stdio.h – Standard I/O Library

An Advanced Introduction to Unix/C Programming

John Dempsey
COMP-232 Programming Languages
California State University, Channel Islands

Library Calls

What is a library call?

Can you write C programs without using library calls?

C Library Functions

- To call a library function, an include file is necessary.
- Library calls make it easier to write programs by providing a function you can call to perform an operation.
- Library calls are defined in Section 3 of the manual pages.

/usr/include Header Files

- There are a large number of ANSI Standard Libraries defined. Function prototypes can be found in the following /usr/include files:

assert.h	signal.h
ctype.h	stdarg.h
errno.h	stddef.h
float.h	stdio.h ←
limits.h	stdlib.h
locale.h	string.h
math.h	time.h
setjmp.h	

stdio.h Library Calls

- stdio.h supports library function calls for file input and output.
- To use, add: **#include <stdio.h>**
- #include is a preprocessor statement that copies the stdio.h file into your program.
- The ‘<‘ and ‘>’ symbols specify that the stdio.h file is found in the /usr/include directory.

/usr/include/stdio.h

john@oho:~$ **ls -l /usr/include/stdio.h**
-rw-r--r-- 1 root root 29950 Apr 6 18:24 /usr/include/stdio.h

john@oho:~$ **wc -l /usr/include/stdio.h**
875 /usr/include/stdio.h

john@oho:~$ **head -3 /usr/include/stdio.h**
/* Define ISO C stdio on top of C++ iostreams.
   Copyright (C) 1991-2020 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

man Command

% **man abort**

ABORT(3) Linux Programmer's Manual ABORT(3)

NAME

abort - cause abnormal process termination

SYNOPSIS

#include <stdlib.h>

void abort(void);

DESCRIPTION

The abort() function first unblocks the SIGABRT signal, and then raises that signal for the calling process (as though raise(3) was called).

RETURN VALUE

The abort() function never returns.

Function Prototypes

- Function prototypes define the calling interface to a library function.
- Specifies the arguments to call the function.
- Specifies what is returned by the function.
- Some functions may set errno (or t_errno), which can be used to help determine why the function failed.

Function Prototype Examples

Let’s understand the syntax used for function prototypes.

void abort(void);

1. There are no parameters to the function.
2. The function does not return any values.

Function Prototype Examples

int isupper(int c);

1. Input parameter is an integer value.
2. Function returns a non-zero value if TRUE or 0 if FALSE.

Is a non-zero value always 1?

Function Prototypes Example

john@oho:~$ **cat isupper.c**

```
#include <stdio.h>
#include <ctype.h>
void main()
{
	printf("isupper('A') = %d\n", isupper('A'));
	printf("isupper('B') = %d\n", isupper('B'));
	printf("isupper('~') = %d\n", isupper('~'));
}
```

john@oho:~$ **gcc isupper.c; a.out**

```
isupper('A') = 256
isupper('B') = 256
isupper('~') = 0
```

← isupper returns 256 instead of 1.

← if not upper case character, 0 is always returned.

Function Prototype Example

char *strcpy(char *dest, const char *src);

1. **src is a constant and its value does not change after the call.**
2. **dest gets modified after the call.**
3. **Function returns a pointer to the dest string.**

Function Prototype Example

FILE *fopen(const char *pathname, const char *mode);

1. There are two constant inputs, pathname and mode, which do not change after the function is called.
2. Both are character pointers.
3. pathname can be hard coded string, e.g., “/tmp/temp.txt”, a pointer to a string, or can be an array.
4. mode can be a constant like “r” or “w+”.
5. fopen returns a file pointer, which can be used in later calls.
6. But if a NULL file pointer is returned, file is not opened and **errno** is set to indicate what error occurred.

Function Prototype Example

void *malloc(size_t size);

1. malloc allocates **size** bytes of memory. **But why use size_t instead of just int?**
2. malloc returns a void pointer but without any associated structure for the actual allocated memory.
3. So to specify the format of the allocated memory, you can use:

```
typedef enum {
   INT_TYPE,                                  ← Integer type (0)
   FLOAT_TYPE                                 ← Float type (1)
} NUM_TYPE;

typedef struct number {
   NUM_TYPE type;
   double value;
} NUMBER;                                     ← NUMBER is a typedef.

NUMBER *result = malloc(sizeof(NUMBER));      ← NUMBER defines the structure of the allocated
                                                memory pointed to by the result pointer.
```

But why use size_t instead of just int?

size_t is guaranteed to be set to the largest size the host can support.

For example, if the compiler runs on a 32-bit system, size_t can be defined as an unsigned int, i.e., 4 bytes.

But on a 64-bit system, size_t would be defined as an unsigned long long, i.e., 8 bytes.

Memory allocation functions, like calloc, malloc, and sizeof, use size_t instead of an int.

This supports more portable code when compiling on different systems.

Standard I/O Streams

There are three standard I/O streams opened when a program runs, which are:

stdin - Standard Input Stream for reading (e.g., from keyboard).
stdout - Standard Output Stream for writing to user's screen.
stderr - Standard Error Stream for reporting errors.

```
void main() {
        fclose(stdin);      ← stdin, stdout, and stderr are open by default
        fclose(stdout);     ← But some production code closes these and
        fclose(stderr);     ← writes to log files instead.
        …
```

#include <stdio.h>

- Adding **#include <stdio.h>** as one of your first lines in your C program lets you call any of the supported functions defined by stdio.h.

```
john@oho:~$ cat c.c
int main() {
    printf("Hello world.\n");
}
john@oho:~$ gcc c.c
c.c: In function ‘main’:
c.c:3:5: warning: implicit declaration of function ‘printf’
[-Wimplicit-function-declaration]
    3 |     printf("Hello world.\n");
      |     ^~~~~~
c.c:3:5: warning: incompatible implicit declaration of built-
in function ‘printf’
c.c:1:1: note: include ‘<stdio.h>’ or provide a declaration of
‘printf’
  +++ |+#include <stdio.h>
    1 | int main()
john@oho:~$ a.out
Hello world.
```

```
john@oho:~$ cat c.c
#include <stdio.h>
int main() {
    printf("Hello world.\n");
}
john@oho:~$ gcc c.c
john@oho:~$ a.out
Hello world.
```

stdio.h

stdio.h Library Functions		
Function	**Function Prototype**	**Description**
fclose	int fclose(FILE *stream);	Closes file.
feof	int feof(FILE *stream);	Checks for end of file indicator.
fflush	int fflush(FILE *stream);	Flush a file stream.
fgetc	int fgetc(FILE *stream);	Reads the next character in stream and returns unsigned char as an int or EOF on end of file.
fgetpos	int fgetpos(FILE *, fpos_t *);	Gets current file position.
fgets	char *fgets(char *s, int size, FILE *stream);	Reads in at most one less than size characters from stream. Stores them into the buffer pointed to by s.
fopen	FILE *fopen(const char *pathname, const char *mode);	Opens file named in pathname and assigns stream.
fprintf	int fprintf(FILE *stream, const char *format, …);	Write formatted output.
fputc	int fputc(int c, FILE *stream);	Write character c as unsigned char to stream.
fputs	int fputs(const char *s, FILE *stream);	Writes string s to stream without null byte \0.

stdio.h

stdio.h Library Functions		
Function	**Function Prototype**	**Description**
fread	size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);	Reads nmemb items of data, each size bytes long, from stream. Stores at location pointed to by ptr.
freopen	FILE *freopen(const char *pathname, const char *mode, FILE *stream);	Opens pathname and associates stream pointed to by stream. Original stream is closed, if it exists.
fscanf	int fscanf(FILE *stream, const char *format, …);	Reads input from the stream pointer stream.
fseek	int fseek(FILE *stream, long offset, int whence);	Sets file position for stream pointed to by stream.
fsetpos	int fsetpos(FILE *stream, const fpos_t *pos);	Sets file position for stream pointed to by stream.
ftell	long ftell(FILE *stream);	Returns file position in stream pointed to by stream.
fwrite	size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);	Write nmemb items of data, each size bytes long, to stream, data pointed to by ptr.
getc	int getc(FILE *stream);	Reads next character from stream.
getchar	int getchar(void);	Equivalent to getc(stdin).

stdio.h

stdio.h Library Functions		
Function	**Function Prototype**	**Description**
gets	char *gets(char *s);	Get a string from standard input. (Deprecated)
printf	printf(const char *format, …);	Prints formatted output to stdout.
putc	int putc(int c, FILE *stream);	Writes character to stream.
putchar	int putchar(int c);	Equivalent to putc(c, stdout).
puts	int puts(const char *s);	Writes string s and newline character to stdout.
remove	int remove(const char *pathname);	Deletes named file or directory from filesystem.
rewind	void rewind(FILE *stream);	Sets file position to the beginning of file.
scanf	int scanf(const char *format, …);	Reads input from stdin.

stdio.h

stdio.h Library Functions		
Function	**Function Prototype**	**Description**
sprintf	int sprintf(char *str, const char *format, …);	Write to the character string str.
srand	int srand(void);	Pseudo-random number generator.
sscanf	int sscanf(const char *str, const char *format, …);	Reads input from character string pointed to by str.
tmpfile	FILE *tmpfile(void);	Create a temporary file in /tmp with r/w. File deleted when closed or program terminates.
ungetc	int ungetc(int c, FILE *stream);	Pushes c back to stream. Available for future reads.

errno.h – fopen RETURN VALUES

NAME

fopen - open a stream

SYNOPSIS

#include <stdio.h>

FILE *fopen(const char *filename, const char *mode);

DESCRIPTION

The fopen() function opens the file whose pathname is the string pointed to by filename, and associates a stream with it.

RETURN VALUES

Upon successful completion, fopen() returns a pointer to the object controlling the stream. **Otherwise, a null pointer is returned and errno is set to indicate the error.**

errno Example

```
// Open file for writing.
      if ((fdout = fopen(filename_out, "w")) == NULL) {
        switch (errno) {
          case EACCES:
              printf("   ERROR: EACCES seen. Check file permissions to file %s.\n", filename_out);
              break;
          case EINTR:
              printf("   ERROR: EINTR seen. A signal was caught during execution of fopen().\n");
              break;
          case ENOENT:
              printf("   ERROR: ENOENT seen. The file %s does not exist.\n", filename_out);
              break;
        }
        printf("icopy will now exit.\n");
        exit(1);
      }
```

errno.h

ERRORS

The fopen() function will fail if:

- **EACCES** Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by mode are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.
- **EINTR** A signal was caught during the execution of fopen().
- **EISDIR** The named file is a directory and mode requires write access.
- **ELOOP** Too many symbolic links were encountered in resolving path.
- **EMFILE** There are {OPEN_MAX} file descriptors currently open in the calling process.
- **ENAMETOOLONG** The length of the filename exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
- **ENFILE** The maximum allowable number of files is currently open in the system.
- **ENOENT** A component of filename does not name an existing file or filename is an empty string.

errno.h – Checking errno

```
if ((fd=fopen(myfile,"a+")) == NULL) {
    printf("----> ERROR: Cannot open file %s.\n", myfile);
    switch (errno) {
            case EACCES:        fprintf(stderr, "----> ERROR: errno returns EACCES.\n"); break;
            case EINTR:         fprintf(stderr, "----> ERROR: errno returns EINTR.\n"); break;
            case EISDIR:        fprintf(stderr, "----> ERROR: errno returns EISDIR.\n"); break;
            case ELOOP:         fprintf(stderr, "----> ERROR: errno returns ELOOP.\n"); break;
            case EMFILE:        fprintf(stderr, "----> ERROR: errno returns EMFILE.\n"); break;
            case ENAMETOOLONG:  fprintf(stderr, "----> ERROR: errno returns ENAMETOOLONG.\n"); break;
            case ENFILE:        fprintf(stderr, "----> ERROR: errno returns ENFILE.\n"); break;
            case ENOENT:        fprintf(stderr, "----> ERROR: errno returns ENOENT.\n"); break;
            case ENOSPC:        fprintf(stderr, "----> ERROR: errno returns ENOSPC.\n"); break;
            default:            fprintf(stderr, "----> ERROR: errno returns undefined value.\n"); break;
    }
    printf("Program will now exit.\n");
    exit(1);
}
```

fseek

```
john@oho:~/VR$ man fseek | cat
FSEEK(3)                    Linux Programmer's Manual                    FSEEK(3)

NAME
       fgetpos, fseek, fsetpos, ftell, rewind - reposition a stream

SYNOPSIS
       #include <stdio.h>
       int fseek(FILE *stream, long offset, int whence);
       long ftell(FILE *stream);
       void rewind(FILE *stream);
       int fgetpos(FILE *stream, fpos_t *pos);
       int fsetpos(FILE *stream, const fpos_t *pos);

DESCRIPTION
       The  fseek()  function  sets  the file position indicator for the stream pointed to by stream.  The new position, measured in
       bytes, is obtained by adding offset bytes to the position specified by 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.  A
       successful call to the fseek() function clears the end-of-file indicator for  the  stream  and  undoes  any  effects  of  the
       ungetc(3) function on the same stream.

       The ftell() function obtains the current value of the file position indicator for the stream pointed to by stream.

       The  rewind() function sets the file position indicator for the stream pointed to by stream to the beginning of the file.  It
       is equivalent to:   (void) fseek(stream, 0L, SEEK_SET)
```

stdio.h – Using fputs, fgets, and fgetc

```
#include <stdio.h>
int main()
{
      FILE   *fd;
      char    string[50];

      if ((fd = fopen("temp.txt", "w+")) == NULL) {
        printf("ERROR: Cannot open temp.txt for write.\n");
        return 1;
      }
      fputs("ABCDEFGHIJKLMNOPQRSTUVWXYZ", fd);
      printf("In temp.txt after fputs, we're at the %ld position.\n", ftell(fd));
      fgets(string, 100, fd);
      printf("fgets read %s\n\n", string);
      rewind(fd);
      printf("In temp.txt after rewind, we're at the %ld position.\n", ftell(fd));
      fgets(string, 100, fd);
      printf("fgets read %s\n\n", string);
      rewind(fd);
      printf("In temp.txt after second rewind, we're at the %ld position.\n", ftell(fd));
      if (fgets(string, 6, fd) != NULL)
         printf("After running fgets, the first 5 characters are: %s\n\n", string);
     printf("In temp.txt after fgets, we're at the %ld position.\n", ftell(fd));
     printf("This is the next character we see: %c\n\n", fgetc(fd));
      fseek(fd, -6, SEEK_END);
      printf("In temp.txt after second fseek, we're at the %ld
                position.\n", ftell(fd));
      if (fgets(string, 6, fd) != NULL)
         printf("Last 5 characters are: %s\n", string);
      fclose(fd);
}
```

```
john@oho:~$ gcc stdio.c; a.out
In temp.txt after fprintf, we're at the 26 position.
fgets read G?                          ← Garbage is read in.

In temp.txt after rewind, we're at the 0 position.
fgets read ABCDEFGHIJKLMNOPQRSTUVWXYZ

In temp.txt after second rewind, we're at the 0 position.
After running fgets, the first 5 characters are: ABCDE

In temp.txt after fgets, we're at the 5 position.
This is the next character we see: F

In temp.txt after second fseek, we're at the 20 position.
Last 5 characters are: UVWXY
```

stdio.h – Using fscanf and fprintf

```
#include <stdio.h>
int main()
{
	char	ch;
	FILE	*fd;
	char	string[50];

	if ((fd = fopen("temp.txt", "w+")) == NULL) {
	  printf("ERROR: Cannot open temp.txt for write.\n");
	  return 1;
	}
	fprintf(fd, "%s", "ABCDEFGHIJKLMNOPQRSTUVWXYZ");
	printf("In temp.txt after fprintf, we're at the %ld position.\n", ftell(fd));
	fscanf(fd, "%s", string);	// This will read in garbage.
	printf("fscanf read %s\n\n", string);
	rewind(fd);
	printf("In temp.txt after rewind, we're at the %ld position.\n", ftell(fd));
	fscanf(fd, "%s", string);
	printf("fscanf read %s\n\n", string);
	rewind(fd);

	printf("In temp.txt after second rewind, we're at the %ld position.\n", ftell(fd));
	fscanf(fd, "%5c", string);
	string[6] = '\0';
	printf("After running fscanf, the first 5 characters are: %s\n\n", string);
	printf("In temp.txt after fscanf, we're at the %ld position.\n", ftell(fd));
	fscanf(fd, "%c", &ch);
	printf("This is the next character we see: %c\n\n", ch);
	fseek(fd, -6, SEEK_END);
	printf("In temp.txt after second fseek, we're at the %ld position.\n", ftell(fd));
	fscanf(fd, "%5c", string);
	string[6] = '\0';
	printf("Last 5 characters are: %s\n", string);
	fclose(fd);
}
```

```
john@oho:~$ gcc stdio.c; a.out
In temp.txt after fputs, we're at the 26 position.
fgets read □

In temp.txt after rewind, we're at the 0 position.
fgets read ABCDEFGHIJKLMNOPQRSTUVWXYZ

In temp.txt after second rewind, we're at the 0 position.
After running fgets, the first 5 characters are: ABCDE

In temp.txt after fgets, we're at the 5 position.
This is the next character we see: F

In temp.txt after second fseek, we're at the 20 position.
Last 5 characters are: UVWXY
```