162         unsigned int type:6;            /* segment type like CURSEG_XXX_TYPE */
163         unsigned int valid_blocks:10;   /* # of valid blocks */
164         unsigned int ckpt_valid_blocks:10;      /* # of valid blocks last cp */
165         unsigned int padding:6;         /* padding */
166         unsigned char *cur_valid_map;   /* validity bitmap of blocks */
167         /*
168          * # of valid blocks and the validity bitmap stored in the the last
169          * checkpoint pack. This information is used by the SSR mode.
170          */
171         unsigned char *ckpt_valid_map;  /* validity bitmap of blocks last cp */
172         unsigned char *discard_map;
173         unsigned long long mtime;       /* modification time of the segment */
174 };
175
176 struct sec_entry {
177         unsigned int valid_blocks;      /* # of valid blocks in a section */
178 };
179
180 struct segment_allocation {
181         void (*allocate_segment)(struct f2fs_sb_info *, int, bool);
182 };

193 struct inmem_pages {
194         struct list_head list;
195         struct page *page;
196         block_t old_addr;               /* for revoking when fail to commit */
197 };
198
199 struct sit_info {
200         const struct segment_allocation *s_ops;
201
202         block_t sit_base_addr;          /* start block address of SIT area */
203         block_t sit_blocks;             /* # of blocks used by SIT area */
204         block_t written_valid_blocks;   /* # of valid blocks in main area */
205         char *sit_bitmap;               /* SIT bitmap pointer */
206         unsigned int bitmap_size;       /* SIT bitmap size */
207
208         unsigned long *tmp_map;                 /* bitmap for temporal use */
209         unsigned long *dirty_sentries_bitmap;   /* bitmap for dirty sentries */
210         unsigned int dirty_sentries;            /* # of dirty sentries */
211         unsigned int sents_per_block;           /* # of SIT entries per block */
212         struct mutex sentry_lock;               /* to protect SIT cache */
213         struct seg_entry *sentries;             /* SIT segment-level cache */
214         struct sec_entry *sec_entries;          /* SIT section-level cache */
215
216         /* for cost-benefit algorithm in cleaning procedure */
217         unsigned long long elapsed_time;        /* elapsed time after mount */
218         unsigned long long mounted_time;        /* mount time */
219         unsigned long long min_mtime;           /* min. modification time */
220         unsigned long long max_mtime;           /* max. modification time */
221 };
```

fs/f2fs/segment.h

```
223 struct free_segmap_info {
224         unsigned int start_segno;       /* start segment number logically */
225         unsigned int free_segments;     /* # of free segments */
226         unsigned int free_sections;     /* # of free sections */
227         spinlock_t segmap_lock;         /* free segmap lock */
228         unsigned long *free_segmap;     /* free segment bitmap */
229         unsigned long *free_secmap;     /* free section bitmap */
230 };
231
232 /* Notice: The order of dirty type is same with CURSEG_XXX in f2fs.h */
233 enum dirty_type {
234         DIRTY_HOT_DATA,         /* dirty segments assigned as hot data logs */
235         DIRTY_WARM_DATA,        /* dirty segments assigned as warm data logs */
236         DIRTY_COLD_DATA,        /* dirty segments assigned as cold data logs */
237         DIRTY_HOT_NODE,         /* dirty segments assigned as hot node logs */
238         DIRTY_WARM_NODE,        /* dirty segments assigned as warm node logs
*/
239         DIRTY_COLD_NODE,        /* dirty segments assigned as cold node logs */
240         DIRTY,                  /* to count # of dirty segments */
241         PRE,                    /* to count # of entirely obsolete segments */
242         NR_DIRTY_TYPE
243 };
244
245 struct dirty_seglist_info {
246         const struct victim_selection *v_ops;   /* victim selction operation */
247         unsigned long *dirty_segmap[NR_DIRTY_TYPE];
248         struct mutex seglist_lock;              /* lock for segment bitmaps */
249         int nr_dirty[NR_DIRTY_TYPE];            /* # of dirty segments */
250         unsigned long *victim_secmap;           /* background GC victims */
251 };
252
253 /* victim selection function for cleaning and SSR */
254 struct victim_selection {
255         int (*get_victim)(struct f2fs_sb_info *, unsigned int *,
256                                                 int, int, char);
257 };
258


259 /* for active log information */
260 struct curseg_info {
261         struct mutex curseg_mutex;              /* lock for consistency */
262         struct f2fs_summary_block *sum_blk;     /* cached summary block */
263         struct rw_semaphore journal_rwsem;      /* protect journal area */
264         struct f2fs_journal *journal;           /* cached journal info */
265         unsigned char alloc_type;               /* current allocation type */
266         unsigned int segno;                     /* current segment number */
267         unsigned short next_blkoff;             /* next block offset to write */
268         unsigned int zone;                      /* current zone number */
269         unsigned int next_segno;                /* preallocated segment */
270 };
271
272 struct sit_entry_set {
273         struct list_head set_list;      /* link with all sit sets */
274         unsigned int start_segno;       /* start segno of sits in set */
275         unsigned int entry_cnt;         /* the # of sit entries in set */
276 };
```