

CS241 System Programming

File System IV

Klara Nahrstedt

Lecture 23

3/15/2006



Content

- Inheritance of file descriptors
- Filters and Redirection
- File Control
- File System Navigation
- Directory Access
- Summary

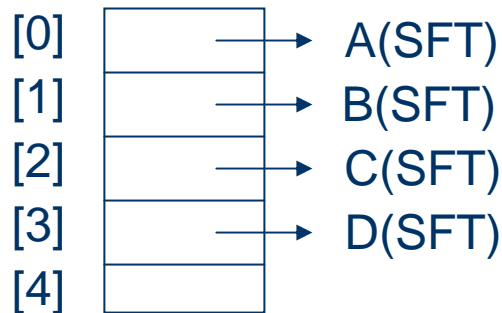
Administrative

- R&R: Ch 4 pp92-135
- R&R: Ch 5 pp145-158
- MP3 is posted, due April 3, 2006
- Quiz 6 is March 17, 2006

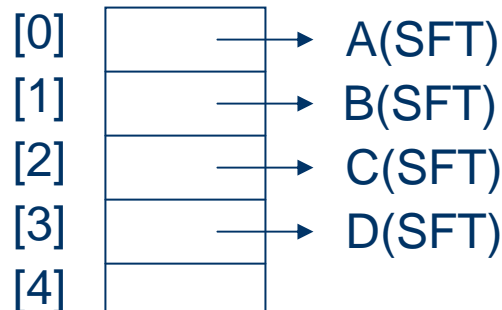
Inheritance of File Descriptors

- “fork” creates a child and the child inherits a copy of most of the parent’s environment and context
 - Signal state
 - Scheduling parameters
 - File descriptor table

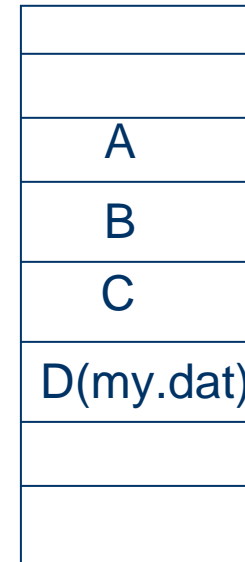
Parent
File
Descriptor
Table



Child’s
File descriptor
Table



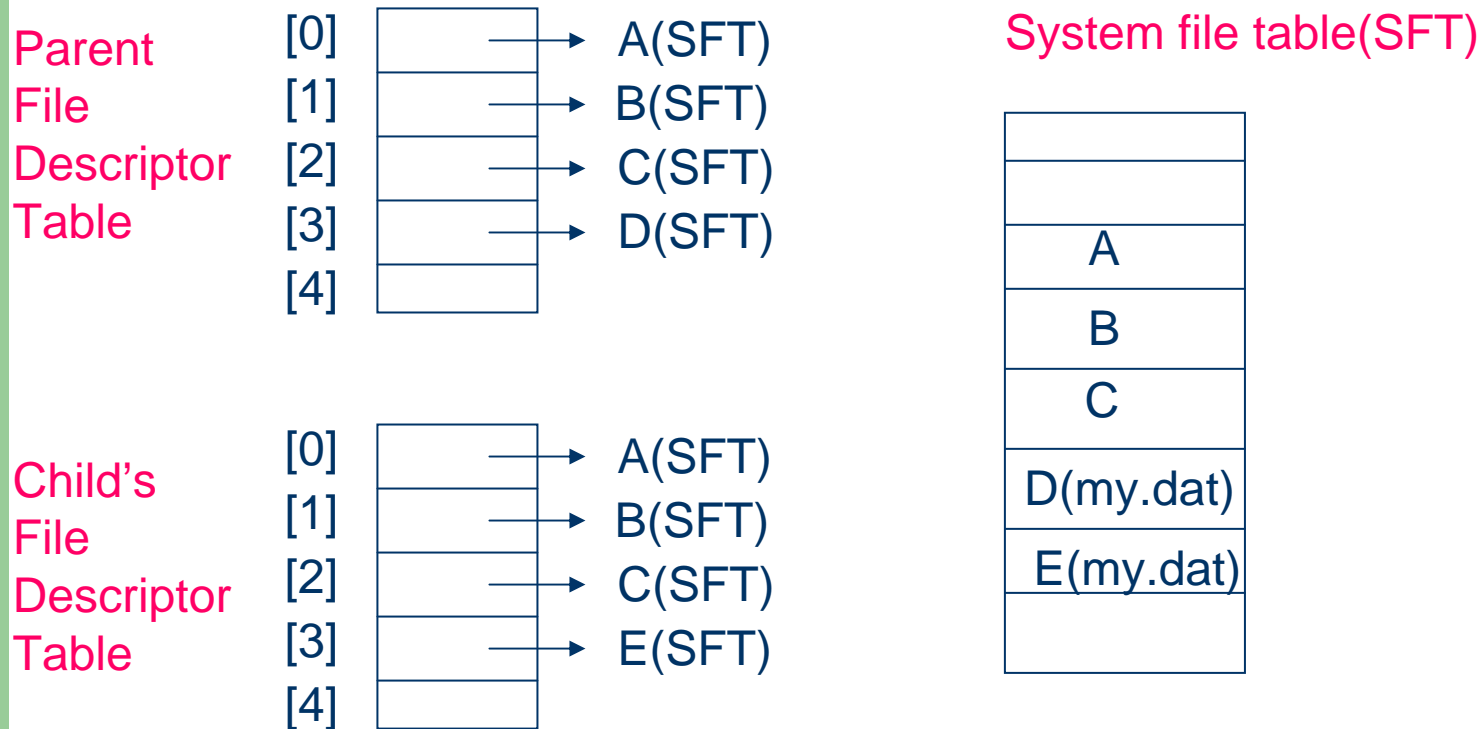
System file table(SFT)



If parent opens my.dat before forking
Both parent and child share SFT entry

CS 241 - System Programming,
Klara Nahrstedt

Inheritance of File Descriptors



If parent and child open my.dat after the fork Call, their file descriptor table entries point to Different SFT entries

Filters and Redirection

- **Filters** – read standard input, perform transformation and output results to standard output
- Examples of useful UNIX filters
 - head, tail, more, sort, grep, awk, cat
- **Redirection** – action to redirect output
 - Program can modify the file descriptor table entry so that it points to a different entry in the SFT
- Redirection uses
 - < redirection of standard input
 - > redirection of standard output

Redirection

```
#include <unistd.h>
```

```
int dup2(int fildes, int fildes2);
```

Copies file descriptor entry fildes to entry fildes2

After open

[0]	Standard input
[1]	Standard output
[2]	Standard error
[3]	Write to my.file

After dup2

[0]	Standard input
[1]	Write to my.file
[2]	Standard error
[3]	Write to my.file

After close

[0]	Standard input
[1]	Write to my.file
[2]	Standard error

File Control

- **fcntl** – general purpose function for retrieving and modifying flags associated with an open file descriptor

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

```
int fcntl(int fildes, int cmd, /*arg */. . . );
```

Fildes – file descriptor, cmd – argument to specify operation

- **F_DUPFD** – duplicate file descriptor
- **F_SETFD** – set file descriptor flags
- **F_GETFL** – get file status flags and access modes

Directories

R&R: Chapter 5



UNIX File System Navigation

- **Directory** –
 - file containing directory entries
 - Associates a file name with physical location of a file on a disk
- **Tree-structured organization of a typical file system**
 - / - root directory
 - Absolute – fully qualified pathnames
 - /dirA/dirB/my1.dat
 - Current working directory
 - ../my2.dat (/dirA/my2.dat)
 - ../dirB/my1.dat (/dirA/dirB/my1.dat)

Path

- **PWD** environment variable – specifies current working directory
- **Chdir** function – causes directory specified by path to become current working directory

```
#include <unistd.h>  
int chdir(const char *path);
```

- **Getcwd** function – returns pathname of the current directory

```
char * getcwd(char *buf, size_t size);
```

- Other path functions
 - **fpathconf /pathconf** – report limits associated with a particular file or directory

Directory Access

- Directories should not be accessed with the ordinary open, close and read functions.
- We need specialized functions
 - `Opendir`
 - `Closedir`
 - `Readdir`

```
#include <dirent.h>
DIR *opendir(const char *dirname);
struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);
void rewinddir(DIR *dirp);
```

List files in a directory

```
#include <dirent.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
    struct dirent *direntp;
    DIR *dirp;
    if (argc != 2) {
        fprintf(stderr, "Usage: %s directory_name\n", argv[0]);
        return 1;
    }
    if ((dirp = opendir(argv[1])) == NULL) {
        perror("Failed to open directory");
        return 1;
    }
    while ((direntp = readdir(dirp)) != NULL)
        printf("%s\n", direntp->d_name);
    while ((closedir(dirp) == -1) && (errno == EINTR)) ;
    return 0;
}
```

File Status

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

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

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

- Lstat returns status info and info about symbolic links
- Stat returns status info of files referred to by any links

Struct Stat

```
dev_t      st_dev;      /* device ID of device
                        containing file */
ino_t      st_ino;      /* file inode number */
mode_t     st_mode;     /* file mode */
nlink_t    st_nlink;    /* number of hard links */
uid_t      st_uid;     /* user id of file */
gid_t      st_gid;     /* group id of file */
off_t      st_size;     /* file size in bytes */
time_t     st_atime;    /* time of last access */
time_t     st_mtime;    /* time of last data modification */
time_t     st_ctime;    /* time of last file status
                        change */
```

Determining the type of file

- `st_mode` – file mode member
 - Specifies the access permissions of the file and the type of file
- **Types of file**
 - **Regular file** – randomly accessible sequence of bytes with no further structure imposed by the system
 - **Special files** – specify devices
 - **Character special files** – represent devices such as terminals
 - **Block special files** – represent disk devices

POSIX Macros for testing `st_mode` member for type of file

macro	tests for
<code>S_ISBLK(m)</code>	block special file
<code>S_ISCHR(m)</code>	character special file
<code>S_ISDIR(m)</code>	directory
<code>S_ISFIFO(m)</code>	pipe or FIFO special file
<code>S_ISLNK(m)</code>	symbolic link
<code>S_ISREG(m)</code>	regular file
<code>S_ISSOCK(m)</code>	socket

Summary

- UNIX Indirection, Filters and Redirection
- Access Directories