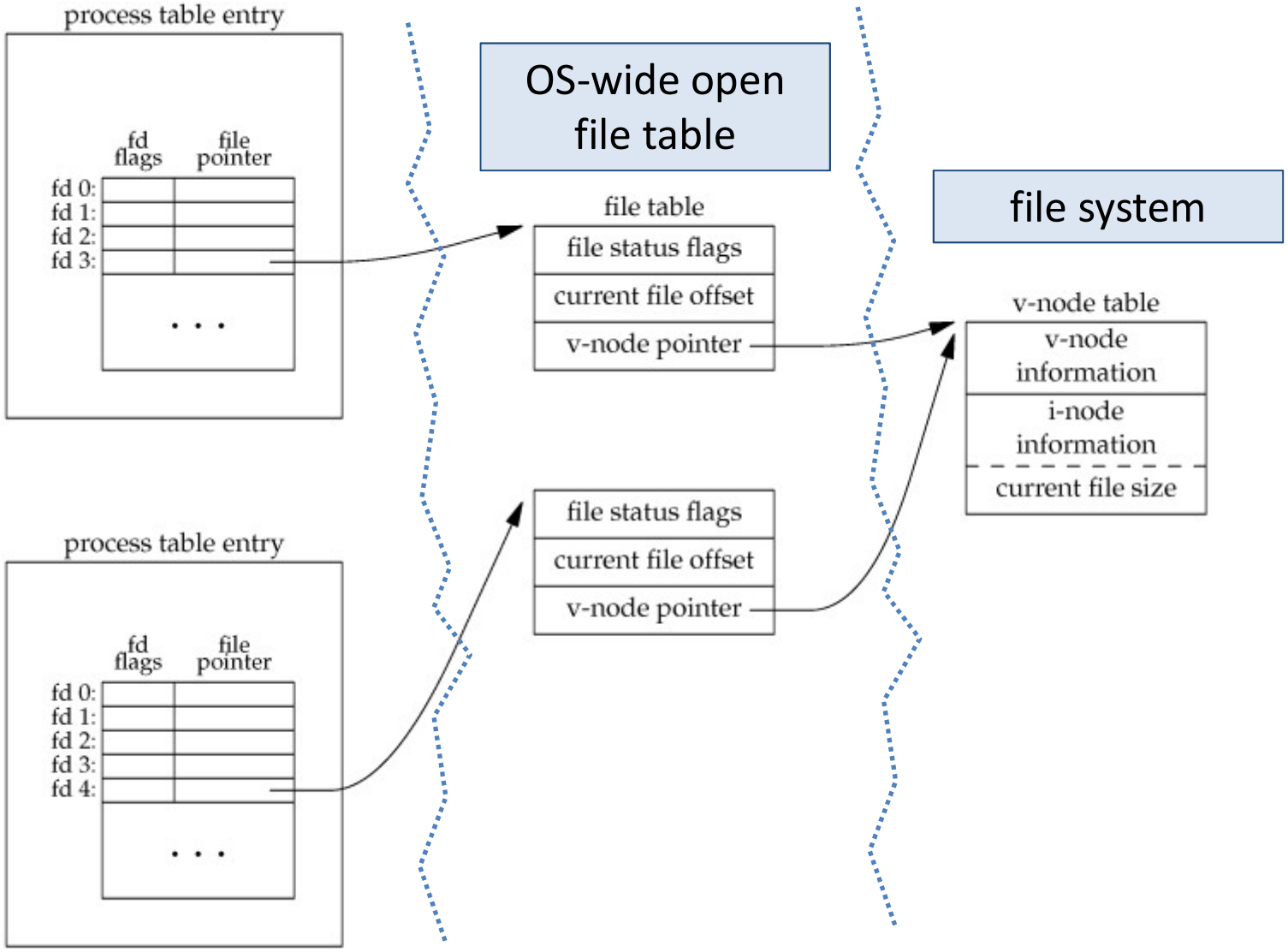
System Programming

3. File IO (2): System Call

Seung-Ho Lim

Dept. of Computer & Electronic Systems Eng.

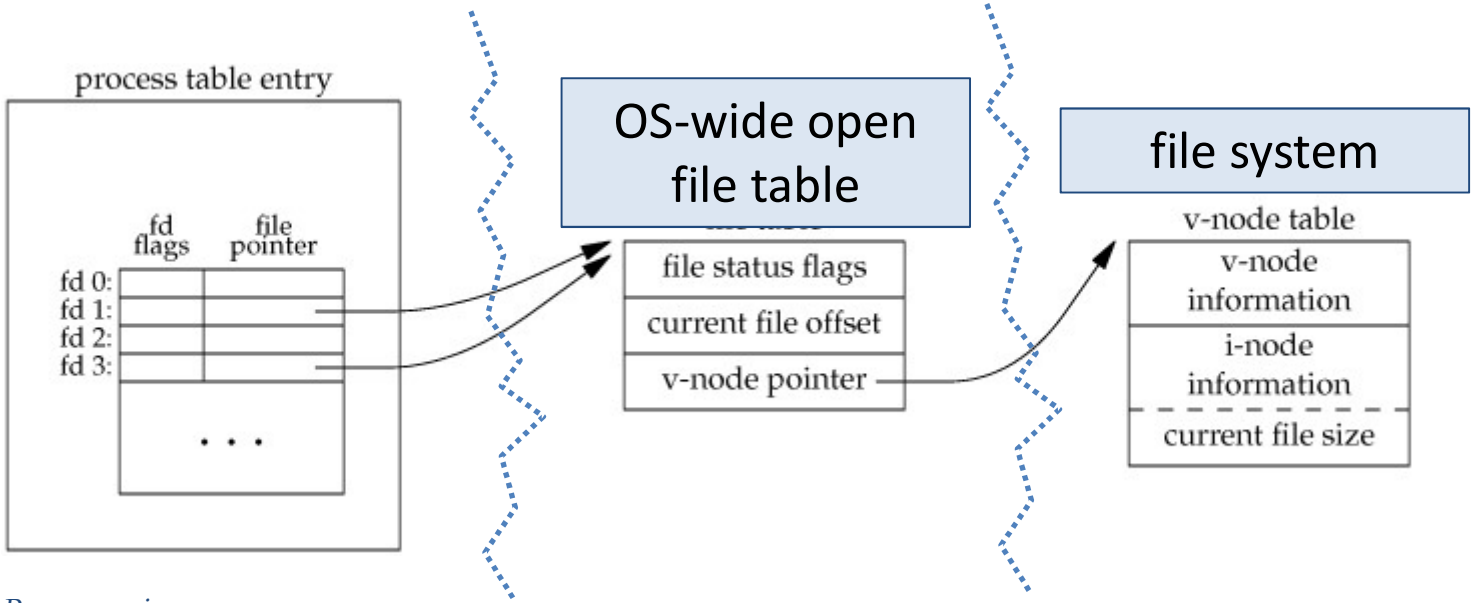
File descriptors & File table



Duplication of file descriptor (1)

```
#include <unistd.h>
int dup(int fd);
```

- parameters
  - *fd* : file descriptor to duplicate
- return
  - newly duplicated file descriptor if OK
  - -1 on error



Duplication of file descriptor (2)

```
#include <unistd.h>
int dup2(int fd1, int fd2);
```

- parameters
  - *fd1* : source file descriptor
  - *fd2* : destination file descriptor
- return
  - copied file descriptor (should be same as fd2) if OK
  - -1 on error
- note
  - functionally same as dup except that dup2 designates destination file descriptor (fd2) the user wants

I/O redirection example (1)

io-redir.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
	int backup_des, stdout_des, ofdes;

	stdout_des = fileno(stdout);
	backup_des = dup(stdout_des);
```

I/O redirection example (2)

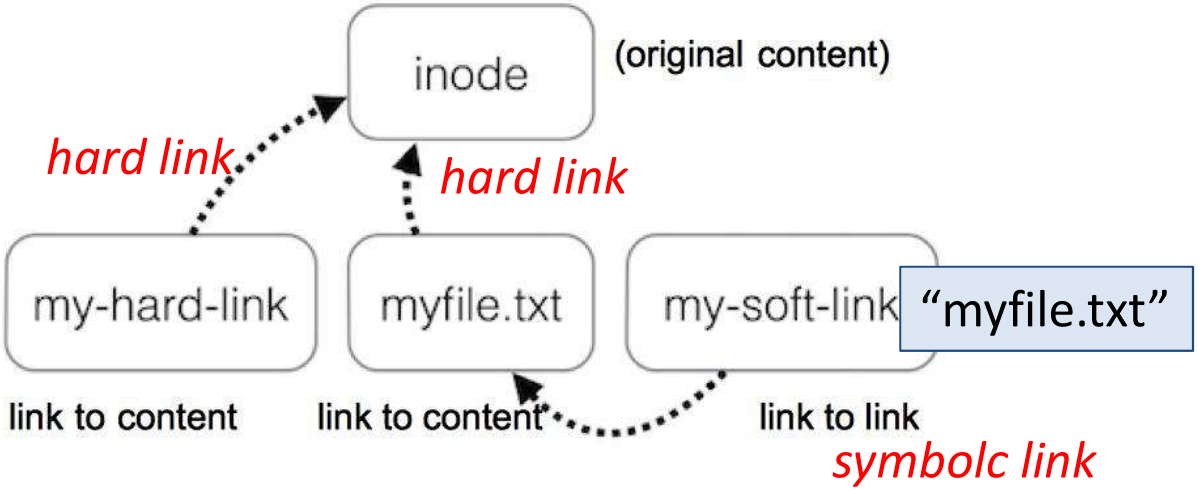
io-redir.c

```
        printf("Hello, world! (1)\n");

        ofdes = open("test.txt", O_WRONLY|O_CREAT|O_TRUNC,
                S_IRUSR|S_IWUSR);

        dup2(ofdes, stdout_des);
        printf("Hello, world! (2)\n");

        dup2(backup_des, stdout_des);
        printf("Hello, world! (3)\n");

        close(ofdes);
}
```

Link (1)

- Symbolic link
  - also called *soft link*
  - a ***file*** which records a path name to a target file
  - if the target file is removed, the link is not valid any longer, but the symbolic link file still exists
- Hard link
  - another link which points to an existing inode which is already used by another file
  - an inode can be shared by two or more filenames (hardlinks)
  - once a file is update, the update can be seen in all the hardlink files
  - though a file is removed, another hardlink can access the file

Link (2)



- Linux command
  - symbolic link:

    ```
    $ ln -s original_file   symbolic_link_name
    ```
  - hard link:

    ```
    $ ln original_file   hard_link_name
    ```

Hard link

```
#include <unistd.h>

int link(const char *existing, const char *new_link);
```

- parameters
  - *existing* : original file name
  - *new_link* : new link name which will share the inode of existing file
- return
  - 0 if OK
  - -1 on error

Hard link example (1)

hlink-ex.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char *argv[])
{       if(argc != 3) {
                perror("argument error");
                return 1;
        }
        if (link(argv[1], argv[2]) < 0) {
                perror("link fail");
                return 2;
        }
}
```

Hard link example (2)

- *Run & Results*

```
$ ls -l my*
drwxr--r-x 3  oskernel  oskernel  512  Jul 9 21:58  .
-rw-r--r-  1  oskernel  oskernel   15  Jul 9 21:58  myfile

$ ./a.out  myfile  myhardlink

$ ls -l my*
drwxr-xr-x 3   oskernel  oskernel  512  Jul 9 22:01   .
-rw-r--r-- 2   oskernel  oskernel   15  Jul 9 22:01   myfile
-rw-r--r-- 2   oskernel  oskernel   15  Jul 9 22:01   myhardlink

$
```

Symbolic link

```
#include <unistd.h>

int symlink(const char *existing, const char *link_name);
```

- parameters
  - *existing* : original file name
  - *link_name* : link name which points the existing file
- return
  - 0 if OK
  - -1 on error

Symbolic link example (1)

symlink-ex.c

```
#include <unistd.h>

int main(int argc, char *argv[])
{
	if(argc != 3) {
		perror("argument error");
		return 1;
	}
	if (symlink(argv[1], argv[2]) < 0) {
		perror("symlink fail");
		return 2;
	}
}
```

Symbolic link example (2)

- *Run & Results*

```
$ ls -l my*
-rw-r--r-   1  oskernel  oskernel    22    Jul 7 22:21    myfile

$ ./a.out myfile mylink

$ ls -l my*
-rw-r--r--  1  oskernel  oskernel    22    Jul 7 22:21    myfile
lrwxrwxrwx  1  oskernel  oskernel     6    Jul 9 22:18    mylink
```

- *What does the file "mylink" contain?*
  - *"myfile" → thus, the file size is 6 bytes.*

system calls and symbolic links

system calls which does NOT follow a symbolic link	system calls which follow a symbolic link
lchown, lstat, remove, readlink, rename, unlink	accessm chdir, chmod, chown, creat, exec, link, mkdir, mkfifo, mknod, open, opendir, pathconf, stat, truncate

these system calls handle a symbolic link itself as a FILE!

Following a link (1)

```
#include <unistd.h>

int readlink(const char *path, void *buf, size_t bufsize);
```

- parameters
  - *path* : link name
  - *buf* : buffer address (original file's name)
  - *bufsize* : buffer size
- return
  - number of bytes read if OK
  - -1 on error

Following a link example (1)

readlink-ex.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>

#define BUFFER_SIZE 100

int main(int argc, char *argv[])
{
          char buf[BUFFER_SIZE];
          int read_size = 0;

          if(argc != 2) {
                    perror("argument error");
                    return 1;
          }
          if ((read_size = readlink(argv[1], buf, BUFFER_SIZE)) < 0) {
                    perror("readlink");
                    return 2;
          }

          buf[read_size] = '\0' ;
          printf("%s\n", buf);
}
```

Following a link example (2)

Run & Results

```
$ ls  -ld  myfile  mylink
-rw-r--r-- 1  oskernel   oskernel   22   Jul 7 22:21   myfile
lrwxrwxrwx 1  oskernel   oskernel    6   Jul 7 22:18   mylink -> myfile

$ ./a.out mylink
myfile

$
```

File information retrieval (1)

```
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>

int stat(const char *path, struct stat *buf);
```

- parameters
  - *path* : file path name
  - *buf* : address of `struct stat` (which contains a file information)
- return
  - 0 if OK
  - -1 on error

File information retrieval (2)

```
int lstat(const char *path, struct stat *buf);
```

- basically, same as stat(), but
  - if the file is a symbolic link, retrieve the information of the link file itself (does not follow the link)
- parameters
  - *path* : file path name
  - *buf* : address of `struct stat` (which contains a file information)
- return
  - 0 if OK
  - -1 on error

File information retrieval (3)

```
int fstat(int fd, struct stat *buf)
```

- parameters
  - *fd* : file descriptor
  - *buf* : address of `struct stat` (which contains a file information)
- return
  - 0 if OK
  - -1 on error

struct stat *fields*

```
struct stat    {
        dev_t   st_dev;         // device
        ino_t   st_ino;         // i-node #
        mode_t  st_mode;        // access mode
        nlink_t st_nlink;       // number of hard links
        uid_t   st_uid;         // owner id
        gid_t   st_gid;         // group owner
        dev_t   st_rdev;        // device type (if inode device)
        off_t   st_size;        // total size of file
        long    st_blksize;     // block size for I/O
        long    st_blocks;      // number of blocks allocated
        time_t  st_atime;       // time of last access
        time_t  st_mtime;       // time of last modification
        time_t  st_ctime;       // time of last change (including
                                // ownership change)
};
```

Macros for struct stat

Macro	Functions
S_ISREG(st_mode)	return true if the file is *regular file*
S_ISDIR(st_mode)	return true if the file is *directory*
S_ISCHR(st_mode)	return true if the file is *character device file*
S_ISBLK(st_mode)	return true if the file is *block device file*
S_ISFIFO(st_mode)	return true if the file is *FIFO file*
S_ISLNK(st_mode)	return true if the file is *link file*
S_ISCOCK(st_mode)	return true if the file is *socket file*

File stat example (1)

fstat-ex.c

```
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
        struct stat statbuf;

        if(argc != 3) {
                perror("argument error");
                return 1;
        }
        if (!strcmp(argv[1], "stat")) {
                if (stat(argv[2], &statbuf) < 0) {
                        perror("stat");
                        return 2;
                }
        }
```

File stat example (2)

fstat-ex.c

```
        else if (!strcmp(argv[1], "fstat")) {
                int filedes = open(argv[2], O_RDWR);
                if (fstat(filedes, &statbuf) < 0) {
                        perror("stat");
                        return 3;
                }
        }
        else if(!strcmp(argv[1], "lstat")) {
                if (lstat(argv[2], &statbuf) < 0) {
                        perror("lstat");
                        return 4;
                }
        }
        if(S_IREG(statbuf.st_mode))
                printf("%s is Regular File\n", argv[2]);
        if(S_ISDIR(statbuf.st_mode))
                printf("%s is Directory\n", argv[2]);
        if(S_ISLNK(statbuf.st_mode))
                printf("%s is Link File\n", argv[2]);
}
```

File stat example (3)

- Run & Results

```
$ ls -ld mydir myfile mylink
drwxr-xr-x 2    root root   512        Jul 9 19 : 59     mydir
-rw-r--r-- 1    root root   26         Jul 2 23 : 41     myfile
lrwxrwxrwx 1    root root   12         Jul 9 19 : 59     mylink ->mydir/myfile

$ ./a.out stat myfile
myfile is Regular File

$ ./a.out stat mydir
mydir is Directory

$ ./a.out stat mylink
mylink is Requlat File

$ ls -l mydir
-rwxrwxrwx 1 peace peace 0 Jul 9 19 : 59 myfile

$ ./a.out lstat mylink
Mylink is Link File

$ ./a.out fstat mylink
mylink is Regular File
```