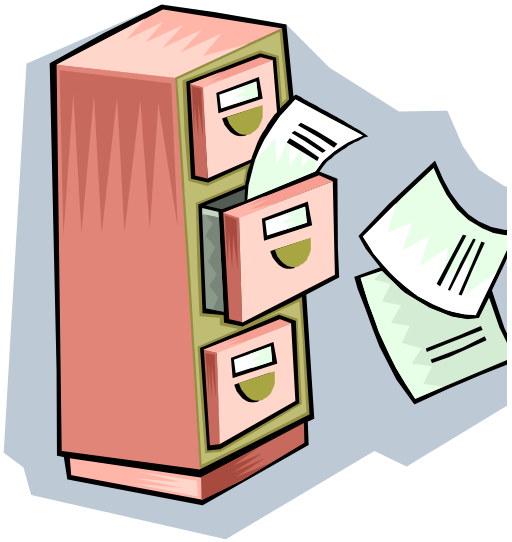
CS240: Programming in C

Lecture 13 si 14: Unix interface for working with files.

PURDUE UNIVERSITY

Working with Files (I/O)

- File system: specifies how the information is organized on the disk and can be accessed
  - Directories
  - Files
- In UNIX the following are **files**
  - Peripheral devices (keyboard, screen, etc)
  - Pipes (inter process communication)
  - Sockets (communication via computer networks)
- Files representation
  - Text files (human readable format)
  - Binaries (for example executables files)



Functions and System Calls

- **System calls**: services provided by the operating system.
- C Library provides support such that a user can invoke system calls through C functions.
- Example:
  - I/O operations (I/O access is slower than memory access)
  - Memory allocation

File Descriptors

- In order to perform operations on a file, the filed must be opened. Any opened file has associated a non-negative integer called **file descriptor**.
- For each program the operating system opens implicitly three files: **standard input, standard output and standard error**, that have associated the file descriptors 0, 1, 2 in this order.

C Library I/O Interface

- The C Library provides two mechanisms for working with files:
  - file descriptors: objects of type int

    int fd;
  - streams: FILE * objects.

    FILE * fp;

File Descriptors

- Primitive, low-level interface to input and output operations.
- Must be used for control operations that are specific to a particular kind of device.

Streams

- Higher-level interface, layered on top of the primitive file descriptor facilities.
- More powerful set of functions for performing actual input and output operations than the corresponding facilities for file descriptors.
- It is implemented in terms of file descriptors
  - the file descriptor can be extracted from a stream and then perform low-level operations directly on the file descriptor
  - a file can be open as a file descriptor and then make a stream associated with that file descriptor.

Standard Input, Output and Error

- For every program the operating system opens three files and provides pointers to them:
  - Standard input: stdin
  - Standard output: stdout
  - Standard error: stderr
- Normally, stdin is connected to the keyboard and stdout to the screen, but they may be redirected to pipes or files.

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

Formatted Output

```
printf(char *format, arg1, arg2, …);
sprintf(char *string, char *format, arg1, arg2, …);
fprintf(FILE *fp, char *format, arg1, arg2, …);
```

Examples:

```
int int_val;
char ch_val[10];
char buffer[256];
FILE *f;
…
printf("%d, %s, %p\n", int_val, ch_val, &int_val);
fprintf(stderr, "%d, %s\n", int_val, ch_val);
fprintf(f, "%d, %s\n", int_val, ch_val);
sprintf(buffer, "%d, %s\n", int_val, ch_val);
```

Formatted Input

```
scanf(char *format, arg1, arg2…);
sscanf(char *string, char *format, arg1, arg2, …);
```

Examples

```
double sum, v;
char ch[10];

sum = 0;
scanf("%s", ch);
while(scanf("%lf", &v) == 1) {
   printf("\t%.2f\n, sum += v");
}
```

The arguments to scanf and sscanf MUST be pointers !

Printing Error Messages

- Standard output stream (stdout) can be redirected and in that case the error messages will not be printed on the screen
- Solution: print to standard error file stream, stderr. Output written to stderr will appear on the screen even if stdout is redirected.

```
fprintf(stderr, "Error in function %s\n",
   func_name);
```

Example: output redirection

```
#include <stdio.h>

int main() {

  fprintf(stdout, "Printing to stdout with fprintf\n");
  printf("Printing to stdout with printf\n");

  fprintf(stderr, "Printing to stderr.\n");

  return 0;
}

./test
./test > a.txt
```

Example: input redirection

```
#include <stdio.h>

int main() {
  int c;

  while((c=getchar())!= EOF) {
    putchar(c);
  }
  return 0;
}
```

gcc -std=c99 stdin_test.c -o stdin_test

./stdin_test < a.txt

The Stream Interface – Open

```
#include <stdio.h>

FILE *fopen (const char *path, const char* mode);
```

- Opens the file path. Mode specifies if the file is opened for write, read or both, if is opened as a text or as a binary file.
- Returns NULL if it fails and *errno* is set to indicate the error.

The Stream Interface – Open - Mode

- **r** Open text file for reading. The stream is positioned at the beginning of the file.
- **r+** Open for reading and writing. The stream is positioned at the beginning of the file.
- **w** Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
- **w+** Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.
- **a** Open for appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.
- **a+** Open for reading and appending (writing at end of file). The file is created if it does not exist. The initial file position for reading is at the beginning of the file, but output is always appended to the end of the file.

The Stream Interface – Close

```
#include <stdio.h>

int fclose (FILE *stream);
```

- Closes the file stream.
- Returns 0 on suceess, otherwise, **EOF** is returned and *errno* is set to indicate the error
- Further access on the file results in undefined behavior.

```
int fileno (FILE *stream);
```

- Returns the file descriptor of the file stream.

Open/Close Streams – Examples

```
FILE *f1, *f2;
int fd;

/* open binary file for reading only. The stream is
   positioned at the beginning of the file. */
f1 = fopen("myfile", "rb");
fd = fileno(f1);
fprintf(stderr, "file descriptor: %d\n", fd);
fclose(f1);

/* open text file for reading and writing. The file is
   created if it does not exist, otherwise it is
   truncated. The  stream is positioned at the
   beginning of the file. */
f2 = fopen("otherfile", "w+");
fclose(f2);
```

Stream Interface – Positioning

```
#include <stdio.h>
int feof (FILE *stream);
```

- Tests the end-of-file indicator for the stream pointed to by stream, returns non-zero if it is set.

```
long int ftell (FILE *stream);
```

- Returns the current value of the file position indicator for the stream pointed to by stream.

```
void rewind (FILE *stream);
```

- Sets the file position indicator for the file pointed to by stream to the beginning of the file.

Stream Interface – Positioning

```
#include <stdio.h>
int fseek (FILE *stream, long int offset, int
   whence);
```

- Sets the file position indicator for the stream pointed to by stream. New position is measured in bytes.

  New position = offset + whence

- If whence is set to SEEK_SET, SEEK_CUR, or SEEK_END, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively.

Positioning Streams - Example

```
int main() {
  long fsize;
  FILE *f;

  f = fopen("myfile", "r");

  /* compute the size of the file */
  fseek(f, 0, SEEK_END) ;
  fsize =  ftell(f) ;
  fprintf(stderr, "file size is: %d\n", fsize);

  fclose(f);
  return 0;
}
```

Stream Interface - Errors

```
#include<stdio.h>
void clearerr (FILE *stream);
```

- Clears the end-of-file and error indicators for the stream pointed to by stream.

```
int ferror (FILE *stream);
```

- Tests the error indicator for the stream pointed to by stream, returning non-zero if it is set. The error indicator can only be reset by the clearerr function.

Text Stream I/O Read

```
#include <stdio.h>
int fgetc (FILE *stream);
```

- Reads the next character from stream and returns it as an unsigned char cast to an int, or EOF on end of file or error.

```
char *fgets (char *s, int size, FILE *stream);
```

- Reads in at most one less than size characters from stream and stores them into the buffer pointed to by s. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A '\0' is stored after the last character in the buffer.

Text Stream I/O Read - Examples

```
#include <stdio.h>

int main() {
  FILE *f;
  int n;
  char buf[100];

  f = fopen("myfile", "r");
  fgets(buf, 10, f);
  fprintf(stderr, "read from file: %s\n", buf);
  fclose(f);

  return 0;
}
```

Text Stream I/O Write

```
#include <stdio.h>
int fputc (int c, FILE *stream);
```

- Writes the character c, cast to an unsigned char, to stream.
- Return the character written as an unsigned char cast to an int or EOF on error.

```
int fputs(const char *s, FILE *stream);
```

- Writes the string s to stream, without '\0'.
- Returns a non - negative number on success, or EOF on error.

Text Stream I/O Write - Example

```
int main() {
  FILE *f;
  int n;

  f = fopen("myfile", "w+");
  n = fputs("Let's just write something\n", f);
  fprintf(stderr, "fputs returned: %d\n", n);
  fclose(f);

  return 0;
}
```

Binary Stream I/O - Read

```
#include <stdio.h>
size_t  fread (void *ptr, size_t size, size_t
   nmemb, FILE *stream);
```

- Reads nmemb elements of data, each size bytes long, from the stream pointed to by stream, storing them at the location given by ptr.
- Returns the number of items successfully read. If an error occurs, or the end-of-file is reached, the return value is a short item count (or zero).
- Does not distinguish between end-of-file and error, use feof and ferror to determine which occurred.

Binary Stream I/O Read - Example

```
int main() {
  FILE *f;
  int n, v[3];

  f = fopen("myfile", "rb");

  /* read 3 int from file f */
  n = fread(v, sizeof(int), 3, f);

  fprintf(stderr, "fread returned: %d\n", n);
  fclose(f);

  return 0;
}
```

Binary Stream I/O - Write

```
#include <stdio.h>
size_t fwrite (const void *ptr, size_t size,
   size_t nmemb, FILE *stream);
```

- Writes nmemb elements of data, each size bytes long, to the stream pointed to by stream, obtaining them from the location given by ptr.
- Returns the number of items successfully written. If an error occurs, or the end-of-file is reached, the return value is a short item count (or zero).

Binary Stream I/O Write - Example

```
int main() {
  FILE *f;
  int n, v[3]={1, 2, 3};

  f = fopen("myfile", "wb+");

  /* writes 3 ints in file f */
  n = fwrite(v, sizeof(int), 3, f);

  fprintf(stderr, "fwrite returned: %d\n", n);
  fclose(f);

  return 0;
}
```

Example: error handling

- Upon successful completion fopen(), fdopen(), and freopen() return a FILE pointer. Otherwise, NULL is returned and the global variable errno is set to indicate the error.

- You can use perror to print the associated error message

Readings and exercises for this lecture

K&R Chapter 5.11, 6.8, 6.9

Code all the examples in the lecture.