

UNIX Systems Programming

(lectures programs)

UNIX File System

➤ Low-level I/O: *open, read, write, lseek, close, unlink*

→ Example FILE1: [append.c](#)

```
main(int argc, char **argv)
{
    int n, in, out;
    char buf[1024];

    /* Open the first file for reading */
    in = open (argv[1], O_RDONLY);

    /* Open the second file for writing */
    out = open (argv[2], O_WRONLY | O_APPEND);

    /* Copy data from the first file to the second */
    while ((n = read (in, buf, sizeof(buf))) > 0)
        write (out, buf, n);
}
```

Examples of usage:

```
% touch t           // creat empty file t
% append append.c t // copy append.c to t
% append append.c t // add a 2nd copy of append.c to t
% append t t        // endlessly copies t to t !
```

→ Example FILE2: [fileseek.c](#)

```
main(...)
{
    int fd, length;
    off_t FileLength, position;
```

```

char buf[1024];
    printf("Usage: fileseek <file>
<seek_position> <#bytes>\n");
    fd = open(argv[1], O_RDONLY);
    position = atoi(argv[2]);
    length = atoi(argv[3]);
    FileLength = lseek(fd, 0, SEEK_END);
    if ((position+length) >= FileLength){
        printf("exceeds file limit\n");
        exit(0);
    }
    lseek(fd, position, SEEK_SET);
    read(fd, buf, length);
    write(1, buf, length);
}

```

Examples of usage:

```

% ls -lt fileseek.c
-rwxr-xr-x  1 cs476  cs476    782 Sep 26  2006
fileseek.c

```

```

% fileseek fileseek.c 700 100
File Length 782
exceeds file limit t

```

```

% fileseek fileseek.c 0 100
File Length 782
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib

```

➤ **Standard I/O:** fopen, fread, fwrite, fseek, fclose

→ **Example FILE3:** [ffileseek.c](#)

```

main(..)
{
    FILE *fp;
    int length;
    off_t FileLength, position;

```

```

char buf[1024];
printf("Usage: fileseek <file>
<seek_position> <#bytes>\n");
fp = fopen(argv[1], "r");
position = atoi(argv[2]);
length = atoi(argv[3]);
fseek(fp, 0, SEEK_END);
FileLength = ftell(fp);
if ((position+length) >= FileLength){
    printf("exceeds file limit\n");
    exit(0);
}
fseek(fp, position, SEEK_SET);
fread (buf, sizeof(char), length, fp);
fwrite(buf, sizeof(char), length, stdout);
}

```

Examples of usage:

```
% ffileseek fileseek.c 10 200 // display 200 bytes starting at position 10
```

NOTE: you may use: `fdopen(fd)` & `fileno(fp)`
to **convert** between file pointer (`fp`) and file descriptor (`fd`).

Example:

```

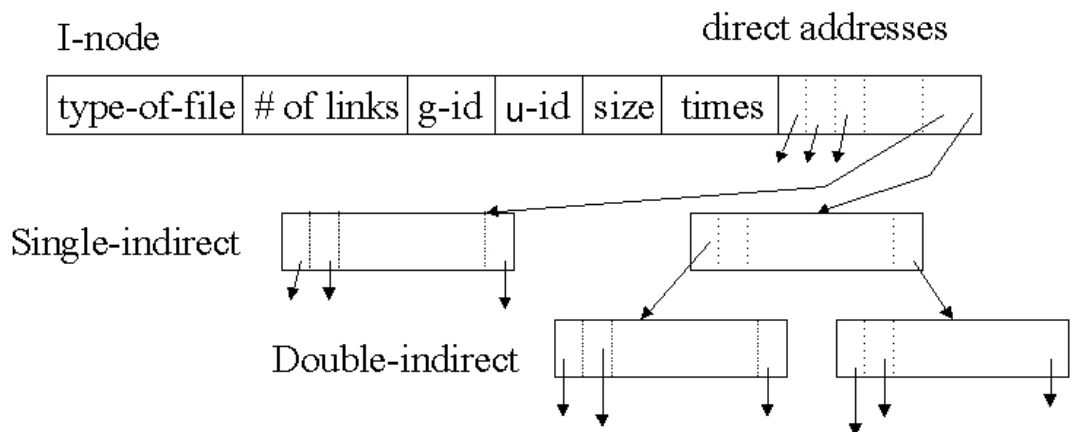
int fd;
FILE *fp;

fd = open ("filename", O_TRUNC | O_CREAT |
O_WRONLY, 0777 );

fp = fdopen(fd, "w+");
fd = fileno(fp);

```

➤ **UNIX Directory Structure:** (See next Figure)



NOTE: The filename is **not stored** in the I-node.
 It is available in the parent directory.
 Every file has a record in the parent directory in the form:

FileName		Inode#
----------	--	--------

Signals

➤ Basic Signal Handling:

✓ **kill** (*pid*, *sig*)

Send process *pid* signal *sig*
 (To list all available signals type: `% kill -l`)

✓ **signal** (*sig*, *handler*)

When signal *sig* occurs, call function *handler*

✓ `pause()`

To wait for the arrival of a signal

→ Example SIG1: `signal1.c`

```
main ..
{
    signal(SIGUSR1, handler);
    signal(SIGUSR2, handler);

    for (;;)
        pause();
}

handler(int sig)
{
    /* Print received signal */
    psignal(sig, "Received signal");
}
```

Examples of usage:

```
% kill -l
HUP INT QUIT ILL TRAP ABRT EMT FPE KILL BUS SEGV SYS
PIPE ALRM TERM USR1 USR2 CLD PWR WINCH URG POLL
STOP TSTP CONT TTIN TTOU VTALRM PROF XCPU XFSZ
WAITING LWP FREEZE THAW CANCEL LOST RTMIN RTMIN+1
RTMIN+2 RTMIN+3 RTMAX-3 RTMAX-2 RTMAX-1 RTMAX
```

```
% kill -USR1 12345 //use ps to find pid of signal1 e.g., 1234
Recived signal: User Signal 1
```

```
% kill -17 12345 //signal #17 is USR2
Recived signal: User Signal 2
```

```
% kill -KILL 12345 //signal #9 is KILL
```

Killed

➤ Using Signals for Timeouts:

alarm (*T*)
sends the **ALRM** signal to the current process after *T* seconds.

alarm (0)
cancels the alarm signal.

→ Example SIG2: [timeout1.c](#)

```
int flag = 0;

main ..

{
    int timeout;
    signal(SIGALRM, handler);
    timeout = alarm(atoi(argv[1]));
    printf("Enter a string (within %d seconds):",
        timeout);
    fgets(buf, sizeof(buf), stdin);
    alarm(0);
    if (!flag)
        printf ("Typed: %s\n", buf);
}

handler(int sig)

{
    printf("Timeout alarm\n");
    flag = 1;
}
```

Example of usage:

```
% timeout1 3
Enter a string (within 3 seconds):
```

Processes

➤ Basic Concepts:

Process identifiers:

- ✓ `getpid()`: to get the process id.
- ✓ `getppid()`: to get the parent process id.

Create a new process:

- ✓ `fork()`: to create a new child process.

Terminate:

- ✓ `exit()`: to terminate the current process.

Zombie Process:

If a process dies and its parent did not call

- ✓ `wait()`

the process is called a *zombie* process.

Orphans Processes:

If a parent process dies,
it's children becomes *orphans* and
are inherited by the system's *init* process (pid *#1*).

→ Example PROC1: [fork1.c](#)

```
main ()
{
    pid_t pid;
    pid = fork();
    if (pid == 0) {
        printf("I am child\n");
        printf("my pid is: %d\n", getpid());
        printf("my parent pid is: %d\n",
getppid())
    }
    else {
        usleep(1); /*so child goes first*/
    }
}
```

```

        printf("I am a parent\n");
        printf("my child pid is: %d\n", pid);
        printf("my pid is: %d\n", getpid());
    }
    /* This code executes in both processes */
    printf("exiting ... %d\n", getpid());
    exit(0);
}

```

Example of usage:

```
% fork1
```

→ Example PROC2: [killparent.c](#)

*This example shows that when the parent is dead,
It's children are inherited by the **init** process (pid # 1).*

```

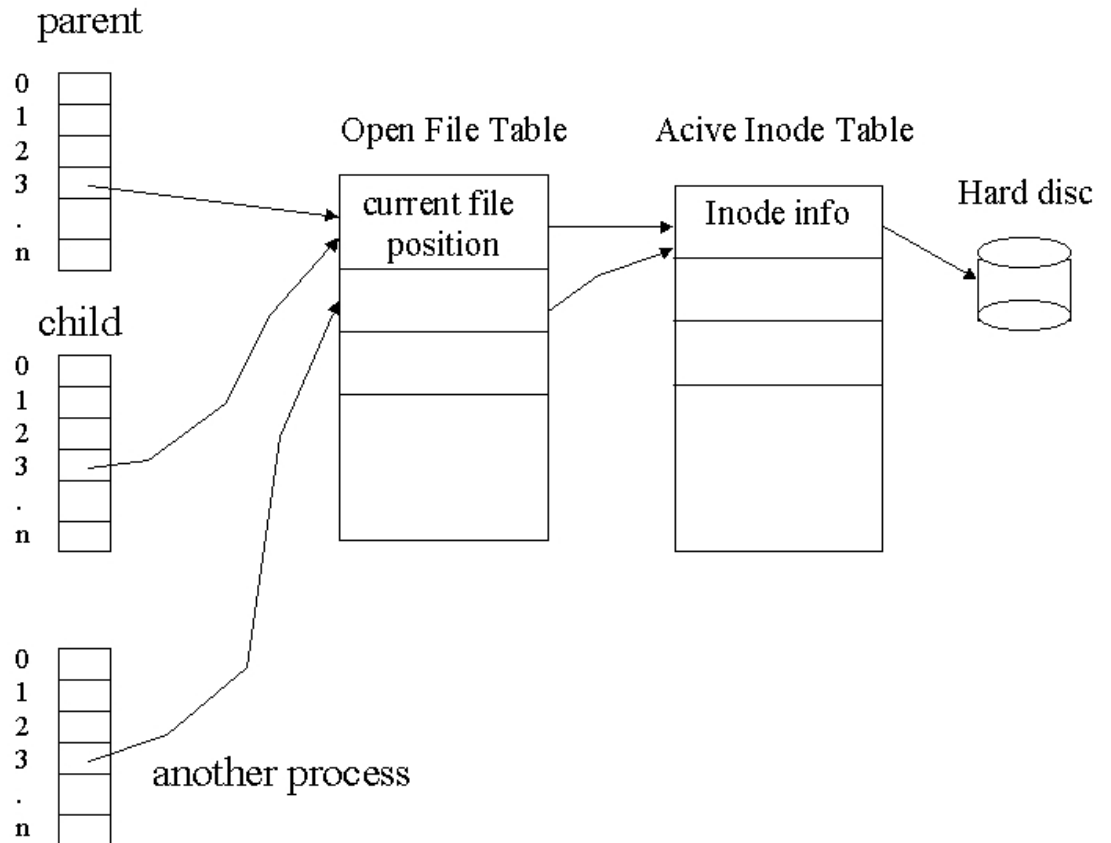
main ...
{
    pid_t pid;
    pid = fork();
    if (pid == 0) {
        printf("my real parent id is: %d\n",
getppid());
        kill ( getppid(), 9);
        usleep(1);
        printf("my foster parent id is:
%d\n",getppid());
    }
    pause();
}

```

Example of usage:

```
% killparent
```

➤ File Sharing between processes: [\(see next Figure\)](#)



→ **Example PROC3:** [fileshare.c](#)

This example shows that both parent and child share the same file pointer.

```
main ()
{
    int in;
    pid_t pid;
    char buf[1024];
    system ("echo abc123 > junk");
    in = open("junk", O_RDONLY);
    pid = fork();
    if (pid == 0) {
        write(1,"Child: ",7);
        read(in, buf, 3);
        write(1, buf, 3);
    }
    else {
        write(1,"Parent: ",8);
        read(in, buf, 3);
    }
}
```

```

        write(1, buf, 3);
    }
    write(1, " end\n", 5);
}

```

Example of usage:

```
% fileshare
```

➤ **EXEC:** many variations, including:

- ✓ `exec1` (path, arg1, arg2, ..., NULL)
- ✓ `execv` (path, *argv[])

→ **Example PRO4:** [forkexec.c](#)

```

main(void)
{
    pid_t pid;
    char *args[4];
    pid = fork();
    if (pid == 0)
        exec1 ("/usr/bin/paste", "paste", "f1", "f2",
NULL);
    args[0] = "comm";
    args[1] = "f1";
    args[1] = "f2";
    args[2] = NULL;
    execv ("/usr/bin/comm", args);
}

```

Example of usage:

```
% forkexec
```

→ **Example PROC5:** [execfileshare.c](#)

Exec preserves file descriptors as illustrated by next example.

```

main()
{
    int in;
    pid_t pid;
    system ("echo abc123 > junk");
}

```

```

in = open("junk", O_RDONLY);
pid = fork() ;
if (pid == 0)
    execl("./fileread", "fileread", 0);
else
    execl("./fileread", "fileread", 0);
}

```

The program exec a file called: [fileread.c](#)

```

main()
{
    char buf[1024];
    read(3, buf, 3);
    write(1, buf, 3);
    write(1, "\n", 1);
}

```

Example of usage:

```
% execfileshare
```

→ Example PROC6: [execsignal.c](#)

Exec preserves some signals (e.g., `SIG_IGN`).

```

main()
{
    pid_t pid;
    signal(SIGINT, SIG_IGN);
    pid = fork();
    if (pid == 0)
        execl("./waitsig1", "waitsig1", 0);
    else
        execl("./waitsig2", "waitsig2", 0);
}

```

waitsig1.c

```

main()
{
    pause();
}

```

waitsig2.c

```

main()
{
    signal(SIGINT, SIG_DFL);
    pause();
}

```

Testing:

`% ps -a`
to find the pid of `waitsig1` and `waitsig2` and Use

`% kill -INT <pid>`
to send an `INT` signal to `waitsig1` and `waitsig2`

➤ **WAIT:** for a child.

→ **Example PROC7:** [forkexecwait.c](#)

```

main()
{
    pid_t pid;
    char *args[4];
    int status;
    pid = fork();
    if (pid == 0)
        execl("/bin/echo", "echo", "Today is:",
NULL);
    wait(NULL) ;
    system("date");
    pause();
}

```

Example of usage:

```

% forkexecwait
Remove wait and you will see that the child
process
is <defunct>, i.e., zombie process.

```

➤ **Simple Shell: wahab's shell (wash)**

[wash.c](#)

```

main()
{
    pid_t pid;
    char command[BUFSIZ];
    for (;;) {
        printf("wash> ");
        fgets(command, sizeof(command), stdin);

        if ((pid = fork()) == 0)
            execlp(command, command, 0);

        wait(NULL);
    }
}

```

Examples use:

```

% wash
wash: ls
wash: who
wash: execsignal
(CTRL-C will kill the command & the shell).

```

➤ **Putting it all together: A reasonable shell (*rash*):**

- It accepts commands with **arguments**.
- It handles input/output **redirection**: **< file** and **> file**.
- It ignores INT and QUIT **signals** that are sent to children.

Example use:

```

% rash
rash: ls -lt
rash: execsignal

```

*(CTRL-C will kill the command not the shell).
compare this with wahab's shell (wash)!*

```
rash: ls -lt > lsfile
rash: wc < lsfile
rash: cb < infile.c > infile.b
```

rash.c

```
main(void)
{
    for (;;) {
        printf("rash:");
        fgets(command, sizeof(command), stdin);
        /* split the command into words */
        n = bufsplit(command, args);
        args[n] = NULL;

        infile = NULL;
        outfile = NULL;
        for (cp = args; *cp != NULL; cp++) {
            if (strcmp(*cp, "<") == 0) {
                *cp++ = NULL;
                infile = *cp;
            }
            else if (strcmp(*cp, ">") == 0) {
                *cp++ = NULL;
                outfile = *cp;
            }
        }
        status = execute(args, infile, outfile);
    }
}

int execute(char **args, char *infile, char *outfile)
{
    infd = -1;
    outfd = -1;
    if (infile != NULL)
        infd = open(infile, O_RDONLY)
    if (outfile != NULL)
        outfd = creat(outfile, 0666);

    /* Ignore keyboard signals; and SIG_CHLD
signals */
    signal(SIGINT, SIG_IGN);
    signal(SIGQUIT, SIG_IGN);
    signal(SIGCHLD, SIG_IGN);
```

```

pid = fork();
if (pid == 0) {
    signal(SIGINT, SIG_DFL);
    signal(SIGQUIT, SIG_DFL);
    signal(SIGCHLD, SIG_DFL);
    if (infd > 0)
        dup2(infd, 0);
    if (outfd > 0)
        dup2(outfd, 1);
    execvp(*args, args);
}

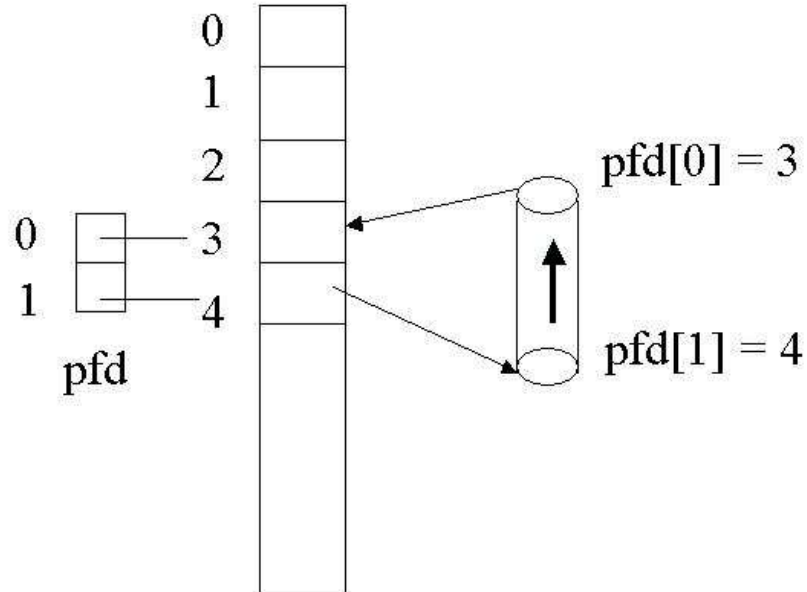
wait(NULL);
    signal(SIGINT, SIG_DFL);
    signal(SIGQUIT, SIG_DFL);
    signal(SIGCHLD, SIG_DFL);
    close(outfd);
    close(infd);
}

size_t bufsplit(char *buf, size_t n, char **a)
{.....
}

```

Interprocess Communications

Pipes: [Figure for illustrating pipe creation](#)



```
int pfd[2];
pipe (pfd);
```

→ **Example PIPE1:** *child writes to parent*

pipedate.c

```
main()
{
    pid_t pid; int pfd[2]; char line[64];

    pipe(pfd);
    pid = fork();
    if (pid == 0) {
        close(pfd[0]); // no need for read end
        dup2(pfd[1], 1);
        close(pfd[1]); // since it is duplicated
        execl("/bin/date", "date", 0);
    }
    close(pfd[1]); // no need for write end
    read (pfd[0], line, sizeof(line));
    printf("date from child is: %s\n", line);
    close(pfd[0]);
    waitpid(NULL);
}
```

```
    exit(0);
}
```

→ **Example PIPE2:** *parent writes to child*

pipemail.c

```
main()
{
    pid_t pid;
    int pfd[2];
    char *username = cuserid(NULL);
    char *msg = "Greetings | Salam| Shalom| Hola|
Salut| Konichiwa | Ciao"

    pipe(pfd);
    pid = fork();
    if (pid == 0) {
        close(pfd[1]); // no need for write end
        dup2(pfd[0], 0);
        close(pfd[0]); // since it is duplicated
        execlp("Mail", "Mail", "-s", "pipemail",
username, 0);
    }
    close(pfd[0]); // no need for read end
    write(pfd[1], msg, strlen(msg));
    close(pfd[1]);
    waitpid(NULL);
    exit(0);
}
```

Pipe handling shell: *pash: extending rash to handle pipe.*

Examples:

```
% pash
pash: ls | wc
pash: ls -lt > f1
pash: cat < f1 | wc > f2
pash: cat < f2 > /dev/tty
```

pash.c

```

main()
{
    for (;;) {
        args2 = args1;
        printf("pash: ");

        if (fgets(command, sizeof(command), stdin) ==
NULL) {
            putchar('\n');
            exit(0);
        }

        n = bufsplit(command, NARGS, args1);
        args1[n] = NULL;

        infile = NULL;
        outfile = NULL;

        for (cp = args1; *cp != NULL; cp++) {
            if (strcmp(*cp, "<") == 0) {
                *cp++ = NULL;
                infile = *cp;
            }
            else if (strcmp(*cp, ">") == 0) {
                *cp++ = NULL;
                outfile = *cp;
            }
            else if (strcmp(*cp, "|") == 0) {
                *cp++ = NULL;
                args2 = cp;
                piping = 1;
            }
        }

        status = execute(args1, args2, piping, infile,
outfile);
    }
}

int
execute(char **args1, char **args2, int piping, char
*infile, char *outfile)
{
    int status;
    pid_t p, pid1, pid2;
    int infd, outfd;

```

```

extern int errno;
infd = -1;
outfd = -1;

if (infile != NULL) {
    if ((infd = open(infile, O_RDONLY)) < 0) {
        perror(infile);
        return(-1);
    }
}

if (outfile != NULL) {
    if ((outfd = creat(outfile, 0666)) < 0) {
        perror(outfile);
        close(infd);
        return(-1);
    }
}

signal(SIGINT, SIG_IGN);
signal(SIGQUIT, SIG_IGN);
signal(SIGCHLD, SIG_IGN);

pipe(pipefd);

pid2 = fork();
if (piping && pid2 != 0)
    pid1 = fork();

if (pid2 == 0) {
    signal(SIGINT, SIG_DFL);
    signal(SIGQUIT, SIG_DFL);
    signal(SIGCHLD, SIG_DFL);

    if ((infd > 0) && !piping) {
        dup2(infd, 0);
        close (infd);
    }

    if (outfd > 0) {
        dup2(outfd, 1);
        close (outfd);
    }

    if (piping) {
        dup2(pipefd[0], 0);
        close(pipefd[0]);
    }
}

```

```

        close(pipefd[1]);
        execvp(*args2, args2);
        exit(0);
    } else
        execvp(*args1, args1);
    exit(0);
}
if ( piping && (pid1 == 0) ) {
    signal(SIGINT, SIG_DFL);
    signal(SIGQUIT, SIG_DFL);
    signal(SIGCHLD, SIG_DFL);
    dup2(pipefd[1], 1);
    close(pipefd[1]);
    close(pipefd[0]);

    if ( infd > 0 ){
        dup2(infd, 0);
        close (infd);
    }
    execvp(*args1, args1);
    exit(0);
}

close(pipefd[0]);
close(pipefd[1]);
close(outfd);
close(infd);

while (waitpid(pid2, &status, 0) < 0) {
    if (errno != EINTR) {
        status = -1;
        break;
    }
}

signal(SIGINT, SIG_DFL);
signal(SIGQUIT, SIG_DFL);
signal(SIGCHLD, SIG_DFL);

return(status);
}

```