System Programming and Error Handling

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

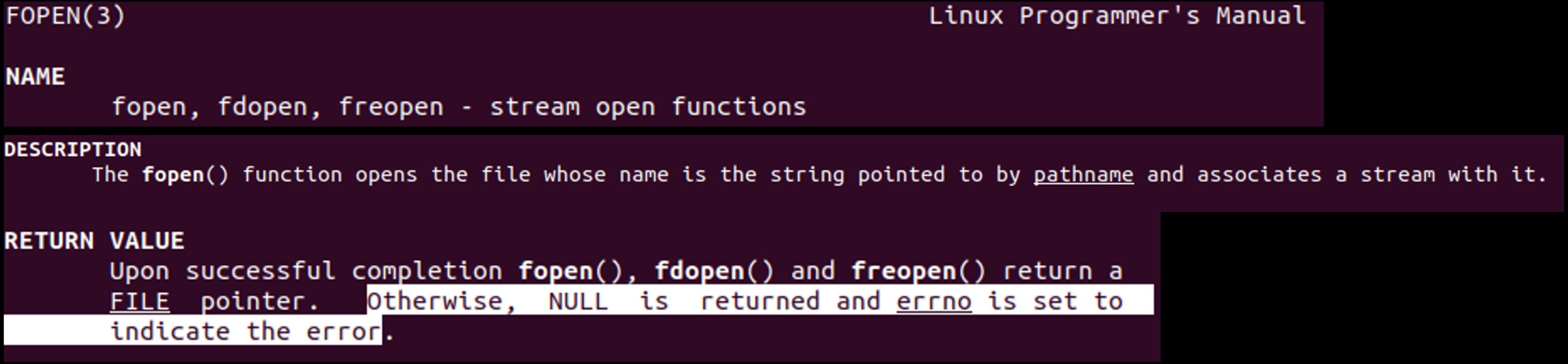
Learning objectives:

Understand errno.

Understand error handling strategies for System Programming.

Errno and error handling

- C Library mechanism for reporting errors is errno <errno.h>



```
FOPEN(3)                                        Linux Programmer's Manual

NAME
       fopen, fdopen, freopen - stream open functions
```

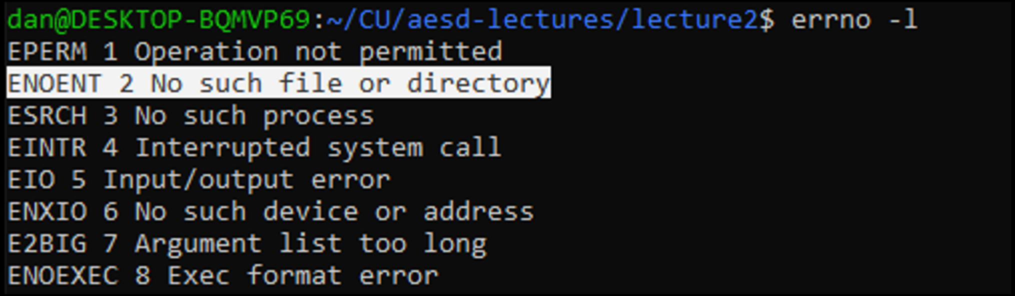
```
DESCRIPTION
       The fopen() function opens the file whose name is the string pointed to by pathname and associates a stream with it.

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

http://www.cplusplus.com/reference/cstdio/fopen/

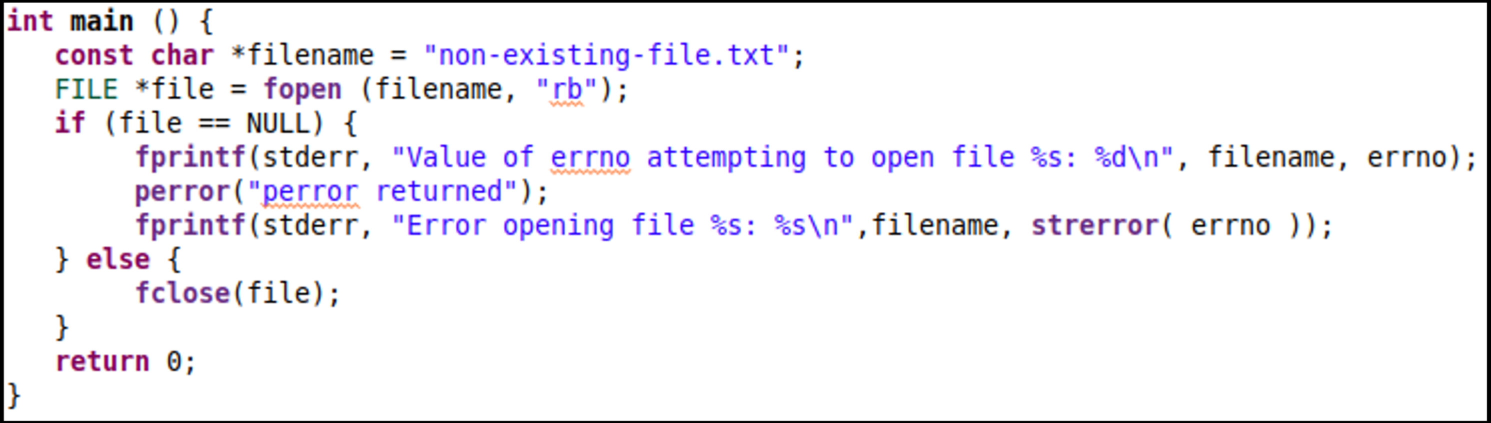
Errno and error handling

- See https://pubs.opengroup.org/onlinepubs/009695399/basedefs/errno.h.html for a list of POSIX defined errors - ENOENT means “No such file or directory”
- Use `errno -l` from the command line to dump error values and names

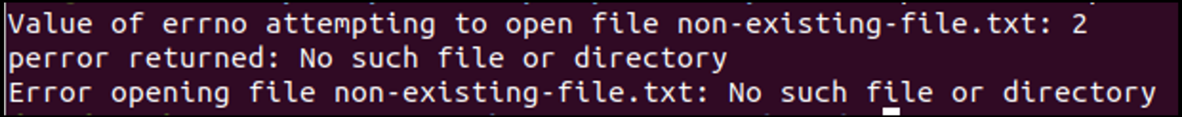


```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures/lecture2$ errno -l
EPERM 1 Operation not permitted
ENOENT 2 No such file or directory
ESRCH 3 No such process
EINTR 4 Interrupted system call
EIO 5 Input/output error
ENXIO 6 No such device or address
E2BIG 7 Argument list too long
ENOEXEC 8 Exec format error
```

Errno and error handling

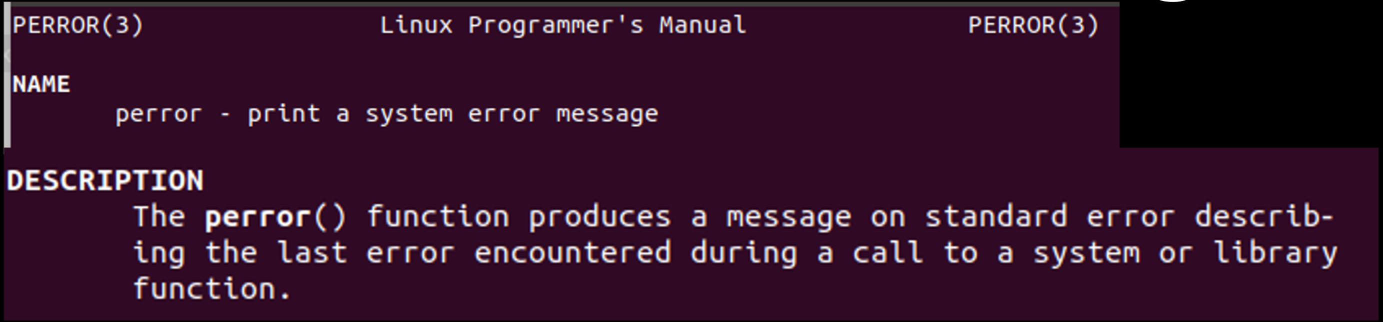


```c
int main () {
   const char *filename = "non-existing-file.txt";
   FILE *file = fopen (filename, "rb");
   if (file == NULL) {
        fprintf(stderr, "Value of errno attempting to open file %s: %d\n", filename, errno);
        perror("perror returned");
        fprintf(stderr, "Error opening file %s: %s\n",filename, strerror( errno ));
   } else {
        fclose(file);
   }
   return 0;
}
```



```
Value of errno attempting to open file non-existing-file.txt: 2
perror returned: No such file or directory
Error opening file non-existing-file.txt: No such file or directory
```

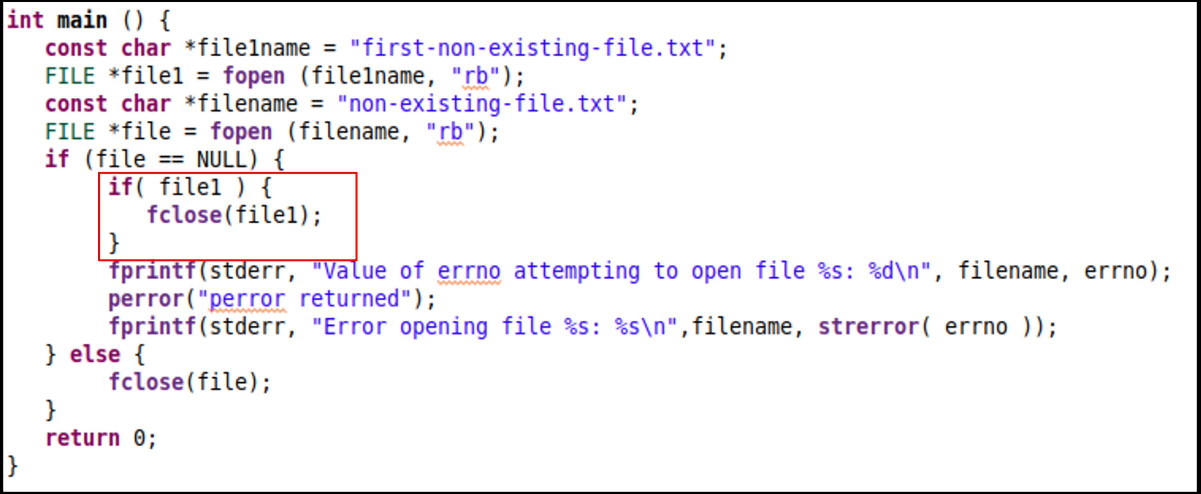
Errno and error handling



```
PERROR(3)                 Linux Programmer's Manual                 PERROR(3)

NAME
       perror - print a system error message

DESCRIPTION
       The perror() function produces a message on standard error describ-
       ing the last error encountered during a call to a system or library
       function.
```

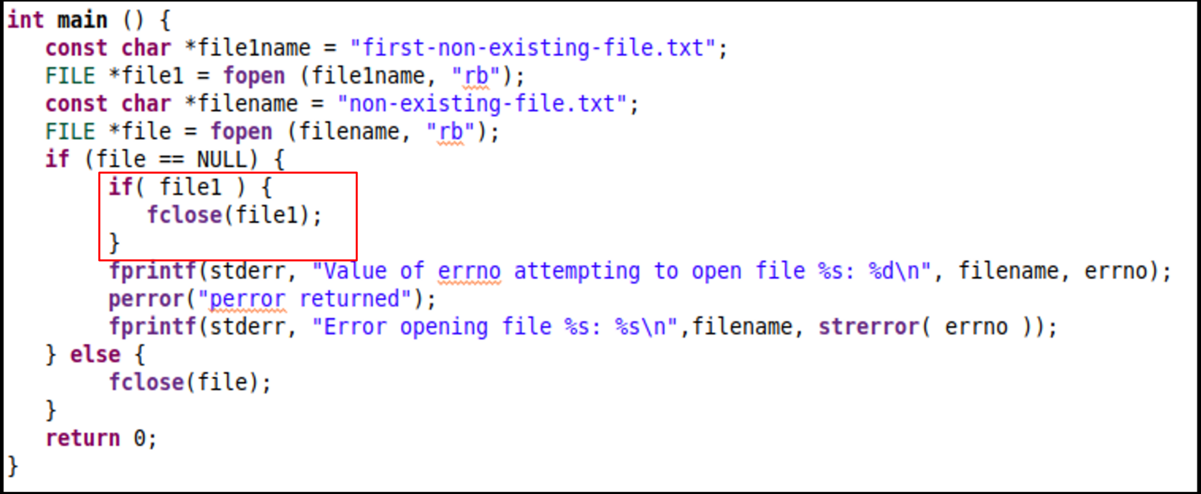


```c
int main () {
   const char *file1name = "first-non-existing-file.txt";
   FILE *file1 = fopen (file1name, "rb");
   const char *filename = "non-existing-file.txt";
   FILE *file = fopen (filename, "rb");
   if (file == NULL) {
        if( file1 ) {
           fclose(file1);
        }
        fprintf(stderr, "Value of errno attempting to open file %s: %d\n", filename, errno);
        perror("perror returned");
        fprintf(stderr, "Error opening file %s: %s\n",filename, strerror( errno ));
   } else {
        fclose(file);
   }
   return 0;
}
```

- What's wrong with this code?
    - fprintf may modify errno

Errno and error handling

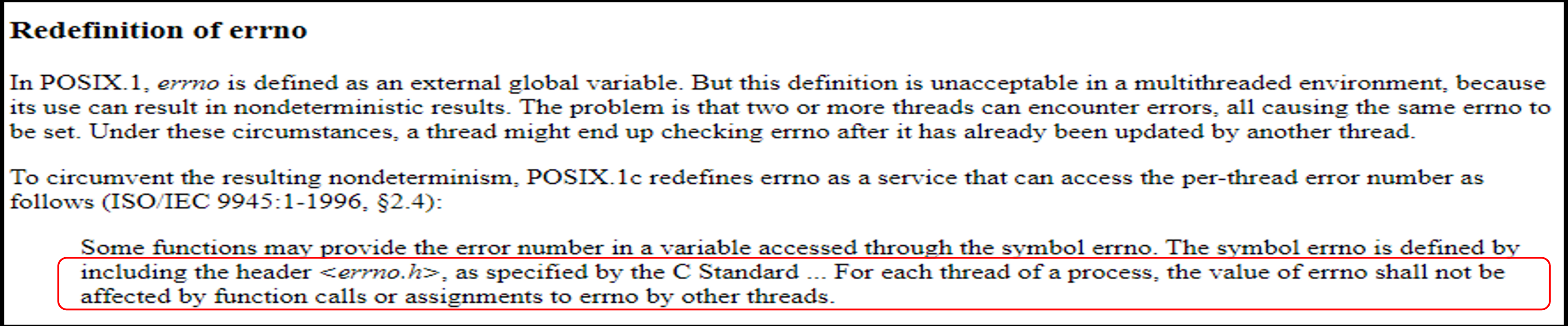
- How could you fix it?
  - Move fprintf after perror
  - Save errno to local var



```c
int main () {
   const char *file1name = "first-non-existing-file.txt";
   FILE *file1 = fopen (file1name, "rb");
   const char *filename = "non-existing-file.txt";
   FILE *file = fopen (filename, "rb");
   if (file == NULL) {
        if( file1 ) {
           fclose(file1);
        }
        fprintf(stderr, "Value of errno attempting to open file %s: %d\n", filename, errno);
        perror("perror returned");
        fprintf(stderr, "Error opening file %s: %s\n",filename, strerror( errno ));
   } else {
        fclose(file);
   }
   return 0;
}
```

Errno and error handling

- Given threads use the same memory space
- Wouldn't an error in one thread override an error in a second thread?
  - This is handled for us by POSIX



Redefinition of errno

In POSIX.1, *errno* is defined as an external global variable. But this definition is unacceptable in a multithreaded environment, because its use can result in nondeterministic results. The problem is that two or more threads can encounter errors, all causing the same errno to be set. Under these circumstances, a thread might end up checking errno after it has already been updated by another thread.

To circumvent the resulting nondeterminism, POSIX.1c redefines errno as a service that can access the per-thread error number as follows (ISO/IEC 9945:1-1996, §2.4):

> Some functions may provide the error number in a variable accessed through the symbol errno. The symbol errno is defined by including the header <*errno.h*>, as specified by the C Standard ... For each thread of a process, the value of errno shall not be affected by function calls or assignments to errno by other threads.

http://www.unix.org/whitepapers/reentrant.html