File Manipulations

- Create files
- Open files
- Transfer data to and from files
- Close files
- Remove files
- Query file attributes
- Truncate files

open (1)

- #include <sys/types.h>

  #include <sys/stat.h>

  #include <fcntl.h>

  int open(const char **pathname*, int *flags*, [mode_t *mode]*);
  - Attempts to open a file and return a file descriptor.
  - *mode* specifies the permission only when a new file is created.

open (2)

- *flags*
  - O_RDONLY, O_WRONLY, or O_RDWR
  - O_CREAT
    - If the file does not exist it will be created.
  - O_EXCL
    - When used with O_CREAT, if the file already exists it is an error and the open will fail.
  - O_TRUNC
    - If the file already exists it will be truncated.
  - O_APPEND
    - Initially, and before each write, the file pointer is positioned at the end of the file.

open (3)

- *mode*
  - Specifies the permissions to use if a new file is created.
  - Should always be specified when O_CREAT is in the flags, and is ignored otherwise.
- Return value
  - Return the new file descriptor, or -1 if an error occurred.
- Examples
  - Refer to the text book (2.1.3)

open (4)

```
fd = open("startup", O_RDONLY);
if(fd == -1)
       errExit("open");

fd = open("myfile", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
if(fd == -1)
       errExit("open");

fd = open ("w.log", O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, S_IRUS
R | S_IWUSR);
if(fd == -1)
       errExit("open");
```

creat

- #include <sys/types.h>
  #include <sys/stat.h>
  #include <fcntl.h>
  int creat(const char **pathname*, mode_t *mode*);
  - Create a new file.
  - Equivalent to open with flags equal to
    - O_CREAT|O_WRONLY|O_TRUNC
  - Example
    - Refer to the text book (2.1.4)

close

- #include <unistd.h>

  int close(int *fd*);
  - Closes a file descriptor.
  - When a process terminates, all open files are automatically closed by the kernel.
  - Return value
    - Zero on success, or -1 if an error occurred.
  - Example
    - Refer to the text book (2.1.5)

read

- #include <unistd.h>
  ssize_t read(int *fd*, void **buf*, size_t *count*);
  - Attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf.*
  - If count is zero, read( ) returns zero and has no other results.
  - Return value
    - On success, the number of bytes read.
    - Zero indicates end of file.
    - On error, -1 is returned.

write

- #include <unistd.h>

  ssize_t write(int *fd*, const void **buf*, size_t *count*);

  – Writes up to *count* bytes to the file referenced by the file descriptor *fd* from the buffer starting at *buf.*

  – Return value

    - The number of bytes written.
    - Zero indicates nothing was written.
    - On error, -1 is returned.

Example #1: Simple File I/O (1)

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

#define BSIZE 1024
#define FPERM 0644

int main(int argc, char *argv[])
{
     int fd1, fd2, n;
     char buf[BSIZE];

     if (argc < 3) {
          fprintf(stderr, "Usage; %s src dest\n", argv[0]);
          exit(1);
     }
```

Example #1: Simple File I/O (2)

```
    if ((fd1 = open(argv[1], O_RDONLY)) < 0) {
         perror("file open error");
         exit(1);
    }
    if ((fd2 = creat(argv[2], FPERM)) < 0) {
         perror("file creation error");
         exit(1);
    }

    while ((n = read(fd1, buf, BSIZE)) > 0)
         /* assume no read/write error */
         write(fd2, buf, n);

    close(fd1);
    close(fd2);
}
```

lseek (1)

- #include <sys/types.h>
  #include <unistd.h>
  off_t lseek(int *fd*, off_t *offset*, int *whence*);
  - Repositions the offset of the file descriptor *fd* to the argument offset.
  - *whence*
    - SEEK_SET
      - The offset is measured from the beginning of the file.
    - SEEK_CUR
      - The offset is measured from the current position of the file.
    - SEEK_END
      - The offset is measured from the end of the file.

lseek (2)

- Hole
  - Allows the file offset to be set beyond the end of the existing end-of-file of the file.
  - If data is later written at this point, subsequent reads of the data in the gap return bytes of zeros.
- Return value
  - Success: the resulting offset location as measured in bytes from the beginning of the file.
  - Error: -1

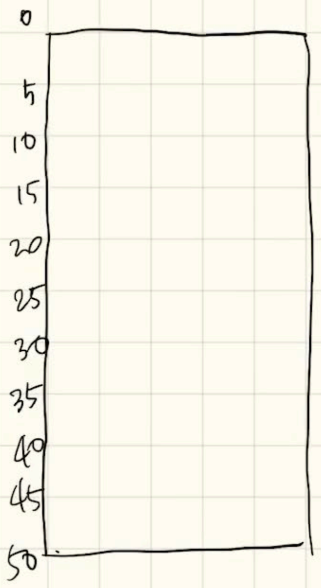
Example #2: lseek (1)

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

char buf1[] = "abcdefghij";
char buf2[] = "ABCDEFGHIJ";

int main(void)
{
    int fd;

    if ((fd = creat("file.hole", 0640)) < 0) {
        perror("creat error");
        exit(1);
    }
```

Example #2: lseek (2)

```
    if (write(fd, buf1, 10) != 10) {
        perror("buf1 write error");
        exit(1);
    }
    /* offset now = 10 */

    if (lseek(fd, 40, SEEK_SET) == -1) {
        perror("lseek error");
        exit(1);
    }
    /* offset now = 40 */

    if (write(fd, buf2, 10) != 10) {
        perror("buf2 write error");
        exit(1);
    }
    /* offset now = 50 */

    exit(0);
}
```

Example #2: lseek (3)



Remove (unlink, rmdir)

- #include <stdio.h>

  int remove(const char **pathname*);
  - C Library function (*not* a system call)
  - Delete a name and possibly the file it refers to.
    - It calls unlink( ) for files, and rmdir( ) for directories.
  - Return value
    - On success, zero is returned.
    - On error, -1 is returned.

fcntl (1)

- #include <unistd.h>

  #include <fcntl.h>

  int fcntl(int *fd*, int *cmd*);

  int fcntl(int *fd*, int *cmd*, long *arg*);

  int fcntl(int *fd*, int *cmd*, struct lock **ldata*);

  - Manipulate file descriptor.
  - Performs one of various miscellaneous operations on *fd*.
  - The operation in question is determined by *cmd*:

fcntl (2)

- F_GETFL
  - Read the descriptor’s flags.
  - All flags (as set by `open( )`) are returned.
- F_SETFL
  - Set the descriptor’s flags to the value specified by *arg.*
  - The other flags are unaffected.
  - On success returns 0, otherwise returns -1.

Example #3: fcntl (1)

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

int main(int argc, char *argv[])
{
    int accmode, val;

    if (argc != 2) {
        fprintf(stderr, "usage: a.out <descriptor#>");
        exit(1);
    }

    if ((val = fcntl(atoi(argv[1]), F_GETFL, 0)) < 0) {
        perror("fcntl error for fd");
        exit(1);
    }

    accmode = val & O_ACCMODE;
```

Example #3: fcntl (2)

```
    if (accmode == O_RDONLY)
        printf("read only");
    else if (accmode == O_WRONLY)
        printf("write only");
    else if (accmode == O_RDWR)
        printf("read write");
    else {
        fprintf(stderr, "unkown access mode");
        exit(1);
    }

    if (val & O_APPEND)
        printf(", append");
    if (val & O_NONBLOCK)
        printf(", nonblocking");
    if (val & O_SYNC)
        printf(", syschronous writes");
    putchar('\n');
    exit(0);
}
```

Example #4: fcntl

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

/* flags are file status flags to turn on */
void set_fl(int fd, int flags)
{
    int val;

    if ((val = fcntl(fd, F_GETFL, 0)) < 0) {
        perror("fcntl F_GETFL error");
        exit(1);
    }

    val |= flags;     /* turn on flags */
    if (fcntl(fd, F_SETFL, val) < 0) {
        perror("fcntl F_SETFL error");
        exit(1);
    }
}
```

dup and dup2 (1)

- #include <unistd.h>

  int dup(int *oldfd*);

  int dup2(int *oldfd*, int *newfd*);
  - Create a copy of the file descriptor *oldfd.*
  - The old and new descriptors may be used interchangeably.
    - If the file position is modified by using lseek( ) on one of the descriptors the position is also changed for the other.
  - dup( ) uses the <u>*lowest-numbered unused* descriptor for the new descriptor</u>. dup2( ) makes *newfd* be the copy of *oldfd*, closing *newfd* first if necessary. (dup2 is now obsolete.)
  - Return value: the new descriptor, or -1 if an error occurred.

Example #5: dup (1)

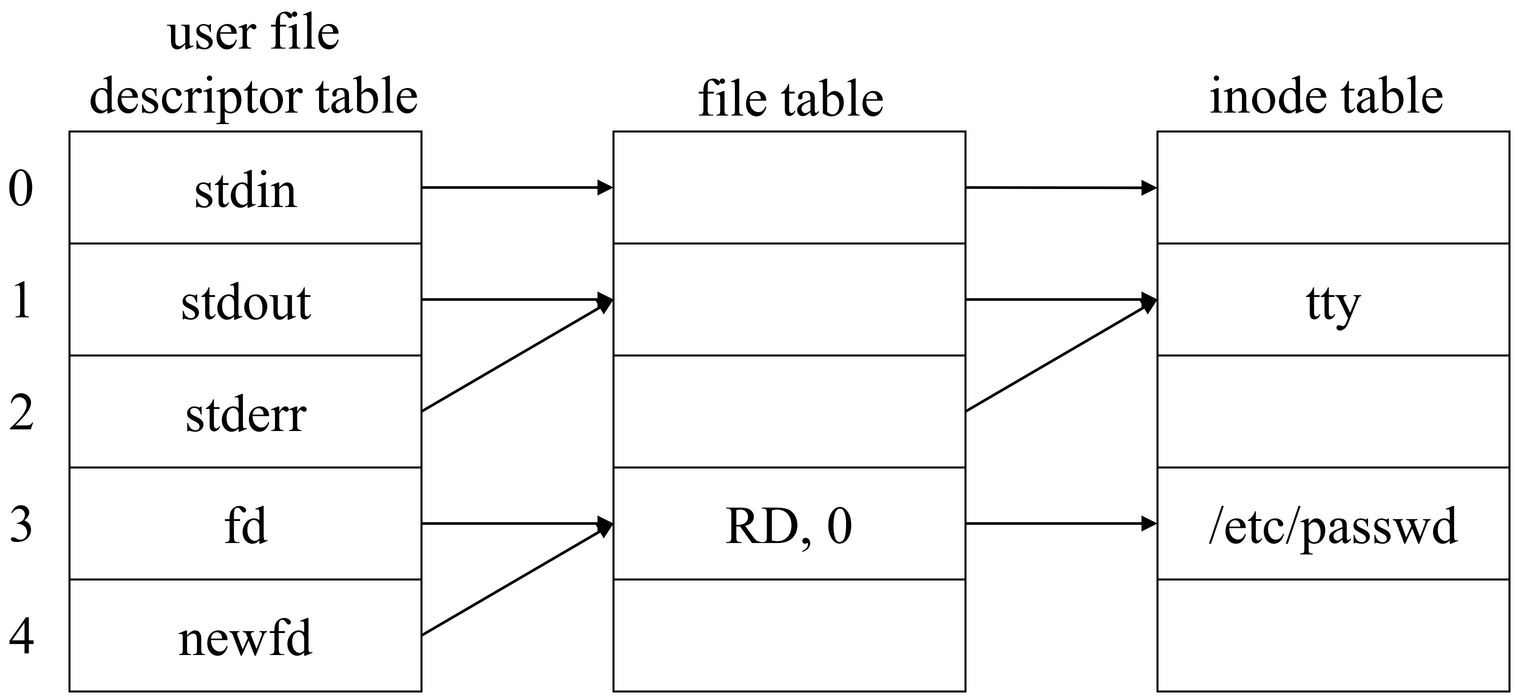
```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

#define BSIZE 80

int main(void)
{
    int fd, newfd, n;
    char buf1[BSIZE], buf2[BSIZE];

    fd = open("/etc/passwd", O_RDONLY);
    newfd = dup(fd);

    n = read(fd, buf1, BSIZE);
    printf("Read from fd:\n\n");
    write(STDOUT_FILENO, buf1, n);
```

Example #5: dup (2)

```
    n = read(newfd, buf2, BSIZE);
    printf(“\n\nRead from newfd:\n\n”);
    write(STDOUT_FILENO, buf2, n);

    close(fd);

    n = read(newfd, buf1, BSIZE);
    printf("\n\nRead from newfd after close(fd):\n\n");
    write(STDOUT_FILENO, buf1, n);
    printf("\n");

    close(newfd);
    exit(0);
}
```

Example #3: dup (3)

```
fd = open("/etc/passwd", O_RDONLY);
newfd = dup(fd);
```



Thank you