I/O

Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Spring 2023

File I/O

- `FILE` structure
  - A particular structure, of which members describe the current state of a file, defined in `stdio.h`

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

```
FILE *variable_name;
```

- Declaration of a pointer variable to FILE structure (or file pointer)

```
fopen()
```

- File open: file pointer initialization

```
fprintf(), fscanf(),
fputc(), fgetc(), ...
```

- Output data to the file or read data from the file

```
fclose()
```

- File close

Opening and Closing a File

- File pointer declaration
  - `FILE *fp;`

- Open a file
  - `fp = fopen("filename", "mode");`

- Close a file
  - `fclose(fp);`

File Modes

- When a mode ends with a '+' character, the file is to be opened for both reading and writing

Mode	Meaning
`"r"`	Open text file for reading
`"w"`	Open text file for writing
`"a"`	Open text file for appending
`"rb"`	Open binary file for reading
`"wb"`	Open binary file for writing
`"ab"`	Open binary file for appending
`"r+"`	Open text file for reading and writing
`"w+"`	Open text file for writing and reading
`"a+"`	Open text file for appending and reading

Formatted File I/O

- Function prototypes is defined in `<stdio.h>`
  - `int fscanf(FILE *fp, const char *format, ...);`
  - `int fprintf(FILE *fp, const char *format, ...);`

```c
#include <stdio.h>

void fileio(void) {
    FILE *ifp, *ofp;                /* file pointer declaration */

    ifp = fopen("in_file", "r");    /* file open */
    ofp = fopen("out_file", "w");

    fscanf(ifp, control_string, other_arguments);
    fprintf(ofp, control_string, other arguments);

    fclose(ifp);                    /* file close */
    fclose(ofp);
}
```

Example

```
#include <stdio.h>

void main(void)
{
    FILE *ifp, *ofp;
    int  a, sum = 0;

    ifp = fopen("infile", "r");
    ofp = fopen("outfile", "w");

    while (fscanf(ifp, "%d", &a) ==  1)
        sum += a;
    fprintf(ofp, "The sum is %d.\n", sum);

    fclose(ifp);
    fclose(ofp);
}
```

`stdin` / `stdout` / `stderr`

- Standard I/O is automatically opened at the start of program and automatically closed at the completion of program
  - `stdin`: standard input file (keyboard)
  - `stdout`: standard output file (screen)
  - `stderr`: standard error file (screen)
  - No need to open and close these files
- Defined in `<stdio.h>`
  - `printf(...);` is equal to `fprintf(stdout, ...);`
  - `scanf(...);` is equal to `fscanf(stdin, ...);`
  - `#define getchar()       getc(stdin);`
  - `#define putchar(c)      putc(c, stdout);`

Character File I/O

- Read a character from file
  - `c = getc(fp);` /* macro */
  - `c = fgetc(fp);` /* function */
  - EOF is returned if the end of file indicator or the error indicator has been set

- Write a character to file
  - `putc(c, fp);` /* macro */
  - `fputc(c, fp);` /* function */
  - If successful, `c` is returned
  - Otherwise, the error indicator is set and EOF is returned

EOF

- End-Of-File
  - Symbolic constant defined as the specific numeric value in `stdio.h`
  - `#define   EOF        -1`

- `int feof(FILE *fp);`
  - Checks EOF
  - If the end-of-file is encountered, the end-of-file indicator is set
  - `feof(fp)` returns non-zero value if the end-of-file indicator has been set

Example: Copying a File

```c
#include <stdio.h>

int main(void)
{
    FILE *srcfp, *destfp;
    int c;

    srcfp = fopen("in.txt", "r");
    destfp = fopen("out.txt", "w");
    while ((c = getc(srcfp)) != EOF)
        putc(c, destfp);
    fclose(srcfp);
    fclose(destfp);
    return 0;
}
```

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    FILE *srcfp, *destfp;
    int c;

    srcfp = fopen(argv[1], "r");
    destfp = fopen(argv[2], "w");
    while ((c = getc(srcfp)) != EOF)
        putc(c, destfp);
    fclose(srcfp);
    fclose(destfp);
    return 0;
}
```

Error Handling

- `int ferror(FILE *fp);`
  - `ferror(fp);` returns non-zero value if the error indicator has been set for the file associated with `fp`

- `void exit(int status);`
  - `exit();` causes normal process termination
  - Returns the value of `status` to the parent process
    - 0: the program is successfully terminated
    - Non-zero: the program did not execute successfully

- (cf.) `return` statement: causes function termination
  - `return` in `main()` ➔ program termination

Example: Double Spacing a File

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    FILE *infp, *outfp;
    int c;

    if (argc != 3) exit(1);
    infp = fopen(argv[1], "r");
    outfp = fopen(argv[2], "w");
    while ((c = getc(infp)) != EOF) {
        putc(c, outfp);
        if (c == '\n') putc('\n', outfp);
    }
    fclose(infp);
    fclose(outfp);
    exit(0);            /* return 0; */
}
```

Line I/O: `fgets()`

- `char *fgets(char *line, int n, FILE *fp);`
  - Line-oriented input function
  - Reads at most (n – 1) characters from the file associated with `fp` into the array pointed to by `line` (buffer)
  - If a newline is read or an end of file is encountered, no additional characters are read from the file
  - `'\0'` is inserted automatically at the end of array
  - If the end of file is encountered right at the start, returns NULL
  - Otherwise, `line` is returned

fgets()

- Doubling spacing a file

in.txt

```
hello,
  world
```

output

```
hello,

  world

```

```
#include <stdio.h>

char *fgets(char *s, int n, FILE *fp) {
    int c;
    char *cs;

    cs = s;
    while ((c = getc(fp)) != EOF && --n > 0)
        if ((*cs++ = c) == '\n')
            break;
    *cs = '\0';
    return (c == EOF && cs == s)? NULL : s;
}
```

```
#define MAXBUF  80

int main(void)
{
    FILE *fp;
    char buf[MAXBUF];

    fp = fopen("in.txt", "r");
    while (fgets(buf, MAXBUF, fp))
        printf("%s\n", buf);
    fclose(fp);
}
```

Line I/O: fputs()

- `int fputs(char *line, FILE *fp);`
  - Line-oriented output function
  - Copy the null-terminated string `line` (except null character itself) into the file associated with `fp` (i.e., appends a newline to the file)
  - A successful call returns a nonnegative value; otherwise EOF

```
int fputs(char *s, FILE *fp)
{
    int c;

    while (c = *s++)
        putc(c, fp);
    return ferror(fp)? EOF : 0;
}
```

Random File I/O: `fseek()`

- `int fseek(FILE *fp, long offset, int whence);`
  - Sets the file position indicator to a value that is `offset` bytes from `whence`
  - `whence`: `SEEK_SET`: the beginning of the file
    `SEEK_CUR`: the current position
    `SEEK_END`: the end of the file
  - Examples:

```
fseek(fp, 0, SEEK_SET);     /* the beginning of the file */
fseek(fp, 0, SEEK_END);     /* the end of the file */
fseek(fp, n, SEEK_SET);     /* the beginning position + n */
fseek(fp, n, SEEK_CUR);     /* the current position + n */
fseek(fp, -n, SEEK_END);    /* the end position – n */
```

  - If the function call is successful, the end-of-file indicator is cleared and zero is returned

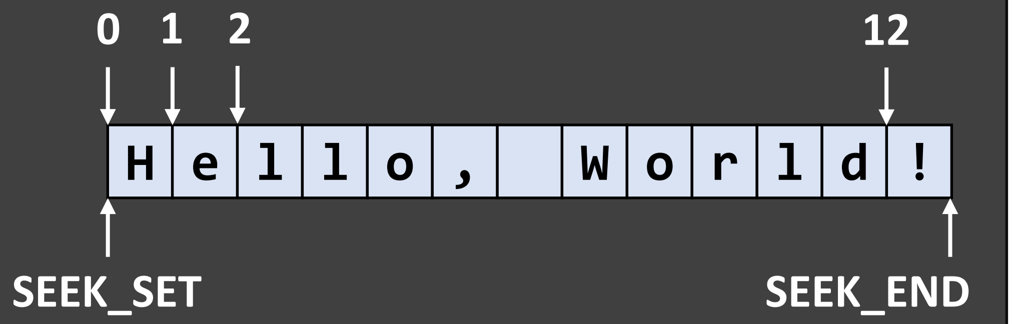
Random File I/O: `rewind()` and `ftell()`

- `void rewind(FILE *fp);`
  - Sets the file position indicator to the beginning of the file
  - Equivalent to `fseek(fp, 0, SEEK_SET);`

- `long ftell(FILE *fp);`
  - Returns the current value of the file position indicator (the number of bytes from the beginning of the file)
  - An unsuccessful call returns -1

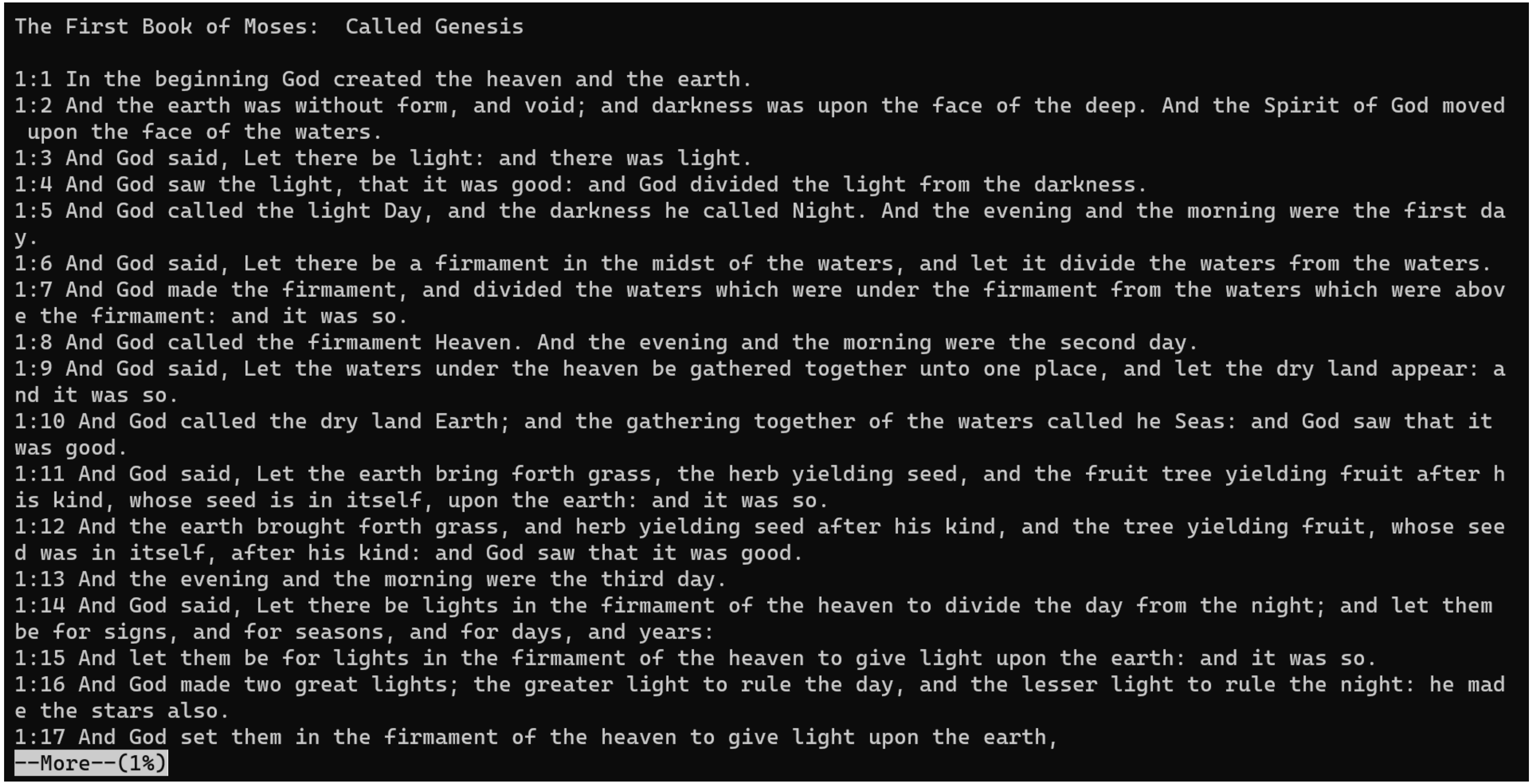
Example: Writing a File Backwards

```
#include <stdio.h>

void main(void) {
    int c;
    FILE *fp;

    fp = fopen("in.txt", "r");
    fseek(fp, 0, SEEK_END);
    if (ftell(fp) > 0) {
        fseek(fp, -1, SEEK_CUR);
        while (1) {
            c = getc(fp);
            putchar(c);
            if (ftell(fp) > 1) fseek(fp, -2, SEEK_CUR);
            else break;
        }
    }
    fclose(fp);
}
```



Finding Top-*k* Words (1)

- Sample file: `genesis.txt`



```
The First Book of Moses:  Called Genesis

1:1 In the beginning God created the heaven and the earth.
1:2 And the earth was without form, and void; and darkness was upon the face of the deep. And the Spirit of God moved
 upon the face of the waters.
1:3 And God said, Let there be light: and there was light.
1:4 And God saw the light, that it was good: and God divided the light from the darkness.
1:5 And God called the light Day, and the darkness he called Night. And the evening and the morning were the first da
y.
1:6 And God said, Let there be a firmament in the midst of the waters, and let it divide the waters from the waters.
1:7 And God made the firmament, and divided the waters which were under the firmament from the waters which were abov
e the firmament: and it was so.
1:8 And God called the firmament Heaven. And the evening and the morning were the second day.
1:9 And God said, Let the waters under the heaven be gathered together unto one place, and let the dry land appear: a
nd it was so.
1:10 And God called the dry land Earth; and the gathering together of the waters called he Seas: and God saw that it
was good.
1:11 And God said, Let the earth bring forth grass, the herb yielding seed, and the fruit tree yielding fruit after h
is kind, whose seed is in itself, upon the earth: and it was so.
1:12 And the earth brought forth grass, and herb yielding seed after his kind, and the tree yielding fruit, whose see
d was in itself, after his kind: and God saw that it was good.
1:13 And the evening and the morning were the third day.
1:14 And God said, Let there be lights in the firmament of the heaven to divide the day from the night; and let them
be for signs, and for seasons, and for days, and years:
1:15 And let them be for lights in the firmament of the heaven to give light upon the earth: and it was so.
1:16 And God made two great lights; the greater light to rule the day, and the lesser light to rule the night: he mad
e the stars also.
1:17 And God set them in the firmament of the heaven to give light upon the earth,
--More--(1%)
```

Finding Top-*k* Words (2)

```c
/* topk.h */

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

#define  NSLOTS  5000

#define  DELIM   " ':,.;!?()\t\n"

struct record {
    char *word;
    int count;
    struct record *next;
};
```

```c
#include "topk.h"

int nwords = 0;
struct record *htable[NSLOTS];
struct record **wc;

int main(int argc, char *argv[])
{
  if (argc != 2) {
    printf("Usage: %s filename\n", argv[0]);
    return -1;
  }
  init_htable();
  read_words(argv[1]);
  find_k(10);
  return 0;
}
```

Finding Top-*k* Words (3)

```c
void read_words(char *filename)
{
  FILE *fp;
  char buf[512];

  fp = fopen(filename, "r");
  if (fp == NULL) {
    printf("Opening %s failed.\n",
           filename);
    exit(1);
  }

  while (fgets(buf, 512, fp))
    parse(buf);
  fclose(fp);
}
```

```c
void parse(char *buf)
{
  char *word;

  word = strtok(buf, DELIM);
  while (word != NULL) {
    if (!isdigit(*word)) {
      char *s = word;
      while (*s != '\0') {
        if (isupper(*s))
          *s = tolower(*s);
        s++;
      }
      if (insert_htable(word))
        nwords++;
    }
    word = strtok(NULL, DELIM);
  }
}
```

Finding Top-*k* Words (4)

```c
int insert_htable(char *word)
{
  struct record *p, **pp;

  if (p = lookup_htable(word)) {
    p->count++;
    return 0;     /* word exists */
  }

  pp = &htable[hash(word)];
  p = malloc(sizeof(struct record));
  p->word = malloc(strlen(word) + 1);
  strcpy(p->word, word);
  p->count = 1;

  p->next = (*pp)? *pp : NULL;
  *pp = p;
  return 1;       /* new word */
}
```

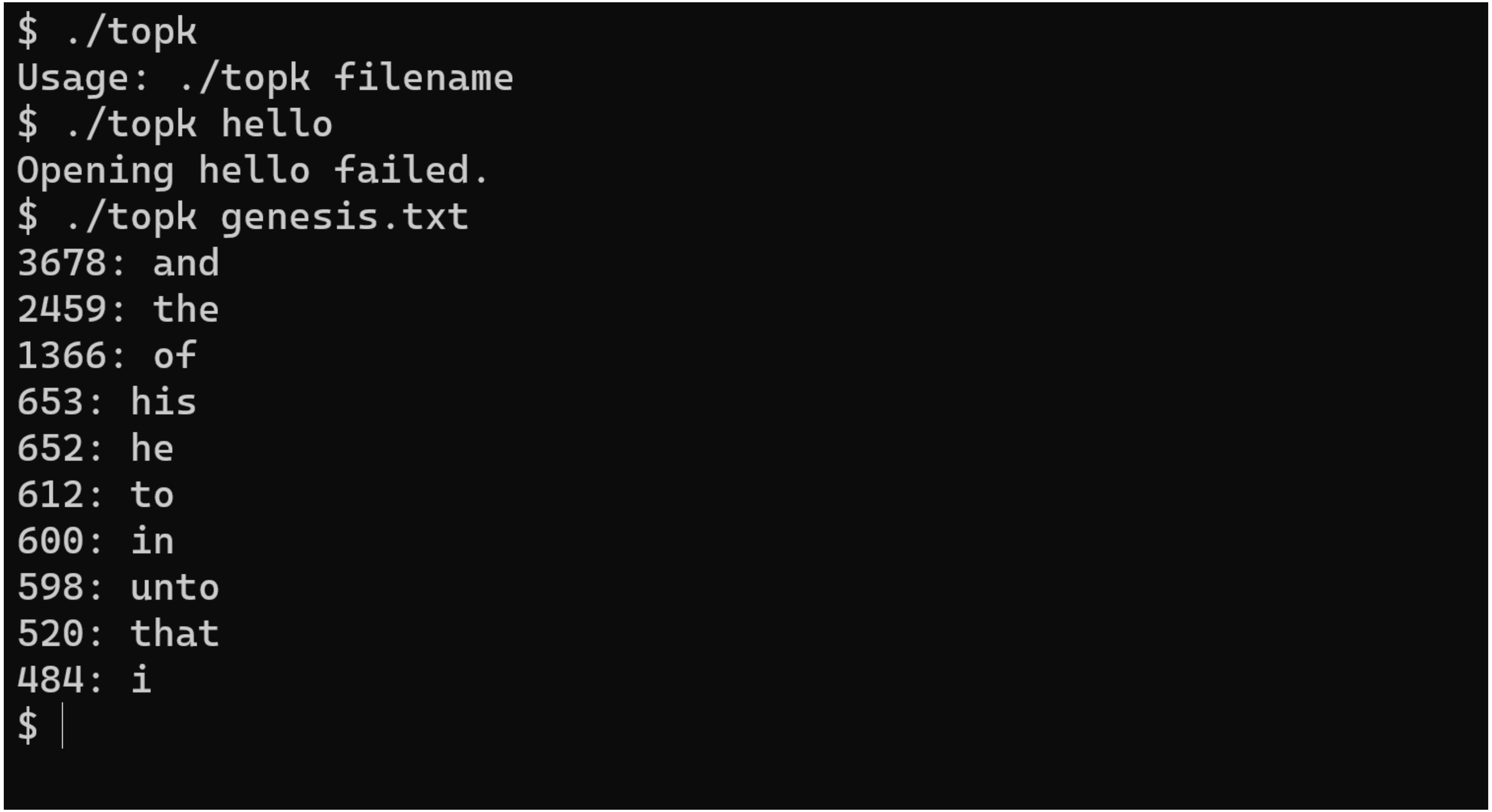
```c
struct record *lookup_htable(char *word)
{
  struct record *p, **pp;

  pp = &htable[hash(word)];
  p = *pp;
  while (p) {
    if (strcmp(word, p->word) == 0)
      return p;

    p = p->next;
  }
  return NULL;
}
```

Finding Top-*k* Words (5)

```c
void sort_wc(struct record **wc)
{
  for (int i = 0; i < nwords - 1; i++)
    for (int j = nwords - 1; j > i; j--)
      if (wc[j-1]->count < wc[j]->count)
        swap(&wc[j-1], &wc[j]);
}

void find_k(int k)
{
  wc = malloc(sizeof(struct record *)
              * nwords);
  get_wc(wc);
  sort_wc(wc);
  for (int i = 0; i < k; i++)
    printf("%d: %s\n",
           wc[i]->count, wc[i]->word);
}
```

```c
void get_wc(struct record **wc)
{
  struct record *p;
  int n = 0;

  for (int i = 0; i < NSLOTS; i++) {
    p = htable[i];
    while (p)
    {
      wc[n++] = p;
      p = p->next;
    }
  }
}
```

Finding Top-*k* Words (6)



```
$ ./topk
Usage: ./topk filename
$ ./topk hello
Opening hello failed.
$ ./topk genesis.txt
3678: and
2459: the
1366: of
653: his
652: he
612: to
600: in
598: unto
520: that
484: i
$ 
```

Low Level I/O – System Calls

- System call
  - A request for a service to an operating system's kernel

- `creat()`
- `open()`
- `close()`
- `read()`
- `write()`
- `lseek()`
- `...`

open()

- `int open(const char *pathname, int flags);`
- `int open(const char *pathname, int flags, mode_t mode);`
  - Opens the file specified by `pathname`
  - If the specified file does not exist, it may optionally be created if `O_CREAT` is specified in `flags`
  - If successful, returns a *file descriptor*, a small, nonnegative integer that is used in subsequent system calls to refer to the file
  - Mandatory `flags`: `O_RDONLY`: open for reading only
    `O_RDONLY`: open for writing only
    `O_RDWR`: open for reading and writing
  - Optional `flags`: `O_APPEND`, `O_CREAT`, `O_TRUNC`, etc.

creat()

- `int creat(const char *pathname, mode_t mode);`
  - Equivalent to `open(pathname, O_CREAT|O_WRONLY|O_TRUNC, mode);`
  - `mode`: the file mode (or permission) bits applied when a new file is created

Mnemonic	Bit representation	Octal representation
r--	100	04
-w-	010	02
--x	001	01
rw-	110	06
r-x	101	05
-wx	011	03
rwx	111	07

mode (owner, group, others)	Octal representation
rw-------	0600
rw-r-----	0640
rwxr-xr-x	0755
rwxrwxrwx	0777

`read()` and `write()`

- `ssize_t read(int fd, char *buf, int count);`
- `ssize_t write(int fd, char *buf, int count);`
  - `fd`: file descriptor
    - Non-negative integer to identify a file in an OS
    - 0: standard input, 1: standard output, 2: standard error
  - `read()` attempts to read *up to* `count` bytes into the buffer starting at `buf`
  - `write()` attempts to write *up to* `count` bytes from the buffer starting at `buf`
  - After each call, the file offset is incremented by the number of bytes actually read or written
  - Each call returns a count of the number of bytes transferred
  - NOTE: the number of bytes read or written can be less than `count`

Example: echo.c

```c
#include <unistd.h>

#define MAXBUF      80

/* copy standard input to standard output */

int main(void)
{
    char buf[MAXBUF];
    int n;

    while ((n = read(0, buf, MAXBUF)) > 0)
        write(1, buf, n);
}
```

Example: cp (1)

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

void main(int argc, char *argv[]) {
    int infd, outfd, nread, nwritten, n;
    char buf[BUFSIZ];       /* BUFSIZ defined in stdio.h */

    if (argc != 3) {
        fprintf(stderr, "Usage: cp src dest\n");
        exit(1);
    }
    if ((infd = open(argv[1], O_RDONLY)) < 0) {
        fprintf(stderr, "can't open %s\n", argv[1]);
        exit(2);
    }
```

Example: cp (2)

```c
    if ((outfd = creat(argv[2], 0666)) < 0) {
        fprintf(stderr, "can't create %s\n", argv[2]);
        exit(3);
    }

    while ((nread = read(infd, buf, BUFSIZ)) > 0) {
        nwritten = 0;
        while (nwritten < nread) {
            n = write(outfd, buf + nwritten, nread - nwritten);
            if (n < 0) {
                fprintf(stderr, "write failed\n");
                exit(4);
            }
            nwritten += n;
        }
    }
}
```

lseek()

- `off_t lseek(int fd, off_t offset, int whence);`
  - Repositions the file offset of the open file associated with `fd` to the argument `offset` according to the directive `whence` as follows:
    `SEEK_SET`: The file offset is set to `offset` bytes
    `SEEK_CUR`: The file offset is set to its current location + `offset` bytes
    `SEEK_END`: The file offset is set to the size of the file + `offset` bytes
  - Returns the resulting offset location in bytes from the beginning of the file

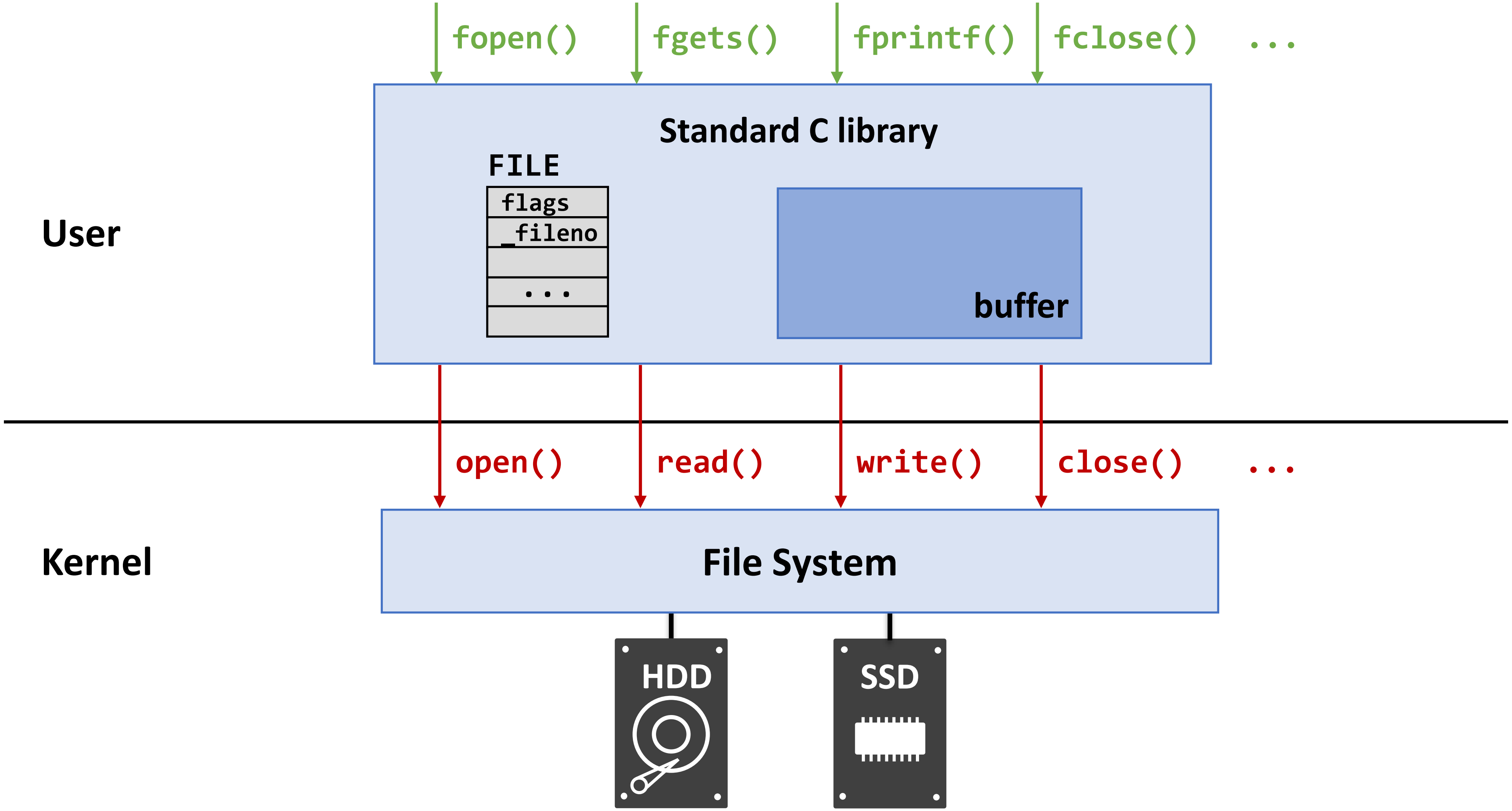
- `lseek()` allows the file offset to be set beyond the end of the file
  - If data is later written at this point, subsequent reads of the data in the gap (a "hole") return null bytes

Random Accessing a File

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

```c
int main(int argc, char *argv[]) {
    int fd, i;
    off_t len, pos;
    char buf[1];

    if (argc != 2) return -1;
    if ((fd = open(argv[1], O_RDONLY)) < 0)
        return -2;
    len = lseek(fd, 0, SEEK_END);
    srand(time(NULL));
    for (i = 0; i < 10; i++) {
        pos = rand() % len;
        lseek(fd, pos, SEEK_SET);
        read(fd, buf, 1);
        printf("%ld: %c\n", pos, buf[0]);
    }
    return 0;
}
```

Summary


fopen()
fgets()
fprintf()
fclose()
...
Standard C library
FILE
flags
_fileno
...
buffer
User
open()
read()
write()
close()
...
Kernel
File System
HDD
SSD