Module 4

Files

Overview

- Objectives
- Relevance

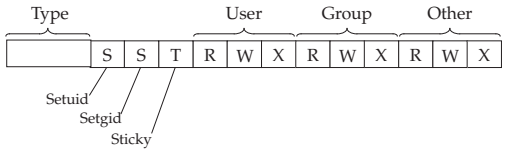
Files Overview

- Structure of regular files
- Characteristics of regular files
- Symbolic links
- File information
- Macros
- Permissions

Macros

```
1   #include <sys/stat.h>
2
3   main() {
4     struct stat buf;
5
6     if (stat("file", &buf)) {
7       perror ("Couldn't stat file");
8     } else {
9
10      /* S_ISDIR defined in <sys/stat.h> */
11      if (S_ISDIR(buf.st_mode)) {
12        printf("It is a directory!\n");
13      }
14    }
15  }
```

Permissions



Permissions

```
1   #include <sys/types.h>
2   #include <sys/stat.h>
3   #include <stdio.h>
4
5   main() {
6     struct stat buf;
7
8     if (stat("filename",&buf)) {
9       perror ("Couldn't stat file");
10    } else {
11
12      /* chmod u+s,u+x,g+x */
13      buf.st_mode |= S_ISUID | S_IXUSR |
S_IXGRP
;
14      /* chmod g-w,o-w */
15      buf.st_mode &= ~(S_IWGRP | S_IWOTH);
16
17      if (chmod("filename",buf.st_mode) ==
-1)
18        {
19          perror("chmod : filename");
20        }
21    }
```

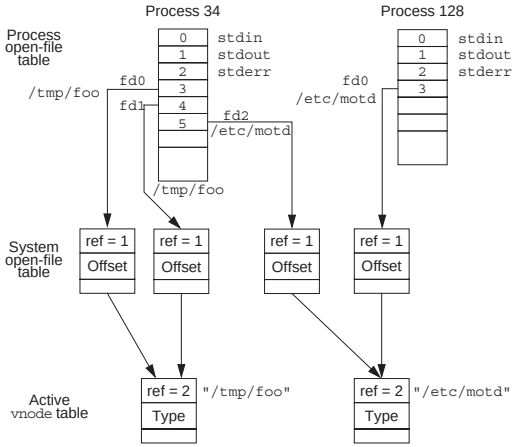

Opening Files

```
1   #include <sys/types.h>
2   #include <sys/stat.h>
3   #include <fcntl.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <unistd.h>
7   main() {
8
9     int  fd, outfd;
10
11    fd = open("/etc/motd", O_RDONLY);
12
13    if (fd == -1) {
14      perror("open");
15      exit(1);
16    }
17
18    outfd = open("output",
19      O_WRONLY | O_CREAT | O_TRUNC, 0777);
```



```
20
21    if (outfd == -1) {
22      perror("open");
23      exit(1);
24    }
25
26    /* Use the file descriptors here */
27    close(fd);
28    close(outfd);
29  }
```

Open File Table



User Mask

```
1   #include <sys/types.h>
2   #include <sys/stat.h>
3   #include <fcntl.h>
4
5   main()   {
6
7      mode_t  oldmask;
8      int     fd;
9
10     /* allow group write permission temporarily */
11     oldmask = umask(002);
12     fd = open("testfile1", O_WRONLY | O_CREAT, 0666);
13     umask(oldmask);
14     fd = open("testfile2", O_WRONLY | O_CREAT, 0666);
15  }
```

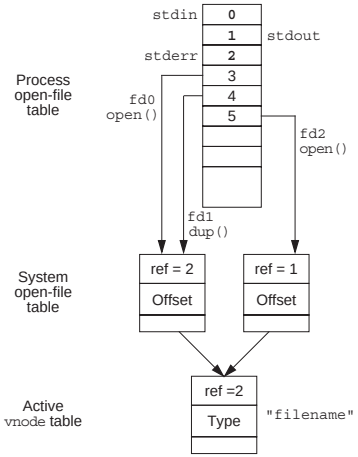

Removing Files

- `unlink()`— Deletes a file or directory; no check for empty directory
- `remove()`— ANSI standard, deletes a file or directory; checks for empty directory

Duplicating Descriptors

- `dup()`— Finds lowest available descriptor and copies open file table pointer
- `dup2()`— Copies open file system pointer into specified descriptor

Duplicating Descriptors



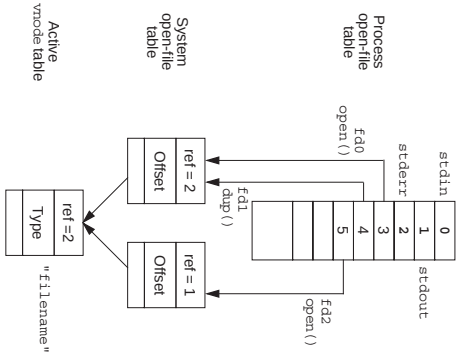
mydup.c

```
1   #include <sys/types.h>
2   #include <sys/stat.h>
3   #include <fcntl.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <unistd.h>
7
8   main() {
9
10    int fd;
11
12    fd = open("/tmp/foo", O_CREAT|O_WRONLY|O_TRUNC, 0660);
13
14    if( fd == -1 ) {
15      perror("open");
16    exit(1);
17    }
18    close(stdout);
19
20    /* If succeeds file desc 1 is "/tmp/foo" */
```



```
21    if( dup(fd) != stdout ) {
22      fprintf(stderr,"dup failed to return 1!\n");
23      exit(1);
24    }
25    /* Don't need fd any more */
26    close(fd);
27
28    /* stdout output goes to "/tmp/foo" */
29    printf("Hi Folks!\n");
30  }
```


Duplicating Descriptors



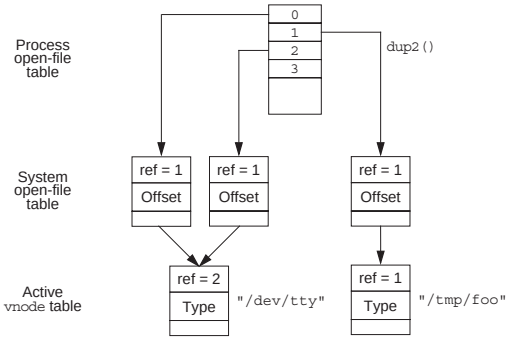
mydup2.c

```
1   #include <sys/types.h>
2   #include <sys/stat.h>
3   #include <fcntl.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <unistd.h>
7
8   main() {
9
10    int fd;
11
12    fd = open("/tmp/foo", O_CREAT|O_WRONLY|O_TRUNC, 0660);
13
14    if( fd == -1 ) {
15      perror("open");
16      exit(1);
17    }
18
19    /* Make file desc 1 refer to "/tmp/foo" */
20    if( dup2(fd, STDOUT_FILENO) == -1 ) {
```



```
21      perror("dup2");
22      exit(1);
23    }
24
25    /* Don't need fd any more */
26    close(fd);
27
28    /* stdout output goes to "/tmp/foo" */
29    printf("Hi Folks!\n");
30  }
```


Duplicating Descriptors



Reading and Writing Files

```
1   #include <sys/types.h>
2   #include <sys/stat.h>
3   #include <fcntl.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <unistd.h>
7
8   #define  MAXSIZE   256
9
10  main() {
11
12    int   fd, n;
13    char  array[MAXSIZE];
14
15    fd = open("/etc/motd", O_RDONLY);
16
17    if( fd == -1 ) {
18      perror("open");
19      exit(1);
```

```
20    }
21
22    while ((n = read(fd, array, MAXSIZE)) > 0) {
23      if (write(STDOUT_FILENO, array, n) != n) {
24        perror("write");
25      }
26    }
27
28    if (n == -1) {
29      perror("read");
30    }
31    close(fd);
32  }
```


Changing the Offset Location

```
1   off_t start;
2   struct record rec;
3
4   /* record current location in "start" */
5   start = lseek(fd, (off_t)0, SEEK_CUR);
6   /* do other stuff in the file */
7     .
8     .
9   /* go back to "start" */
10  lseek(fd, start, SEEK_SET);
11  read(fd, (void *)&rec, sizeof(rec));
12    .
13    .
14  /* rewrite last record */
15  lseek(fd, -sizeof(rec), SEEK_CUR);
16  write(fd, (void *)&rec, sizeof(rec));
```


Closing Files

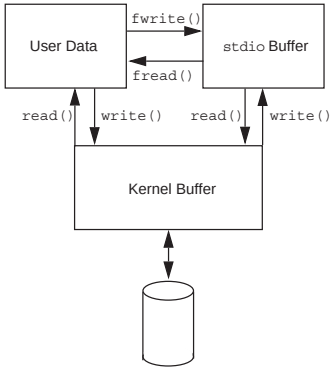
```
1   #include <sys/types.h>
2   #include <fcntl.h>
3   #include <unistd.h>
4
5   main() {
6
7     int fd;
8     char str[]="Howdy, Folks\n";
9
10    fd = open("testfile1", O_WRONLY | O_CREAT, 0777);
11    write(fd, str, strlen(str) );
12
13    close(fd);
14    return(0);
15  }
```


Standard I/O Routines

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   main() {
4
5     FILE *fp;
6     char buf[512];
7     char *filename = "foo";
8     int num;
9
10    /* opens "filename" for read & write */
11    fp = fopen(filename, "r+");
12
13    if( fp == NULL ) {
14      fprintf(stderr, "Error: fopen failed on %s\n",
15              filename);
16      exit(1);
17    }
18
19    /* reads  200 bytes into buf */
```

```
20    num = fread(buf, 1, 200, fp);
21    fflush( fp );
22
23    /* writes 200 bytes from buf */
24    num = fwrite(buf, 1, num, fp);
25    fclose(fp);
26  }
```


Cautions for Using Standard I/O



Related Standard I/O Calls

- `fdopen()` – Obtains file pointer
- `fileno()` – Obtains file descriptor
- `freopen()` – Redirects standard I/O operation

Related Standard I/O Calls

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   main() {
5
6     if( freopen("/tmp/foo3", "w", stdout) != stdout ) {
7       fprintf(stderr,"ERROR: reopen\n");
8       exit(1);
9     }
10
11    /* stdout is now "/tmp/foo3" */
12    printf("Hi Folks!\n");
13  }
```

Temporary Files

- `tmpnam()` – Generates temporary name
- `tempnam()` – Specifies directory and prefix
- `tmpfile()` – Generates temporary file

Anonymous Pipes

Anonymous Pipes

```
1   #include <stdio.h>
2
3   main() {
4
5     FILE *fp;
6     int m;
7
8     /* Write your screen output through "more" */
9     fp = popen("more", "w");
10
11    if (fp == NULL) {
12      fprintf(stderr, "popen failed.\n");
13      return(1);
14    }
15
16    for( m = 1; m <= 100; m++ ) {
17      fprintf(fp, "Lots and lots of stuff.\n");
18    }
19    pclose(fp);
```



```
20    return(0);
21
22    /* Note: Could have used:
          fwrite(buf, size, n, fp) */
23  }
```

Anonymous Pipes for Related Processes

mypipe.c

```
1   #include <sys/types.h>
2   #include <sys/wait.h>
3   #include <stdio.h>
4   #include <unistd.h>
5
6   #define BSIZE 1024
7
8   main() {
9
10    int pd[2];
11    char buf[BSIZE];
12
13    if (pipe(pd) == -1) {
14      perror("pipe");
15      exit(1);
16    }
17
```



```
18    switch(fork()) {
19
20    case -1:
21      perror("fork");
22      exit(1);
23      break;
24
25    /* child */
26    case 0:
27      /* Close read side, won't use it */
28      close(pd[0]);
29      write( pd[1], "\nHi, Mom, I'm your kid!", 24);
30      close(pd[1]);
31      exit(0);
32      break;
33
34    /* parent */
35    default:
36      break;
37    }
38
39    /* Close write side, won't use it */
```



```
40    close(pd[1]);
41
42    /* Assumes a single string of */
43    /* size less than BSIZE was written.  */
44    read( pd[0], buf, BSIZE);
45    puts(buf);
46    close(pd[0]);
47
48    if (waitpid(-1, (int *)NULL, 0) == -1) {
49      perror("waitpid");
50      exit(1);
51    }
52    return 0;
53  }
```


Rules for Reading and Writing to Pipes

I/O attempt	Conditions	Result
read	Empty pipe, writer attached	read blocks
read	Empty pipe, no writer attached	read returns 0 (EOF)
write	Full pipe, reader attached	write blocks
write	No reader attached	SIGPIPE sent

mypipe2.c

```
1   #include <stdio.h>
2   #include <unistd.h>
3
4   main() {
5
6     int p[2];
7     pid_t pid;
8
9     if ( pipe(p) == -1 ) {
10      perror("pipe");
11      exit(1);
12    }
13
14    switch( pid = fork() ) {
15
16    case -1:
17      perror("fork");
18      exit(1);
19      break;
20
```



```
21    /* child */
22    case 0:
23      close(p[1]);
24
25      if (p[0] != 0) {
26        dup2(p[0], 0);
27        close(p[0]);
28      }
29      execlp("mailx","mailx","-s","Error","deac",
30          char *)NULL);
31      perror("execlp");
32      exit(1);
33      break;
34
35    /* parent */
36    default:
37      break;
38    }
39    close(p[0]);
40
41    /* if an error occurs in the program, then do: */
42    write(p[1], "An error occurred.\n", 18);
```



```
43    close(p[1]);
44
45    if (waitpid(pid, (int *)NULL, 0) == -1) {
46      perror("waitpid");
47      exit(1);
48    }
49  }
```


Simple File Control

```
1     int fd, flags, origflags;
2
3     /* Read flag  on fd & change to non-blocking IO,
4      O_NONBLOCK */
5     origflags = fcntl(fd, F_GETFL, 0);
6     flags = origflags | O_NONBLOCK;
7     fcntl(fd, F_SETFL, flags);
8
9     /* try to read some data */
10    while (read(fd, buf, HUGE_VAL) == -1 &&
11          errno == EAGAIN) {
12      /* do something else, check later */
13      process(buf);
14    }
15
16    /* Change flag on fd back to original */
17    fcntl(fd, F_SETFL, origflags);
```

Exercise: Files

- Objectives
- Tasks
- Discussion
- Solutions