한양대학교 HANYANG UNIVERSITY

File I/O

System Programming

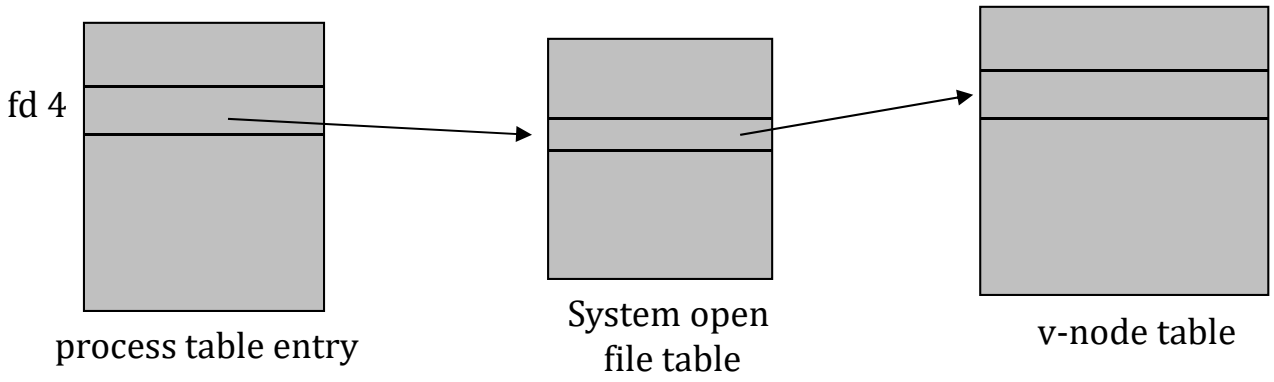
2019 여름 계절학기

한양대학교 공과대학 컴퓨터소프트웨어학부
홍석준

I. Contents

- **Unbuffered I/O (part of POSIX.1 and the Single UNIX Specification, but not of ISO C)**
  - `open`, `read`, `write`, `lseek`, and `close`
- **File sharing among multiple processes**
  - An atomic operation
  - `dup`, `fcntl`, `sync`, `fsync`, and `ioctl`

- A non-negative integer (0 .. OPEN_MAX) returned by open or `create`
- `<unistd.h>` in POSIX-compliant systems
  - 0 – STDIN_FILENO
  - 1 – STDOUT_FILENO
  - 2 – STDERR_FILENO



I. open Function

```
#include <fcntl.h>
int open(const char *pathname, int oflag, … /* , mode_t
   mode */);
```

❑ oflag argument

- One and only one of the three
  - O_RDONLY, O_WRONLY, O_RDWR
- Optional
  - O_APPEND, O_CREAT, O_EXCL, O_TRUNC, O_NOCTTY, O_NONBLOCK,
  - SUS I/O Options: O_DSYNC, O_RSYNC, O_SYNC

❑ Return Value: guaranteed to be the lowest-numbered unused descriptor

I. creat and close Functions

```
#include <fcntl.h>
int creat(const char *pathname, mode_t mode);
```

- Equivalent to `open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);`

```
#include <unistd.h>
int close(int filedes);
```

- When a process terminates, all open files are automatically closed by the kernel.

I. `read` and `write` Functions

```
#include <unistd.h>
ssize_t read(int filedes, void *buf, size_t nbytes);
```

- ❑ **The number of bytes read is returned. At EOF, 0 is returned.**
- ❑ **Less bytes than the requested are read**
  - EOF, a terminal device (per line reading), a network, a pipe or FIFO, a record-oriented device (e.g. a magnetic tape), and interrupted by a signal

```
#include <unistd.h>
ssize_t write(int filedes, const void *buf, size_t nbytes);
```

- ❑ **The number of bytes written is returned**

I. lseek Function

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

`off_t lseek(int` *filedes*`, off_t` *offset*`, int` *whence*`);`

- ❑ **"file offset" in bytes from the beginning**
- ❑ ***whence*:**
  - SEEK_SET: *offset* from the beginning of the file
  - SEEK_CUR: current file offset + *offset*
  - SEEK_END: file size + *offset*
- ❑ **To determine the current offset**
  - `off_t curpos = lseek(fd, 0, SEEK_CUR);`
  - For a pipe, FIFO or socket, it returns -1 and sets `errno` to `ESPIPE`

❑ Program 3.1

```
$ ./a.out < /etc/motd
seek OK
$ cat < /etc/motd | ./a.out
cannot seek
$ ./a.out < /var/spool/cron/FIFO
cannot seek
```

❑ Program 3.2 (creating a hole in a file)

```
$ ./a.out
$ ls –l file.hole
-rw-r--r--   1 sar   16394 Nov 25 01:01 file.hole
$ od –c file.hole
“abcdefghij\0…\0ABCDEFGHIJ”
$ ls –ls file.hole file.nohole
  8 –rw-r--r-- r sar 16394 Nov 25 01:01 file.hole
 20 –rw-r--r-- r sar 16394 Nov 25 01:01 file.nohole
```

Program 3.1

```
#include "apue.h"
int
main(void)
{
  if (lseek(STDIN_FILENO, 0, SEEK_CUR) == -1)
    printf("cannot seek\n");
  else
    printf("seek OK\n");
  exit(0);
}
```

Program 3.2

```
#include "apue.h"
#include <fcntl.h>
char buf1[] = "abcdefghij";  char buf2[] = "ABCDEFGHIJ";
int main(void)
{  int fd;
  if ((fd = creat("file.hole", FILE_MODE)) < 0)
    err_sys("creat error");
  if (write(fd, buf1, 10) != 10)
    err_sys("buf1 write error");
  /* offset now = 10 */
  if (lseek(fd, 16384, SEEK_SET) == -1)
    err_sys("lseek error");
  /* offset now = 16384 */
  if (write(fd, buf2, 10) != 10)
    err_sys("buf2 write error");
  /* offset now = 16394 */
  exit(0);
}
```

I/O Efficiency

- Program 3.4

- BUFFSIZE 4096
  - Reading a 103,316,352-byte file, using 20 different buffer sizes
  - Tested against the Linux `ext2` file system with 4,096-byte blocks

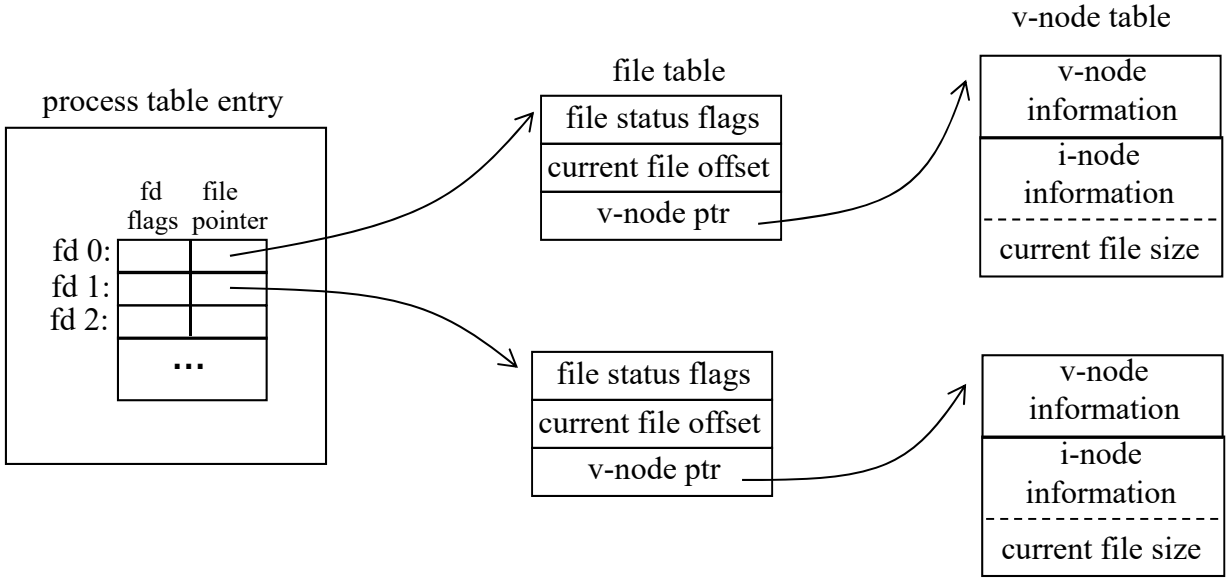
BUFFSIZE	User CPU (seconds)	System CPU (seconds)	Clock time (seconds)	#loops
1	124.89	161.65	288.64	103,316,352
2	63.10	80.96	145.81	51,658,176
4	31.84	40.00	72.75	25,829,088
8	15.17	21.01	36.85	12,914,544
16	7.86	10.27	18.76	6,457,272
32	4.13	5.01	9.76	3,228,636
64	2.11	2.48	6.76	1,614,318
128	1.01	1.27	6.82	807,159
256	0.56	0.62	6.80	403,579
512	0.27	0.41	7.03	201,789
1,024	0.17	0.23	7.84	100,894
2,048	0.05	0.19	6.82	50,447
4,096	0.03	0.16	6.86	25,223
8,192	0.01	0.18	6.67	12,611
16,384	0.02	0.18	6.87	6,305
32,768	0.00	0.16	6.70	3,152
65,536	0.02	0.19	6.92	1,576
131,072	0.00	0.16	6.84	788
262,144	0.01	0.25	7.30	394
524,288	0.00	0.22	7.35	198

Program 3.4

```
#include "apue.h"
#define BUFFSIZE 4096
int
main(void)
{  int n;
   char buf[BUFFSIZE];
   while ((n = read(STDIN_FILENO, buf, BUFFSIZE)) > 0)
     if (write(STDOUT_FILENO, buf, n) != n)
       err_sys("write error");
   if (n < 0)
     err_sys("read error");
   exit(0);
}
```

I. File Sharing

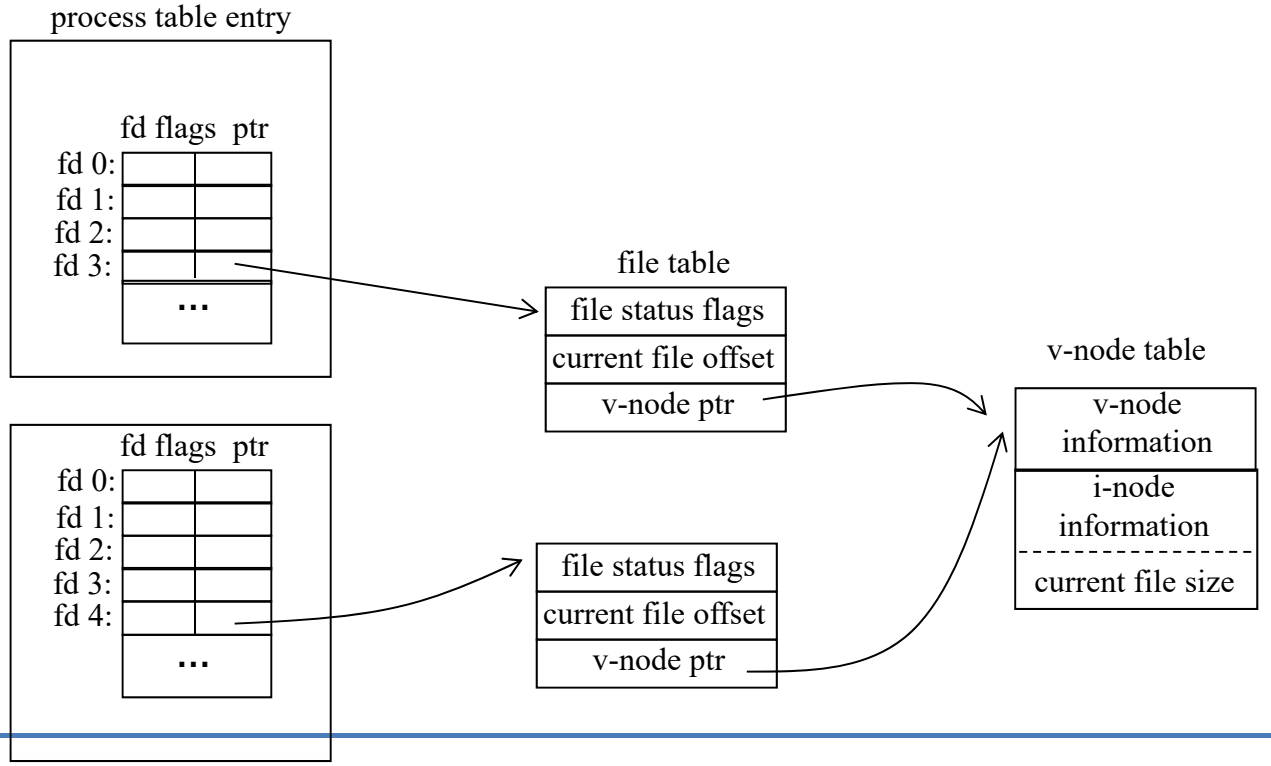
- **An open file descriptor table within each process table entry**
- **Kernel *file table* for all open files**
- ***v-node table***
  - v-node: filesystem independent portion of the i-node
  - i-node: file owner, size, dev, ptrs to data blocks, etc



I. File Sharing

❑ Each process gets its own current offset.

- Each `write` changes the offset (and the size in the i-node table.)
- For each `write` to a file opened with O_APPEND flag set, the size in the i-node table → the offset
- `lseek(fd, 0, SEEK_END)`: the size in i-node table → the offset
- `dup` and `fork`: More than one file descriptor entry pointing to the same file table entry



I. Atomic Operations

- **When a process wants to append to the end of a file,**

```
if (lseek(fd, 0L, SEEK_END) < 0)
   err_sys("lseek error");
if (write(fd, buf, 100) != 100)
   err_sys("write error");
```

- **Assuming process A and B that append to the same file,**
  - A's `lseek` sets its offset to 1500.
  - B's does the same thing, and calls a `write` to increase its offset to 1600. Kernel also updates the size in the v-node table entry.
  - A calls a `write`, which will overwrite the B's data.

- **Any operation that requires more than one function call can not be atomic.**

- **`write` to a file opened with O_APPEND flag → should be atomic**

I. Atomic Operations

- **The Single UNIX Specification**

```
#include <unistd.h>

ssize_t pread(int filedes, void *buf, size_t nbytes, off_t offset);

ssize_t pwrite(int filedes, void *buf, size_t nbytes, off_t offset);
```

- **`pread` equivalent to calling `lseek` followed by a call to `read`**
- **`pwrite` equivalent to calling `lseek` followed by a call to `write`**

❑ open(fd, …|O_CREAT|O_EXCL, *modes*)

– The check for the existence of the file and the creation of the file is performed as an **atomic operation.**

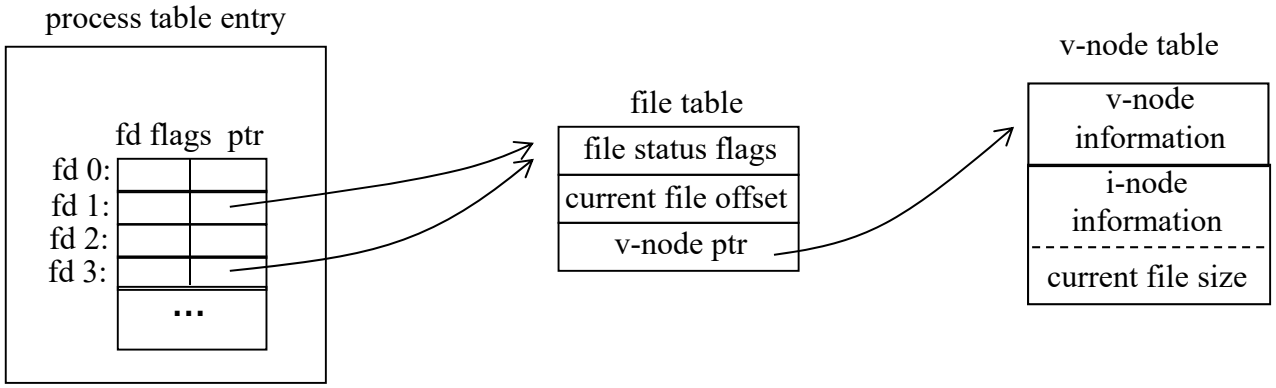
❑ Otherwise, we might try

```
if ((fd = open(pathname, O_WRONLY)) < 0){
  if (errno == ENOENT) {
        if ((fd = creat(pathname, mode)) < 0)
              err_sys("creat error");
  } else {
        err_sys("open error");
  }
}
```

I. dup and dup2 Functions

```
#include <unistd.h>
int dup(int filedes);
int dup2(int filedes, int filedes2);
```

- dup returns the lowest-numbered available file descriptor.
- dup2 makes a copy of *filedes* into *filedes2* (if opened, it first closes *filedes2*.)



I. sync, fsync, and fdatasync Functions

```
#include <unistd.h>
int fsync(int filedes);
int fdatasync(int filedes);
void sync(void);
```

- ***Delayed write*** **via a buffer cache**
  - Data is copied into one of its buffers and queued for writing to disk at some later time.
- **sync simply queues all the modified block buffers for writing and returns.**
- **fsync waits for disk writes, including file attributes, to complete before returning. (only to a single file)**
- **fdatasync affects only the data portions of a file.**

I. fcntl Function

❑ **can change the properties of a file**

```
#include <fcntl.h>

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

❑ ***cmd* argument**

- duplicate an existing descriptor (F_DUPFD)
  - Its FD_CLOEXEC file descriptor flag is cleared.
- get/set file descriptor flags (F_GETFD, F_SETFD)
  - Currently only one file descriptor flag is defined: FD_CLOEXEC flag
- get/set file status flags (F_GETFL, F_SETFL)
  - the flags used by open
- get/set asynchronous I/O ownership (F_GETOWN, F_SETOWN)
- get/set record lock (F_GETLK, F_SETLK, F_SETLKW)

I. fcntl Function

- Program 3.10 – Print file flags for a specified descriptor.
- Program 3.11 – Turn on one or more of the file status flags.
- Figure 3.12 (Linux ext2 timing results)

Operation	User CPU (seconds)	System CPU (seconds)	Clock time (seconds)
read time from Figure 3.5 for BUFFSIZE = 4,096	0.03	0.16	6.86
normal write to disk file	0.02	0.30	6.87
write to disk file with O_SYNC set	0.03	0.30	6.83
write to disk followed by fdatasync	0.03	0.42	18.28
write to disk followed by fsync	0.03	0.37	17.95
write to disk with O_SYNC set followed by fsync	0.05	0.44	17.95

- Figure 3.13 (Mac OS X timing results)

Operation	User CPU (seconds)	System CPU (seconds)	Clock time (seconds)
write to /dev/null	0.06	0.79	4.33
normal write to disk file	0.05	3.56	14.40
write to disk file with O_FSYNC set	0.13	9.53	22.48
write to disk followed by fsync	0.11	3.31	14.12
write to disk with O_FSYNC set followed by fsync	0.17	9.14	22.12

Program 3.10

```
#include "apue.h"
#include <fcntl.h>
int main(int argc, char *argv[])
{  int val;
  if (argc != 2)
    err_quit("usage: a.out <descriptor#>");
  if ((val = fcntl(atoi(argv[1]), F_GETFL, 0)) < 0)
    err_sys("fcntl error for fd %d", atoi(argv[1])
);
  switch (val & O_ACCMODE) {
  case O_RDONLY:
    printf("read only"); break;
  case O_WRONLY:
    printf("write only"); break;
  case O_RDWR:
    printf("read write"); break;
  default:
    err_dump("unknown access mode");
  }
  if (val & O_APPEND)
    printf(", append");
  if (val & O_NONBLOCK)
    printf(", nonblocking");
#if defined(O_SYNC)
  if (val & O_SYNC)
    printf(", synchronous writes");
#endif
#if !defined(_POSIX_C_SOURCE) && defined(O_
FSYNC)
  if (val & O_FSYNC)
    printf(", synchronous writes");
#endif
  putchar('\n');
  exit(0);
}
```

Program 3.11

```
#include "apue.h"
#include <fcntl.h>
void
set_fl(int fd, int flags) /* flags are file status flags to turn on */
{  int val;
   if ((val = fcntl(fd, F_GETFL, 0)) < 0)
      err_sys("fcntl F_GETFL error");
   val |= flags;/* turn on flags */
   if (fcntl(fd, F_SETFL, val) < 0)
      err_sys("fcntl F_SETFL error");
}
```

I.ioctl Function

```
#include <unistd.h> /* SVR4 */
#include <sys/ioctl.h>     /* BSD and Linux */
#include <stropts.h>    /* XSI STREAMS */
int ioctl(int filedes, int request, …);
```

- The catchall for I/O operations
- Each device driver defines its own set of `ioctl` commands.
- FreeBSD `ioctl` operations

Category	Constant Names	Header	Number of `ioctls`
disk labels	DIOxxx	`<sys/disklabel.h>`	6
file I/O	FIOxxx	`<sys/filio.h>`	9
mag tape I/O	MTIOxxx	`<sys/mtio.h>`	11
socket I/O	SIOxxx	`<sys/sockio.h>`	60
terminal I/O	TIOxxx	`<sys/ttycom.h>`	44

Q and A