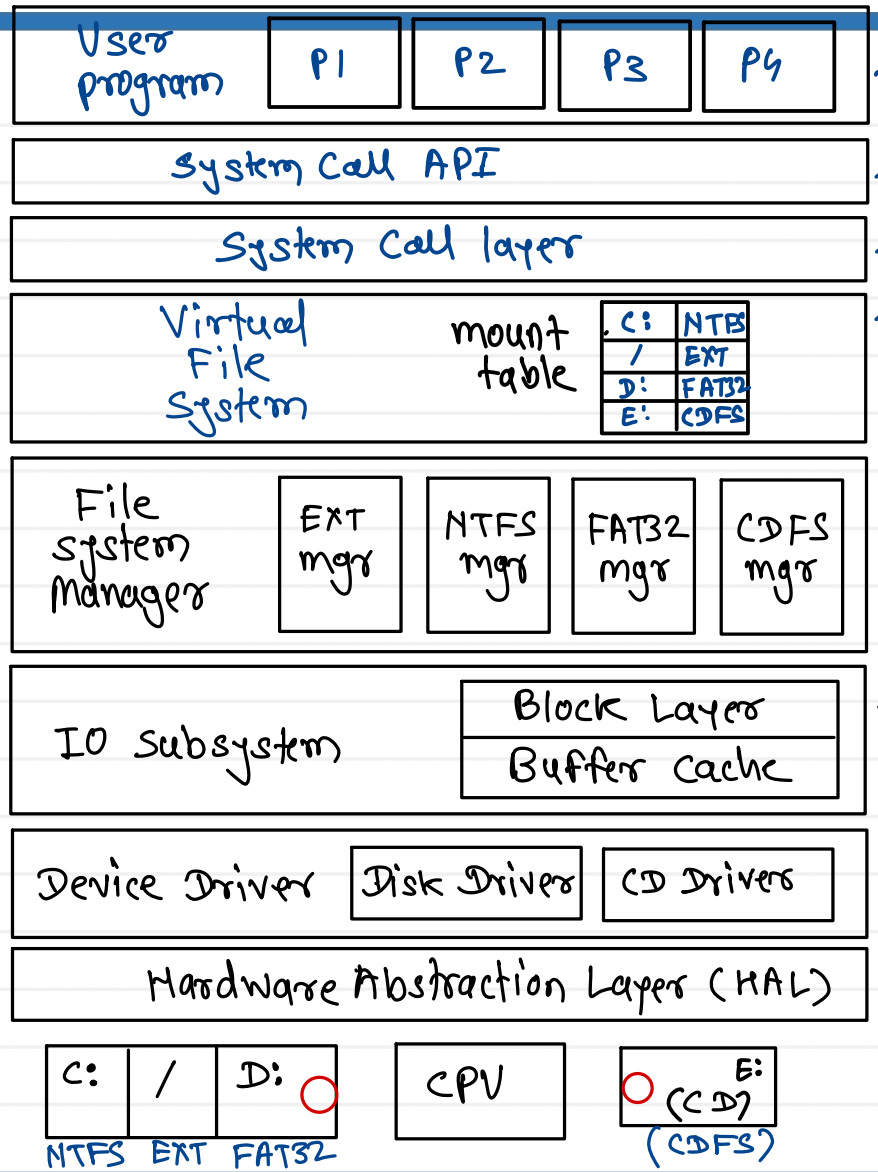
Sunbeam Institute of Information Technology
Pune and Karad

Module - Embedded Operating System

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Virtual file system



→ fopen(), fread(), fwrite(), fclose()

→ open(), read(), write(), close()

→ sys_open(), sys_read(), sys_write(), sys_close()

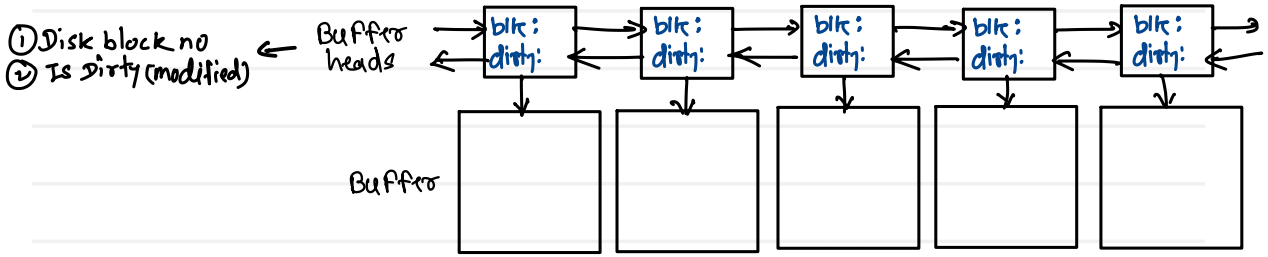
→ mount table : Partition : File system
VFS redirects fs request to respective file system manager
VFS defines many struct/object to deal with file systems
- superblock, file, inode, dentry
VFS also defines operations to be done on above structs
- superblock_ops, file_ops, inode_ops, dentry_ops

→ manages FS layouts on partitions
super block, inode list, data blocks
read/write

Linux : ext2/3/4, NTFS....
Windows : NTFS, CDFS, FAT

→ schedules h/w access requests (request queue)

→ cache the data of storage device



Disk blocks :
- manufacturer → sector = 512B
- File systems → FS Block = 1K, 2K, 4K, 8K..... 2M

VFS structures

```
struct super_block {
        struct list_head        s_list;         /* list of all superblocks */
        dev_t                   s_dev;          /* identifier */
        unsigned long           s_blocksize;    /* block size in bytes */
        unsigned long           s_old_blocksize; /* old block size in bytes */
        unsigned char           s_blocksize_bits; /* block size in bits */
        unsigned char           s_dirt;         /* dirty flag */
        unsigned long long      s_maxbytes;     /* max file size */
        struct file_system_type s_type;         /* filesystem type */
        struct super_operations s_op;           /* superblock methods */
        struct dquot_operations *dq_op;         /* quota methods */
        struct quotactl_ops     *s_qcop;        /* quota control methods */
        struct export_operations *s_export_op;  /* export methods */
        unsigned long           s_flags;        /* mount flags */
        unsigned long           s_magic;        /* filesystem's magic number */
        struct dentry           *s_root;        /* directory mount point */
        struct rw_semaphore     s_umount;       /* unmount semaphore */
        struct semaphore        s_lock;         /* superblock semaphore */
        int                     s_count;        /* superblock ref count */
        int                     s_syncing;      /* filesystem syncing flag */
        int                     s_need_sync_fs; /* not-yet-synced flag */
        atomic_t                s_active;       /* active reference count */
        void                    *s_security;    /* security module */
        struct list_head        s_dirty;        /* list of dirty inodes */
        struct list_head        s_io;           /* list of writebacks */
        struct hlist_head       s_anon;         /* anonymous dentries */
        struct list_head        s_files;        /* list of assigned files */
        struct block_device     *s_bdev;        /* associated block device */
        struct list_head        s_instances;    /* instances of this fs */
        struct quota_info       s_dquot;        /* quota-specific options */
        char                    s_id[32];       /* text name */
        void                    *s_fs_info;     /* filesystem-specific info */
        struct semaphore        s_vfs_rename_sem; /* rename semaphore */
};
```

```
struct super_operations {
        struct inode *(*alloc_inode) (struct super_block *sb);
        void (*destroy_inode) (struct inode *);
        void (*read_inode) (struct inode *);
        void (*dirty_inode) (struct inode *);
        void (*write_inode) (struct inode *, int);
        void (*put_inode) (struct inode *);
        void (*drop_inode) (struct inode *);
        void (*delete_inode) (struct inode *);
        void (*put_super) (struct super_block *);
        void (*write_super) (struct super_block *);
        int (*sync_fs) (struct super_block *, int);
        void (*write_super_lockfs) (struct super_block *);
        void (*unlockfs) (struct super_block *);
        int (*statfs) (struct super_block *, struct statfs *);
        int (*remount_fs) (struct super_block *, int *, char *);
        void (*clear_inode) (struct inode *);
        void (*umount_begin) (struct super_block *);
        int (*show_options) (struct seq_file *, struct vfsmount *);
};
```

VFS structures

```
struct inode {
        struct hlist_node       i_hash;         /* hash list */
        struct list_head        i_list;         /* list of inodes */
        struct list_head        i_dentry;       /* list of dentries */
        unsigned long           i_ino;          /* inode number */
        atomic_t                i_count;        /* reference counter */
        umode_t                 i_mode;         /* access permissions */
        unsigned int            i_nlink;        /* number of hard links */
        uid_t                   i_uid;          /* user id of owner */
        gid_t                   i_gid;          /* group id of owner */
        kdev_t                  i_rdev;         /* real device node */
        loff_t                  i_size;         /* file size in bytes */
        struct timespec         i_atime;        /* last access time */
        struct timespec         i_mtime;        /* last modify time */
        struct timespec         i_ctime;        /* last change time */
        unsigned int            i_blkbits;      /* block size in bits */
        unsigned long           i_blksize;      /* block size in bytes */
        unsigned long           i_version;      /* version number */
        unsigned long           i_blocks;       /* file size in blocks */
        unsigned short          i_bytes;        /* bytes consumed */
        spinlock_t              i_lock;         /* spinlock */
        struct rw_semaphore     i_alloc_sem;    /* nests inside of i_sem */
        struct semaphore        i_sem;          /* inode semaphore */
        struct inode_operations *i_op;          /* inode ops table */
        struct file_operations  *i_fop;         /* default inode ops */
        struct super_block      *i_sb;          /* associated superblock */
        struct file_lock        *i_flock;       /* file lock list */
        struct address_space    *i_mapping;     /* associated mapping */
        struct address_space    i_data;         /* mapping for device */
        struct dquot            *i_dquot[MAXQUOTAS]; /* disk quotas for inode */
        struct list_head        i_devices;      /* list of block devices */
        struct pipe_inode_info  *i_pipe;        /* pipe information */
        struct block_device     *i_bdev;        /* block device driver */
        unsigned long           i_dnotify_mask; /* directory notify mask */
        struct dnotify_struct   *i_dnotify;     /* dnotify */
        unsigned long           i_state;        /* state flags */
        unsigned long           dirtied_when;   /* first dirtying time */
        unsigned int            i_flags;        /* filesystem flags */
        unsigned char           i_sock;         /* is this a socket? */
        atomic_t                i_writecount;   /* count of writers */
        void                    *i_security;    /* security module */
        __u32                   i_generation;   /* inode version number */
        union {
                void            *generic_ip;    /* filesystem-specific info */
        } u;
};
```

```
struct inode_operations {
        int (*create) (struct inode *, struct dentry *,int);
        struct dentry * (*lookup) (struct inode *, struct dentry *);
        int (*link) (struct dentry *, struct inode *, struct dentry *);
        int (*unlink) (struct inode *, struct dentry *);
        int (*symlink) (struct inode *, struct dentry *, const char *);
        int (*mkdir) (struct inode *, struct dentry *, int);
        int (*rmdir) (struct inode *, struct dentry *);
        int (*mknod) (struct inode *, struct dentry *, int, dev_t);
        int (*rename) (struct inode *, struct dentry *,
                        struct inode *, struct dentry *);
        int (*readlink) (struct dentry *, char *, int);
        int (*follow_link) (struct dentry *, struct nameidata *);
        int (*put_link) (struct dentry *, struct nameidata *);
        void (*truncate) (struct inode *);
        int (*permission) (struct inode *, int);
        int (*setattr) (struct dentry *, struct iattr *);
        int (*getattr) (struct vfsmount *, struct dentry *, struct kstat *);
        int (*setxattr) (struct dentry *, const char *,
                         const void *, size_t, int);
        ssize_t (*getxattr) (struct dentry *, const char *, void *, size_t);
        ssize_t (*listxattr) (struct dentry *, char *, size_t);
        int (*removexattr) (struct dentry *, const char *);
};
```

VFS structures

```
struct dentry {
        atomic_t                d_count;      /* usage count */
        unsigned long           d_vfs_flags;  /* dentry cache flags */
        spinlock_t              d_lock;       /* per-dentry lock */
        struct inode            *d_inode;     /* associated inode */
        struct list_head        d_lru;        /* unused list */
        struct list_head        d_child;      /* list of dentries within */
        struct list_head        d_subdirs;    /* subdirectories */
        struct list_head        d_alias;      /* list of alias inodes */
        unsigned long           d_time;       /* revalidate time */
        struct dentry_operations *d_op;       /* dentry operations table */
        struct super_block      *d_sb;        /* superblock of file */
        unsigned int            d_flags;      /* dentry flags */
        int                     d_mounted;    /* is this a mount point? */
        void                    *d_fsdata;    /* filesystem-specific data */
        struct rcu_head         d_rcu;        /* RCU locking */
        struct dcookie_struct   *d_cookie;    /* cookie */
        struct dentry           *d_parent;    /* dentry object of parent */
        struct qstr             d_name;       /* dentry name */
        struct hlist_node       d_hash;       /* list of hash table entries */
        struct hlist_head       *d_bucket;    /* hash bucket */
        unsigned char           d_iname[DNAME_INLINE_LEN_MIN]; /* short name */
};
```

```
struct dentry_operations {
        int (*d_revalidate) (struct dentry *, int);
        int (*d_hash) (struct dentry *, struct qstr *);
        int (*d_compare) (struct dentry *, struct qstr *, struct qstr *);
        int (*d_delete) (struct dentry *);
        void (*d_release) (struct dentry *);
        void (*d_iput) (struct dentry *, struct inode *);
};
```

VFS structures

```
struct file {
        struct list_head        f_list;         /* list of file objects */
        struct dentry           *f_dentry;      /* associated dentry object */
        struct vfsmount         *f_vfsmnt;      /* associated mounted fs */
        struct file_operations  *f_op;          /* file operations table */
        atomic_t                f_count;        /* file object's usage count */
        unsigned int            f_flags;        /* flags specified on open */
        mode_t                  f_mode;         /* file access mode */
        loff_t                  f_pos;          /* file offset (file pointer) */
        struct fown_struct      f_owner;        /* owner data for signals */
        unsigned int            f_uid;          /* user's UID */
        unsigned int            f_gid;          /* user's GID */
        int                     f_error;        /* error code */
        struct file_ra_state    f_ra;           /* read-ahead state */
        unsigned long           f_version;      /* version number */
        void                    *f_security;    /* security module */
        void                    *private_data;  /* tty driver hook */
        struct list_head        f_ep_links;     /* list of eventpoll links */
        spinlock_t              f_ep_lock;      /* eventpoll lock */
        struct address_space    *f_mapping;     /* page cache mapping */
};
```

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char *, size_t, loff_t *);
        ssize_t (*aio_read) (struct kiocb *, char *, size_t, loff_t);
        ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
        ssize_t (*aio_write) (struct kiocb *, const char *, size_t, loff_t);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int);
        int (*aio_fsync) (struct kiocb *, int);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*readv) (struct file *, const struct iovec *,
                          unsigned long, loff_t *);
        ssize_t (*writev) (struct file *, const struct iovec *,
                           unsigned long, loff_t *);
        ssize_t (*sendfile) (struct file *, loff_t *, size_t,
                             read_actor_t, void *);
        ssize_t (*sendpage) (struct file *, struct page *, int,
                             size_t, loff_t *, int);
        unsigned long (*get_unmapped_area) (struct file *, unsigned long,
                                            unsigned long, unsigned long,
                                            unsigned long);
        int (*check_flags) (int flags);
        int (*dir_notify) (struct file *filp, unsigned long arg);
        int (*flock) (struct file *filp, int cmd, struct file_lock *fl);
};
```

VFS structures

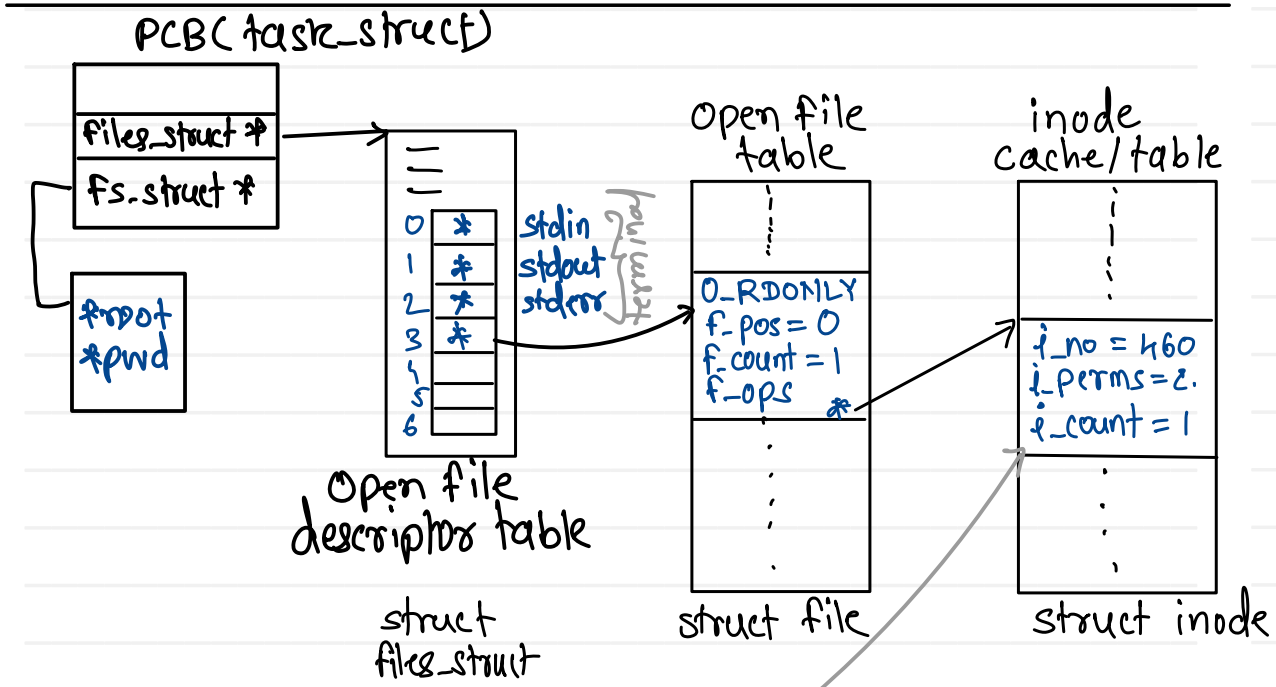
```
struct file_system_type {
        const char              *name;     /* filesystem's name */
        struct subsystem        subsys;    /* sysfs subsystem object */
        int                     fs_flags;  /* filesystem type flags */

        /* the following is used to read the superblock off the disk */
        struct super_block      *(*get_sb) (struct file_system_type *, int,
                                            char *, void *);

        /* the following is used to terminate access to the superblock */
        void                    (*kill_sb) (struct super_block *);

        struct module           *owner;    /* module owning the filesystem */
        struct file_system_type *next;     /* next file_system_type in list */
        struct list_head        fs_supers; /* list of superblock objects */
};
```

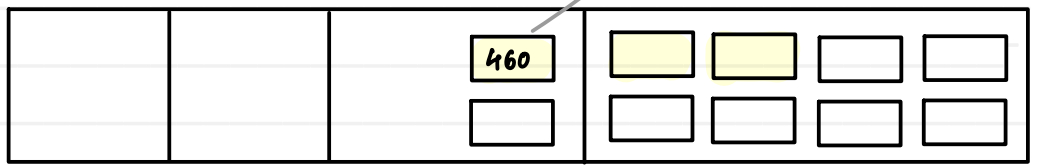
```
struct files_struct {
        atomic_t    count;              /* structure's usage count */
        spinlock_t  file_lock;          /* lock protecting this structure */
        int         max_fds;            /* maximum number of file objects */
        int         max_fdset;          /* maximum number of file descriptors */
        int         next_fd;            /* next file descriptor number */
        struct file **fd;               /* array of all file objects */
        fd_set      *close_on_exec;     /* file descriptors to close on exec() */
        fd_set      *open_fds;           /* pointer to open file descriptors */
        fd_set      close_on_exec_init; /* initial files to close on exec() */
        fd_set      open_fds_init;      /* initial set of file descriptors */
        struct file *fd_array[NR_OPEN_DEFAULT]; /* default array of file objects */
};
```

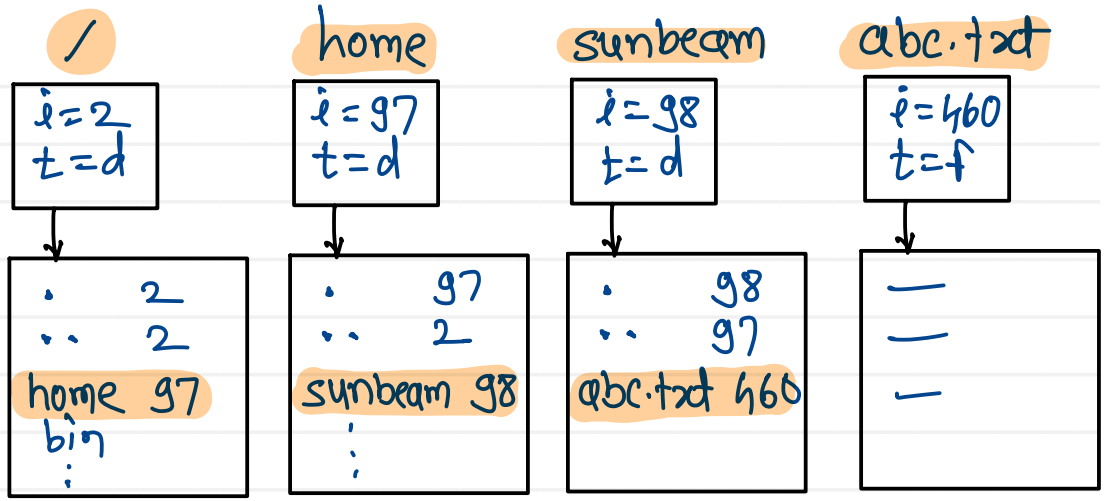
```
struct fs_struct {
        atomic_t        count;      /* structure usage count */
        rwlock_t        lock;       /* lock protecting structure */
        int             umask;      /* default file permissions*/
        struct dentry   *root;      /* dentry of the root directory */
        struct dentry   *pwd;       /* dentry of the current directory */
        struct dentry   *altroot;   /* dentry of the alternative root */
        struct vfsmount *rootmnt;   /* mount object of the root directory */
        struct vfsmount *pwdmnt;    /* mount object of the current directory */
        struct vfsmount *altrootmnt; /* mount object of the alternative root */
};
```

open() system call

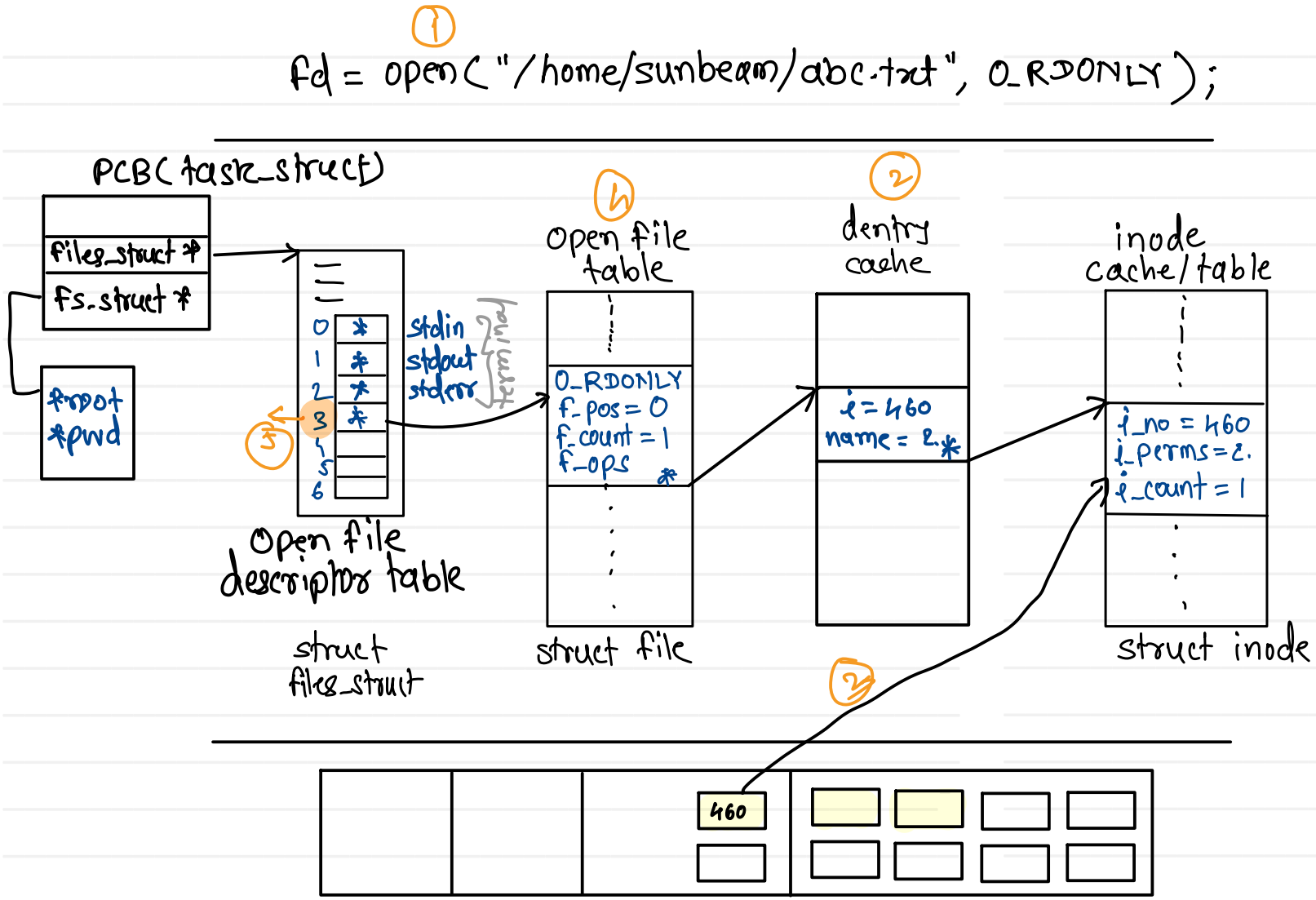
```
fd = open("/home/sunbeam/abc.txt", O_RDONLY);
```





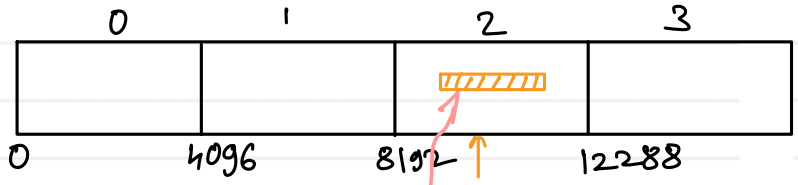


open() system call

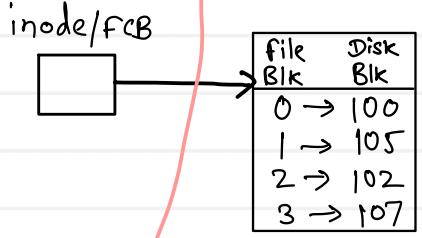


read() system call

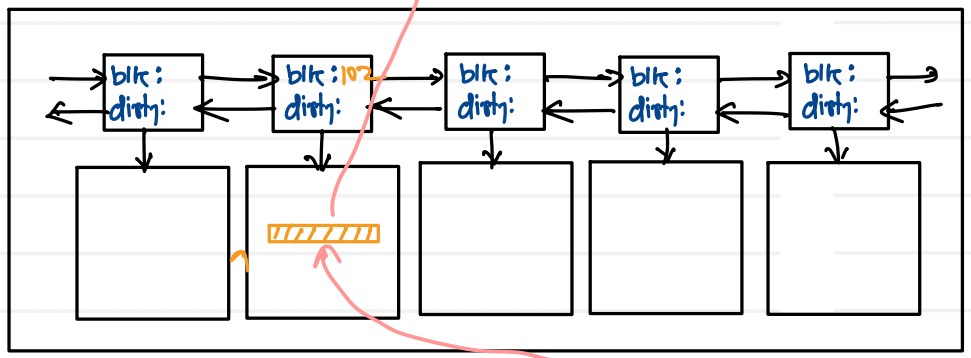
VFS (logical file sys)



FS mgr (Physical file sys)

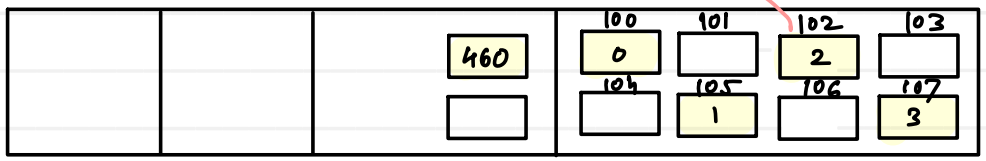


IO Subsystem



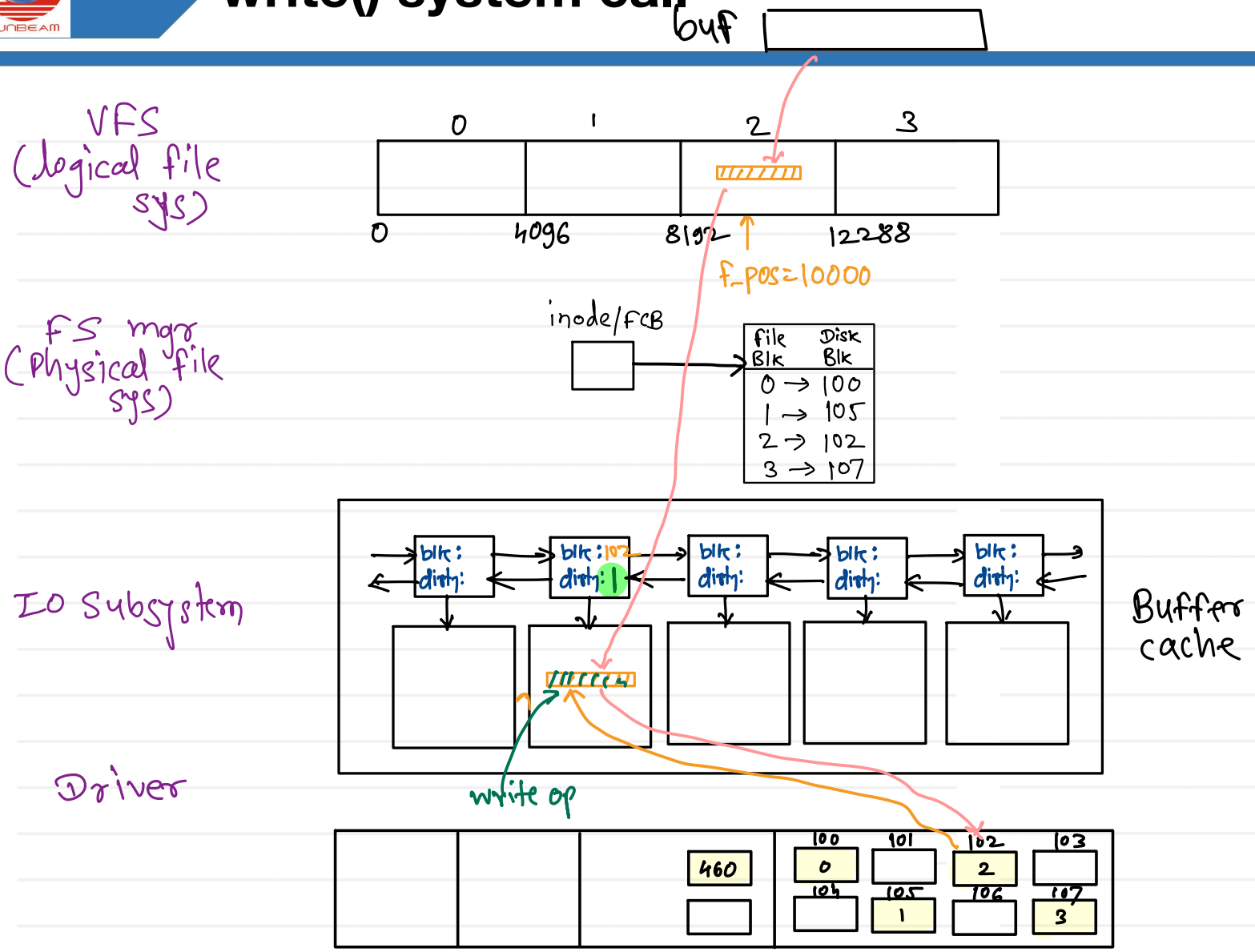
Buffer cache

Driver




```
read( fd, buf , sizeof(buf) )
                      100
        ↓
sys_read( fd , buffer, length)
        ↓
vfs_read( file, buffer, length, inode)
        ↓
ext3_read (file, inode, file_block)
        ↓
disk_read( disk_device , disk_block)
```

write() system call



(fd, buf, sizeof(buf))
100

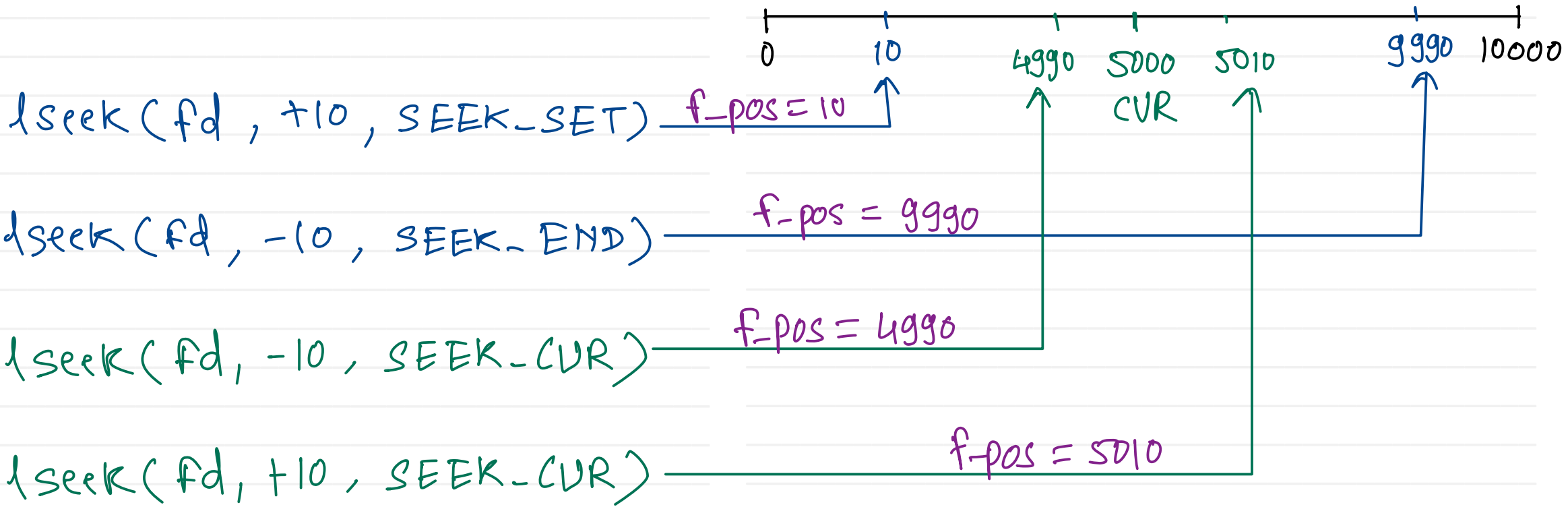
sys_write (fd, buffer, length)

vfs_write (file, buffer, length, inode)

ext3_write (file, inode, file_block)

disk_write (disk_device, disk_block)

lseek() system call



Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com