MASTERS OF COMPUTER APPLICATIONS

SEMESTER 1

PROGRAMMING & PROBLEM-SOLVING USING C

Unit 12
Error Handling

Table of Contents

1. INTRODUCTION

In the previous unit, you studied Advanced Dynamic Memory Allocation. The concept of dynamic memory allocation in C language enables the C programmer to allocate memory at runtime. Dynamic memory allocation in c language is possible by four functions of stdlib.h header file. In this unit, you will be learning about error handling in C. Systems are invariably subject to stresses that are outside the bounds of normal operation, such as those caused by

- attack
- erroneous or malicious inputs
- hardware or software faults
- unanticipated user behaviour
- unexpected environmental changes

These systems must continue to deliver essential services in a timely manner, safely and securely.

As such, C programming does not provide direct support for error handling, but being a system programming language, it provides you access at a lower level in the form of return values. Most of the C or even Unix function calls return -1 or NULL in case of any error and set an error code errno is set, which is a global variable and indicates an error occurred during any function call. You can find various error codes defined in the header file. So, a C programmer can check the returned values and take appropriate action depending on the return value. As a good practice, the developer should set errno to 0 at the time of the program's initialisation. A value of 0 indicates that there is no error in the program.

1.1. Objectives:

At studying this unit, you should be able to:

- ❖ *Recall the purpose of errno in C programming and its role in error handling.*
- ❖ *Identify the function perror and its usage for printing error messages.*
- ❖ *Recognize the function strerror and its purpose.*
- ❖ *Compare and contrast the usage of perror and strerror functions in error handling scenarios.*

2. USING ERRNO

There is no direct support for Error Handling in C language. However, using the return statement in a function, there are a few methods and variables defined in error.h header file that can be used to point out the error. A programmer has to prevent errors in the first place before testing return values from the functions. In C, the function return NULL or -1 value in case of any error. There is a global variable, errno, which sets the error code/number. Hence, while programming, the return value can be used to check errors.

What is errno?

When a function is called in C, a variable named errno is automatically assigned a code (value), which can be used to identify the type of error that has been encountered. It is a global variable indicating the error occurred during any function call and is defined in the header file errno.h.

Errno Value	Error Message
1	Operation not permitted
2	No such file or directory
3	No such process
4	Interrupted system call
5	I/O error
6	No such device or address
7	Argument list too long
8	Exec format error
9	Bad file number
10	No child processes
11	Try again
12	Out of memory
13	Permission denied

Table 1: List of a few different errno values and their corresponding meaning

1. Operation not permitted: Sometimes, while performing operations of file handling in C, we try to read from a file or change the permissions of a file by accessing it. If we don't have ownership rights or system rights to access a file, then we get this error message.
2. No such file or directory: This error message occurs whenever we try to access a file or directory that doesn't exist. It also occurs when we specify the wrong file destination path.
3. No such process: When we perform some operations that are not supported during file handling in C, it gives no such process error.
4. Interrupted System call: If we try to read the user's input, and if there is no input present, then the system calling process will not return any value and will be blocked forever. This results in interrupted system calls.
5. I/O error: I/O stands for input/output errors that occur when the system is not able to perform basic operations like reading from a file or copying data from one file to another.
6. No such device or address: When we specify an incorrect device path or address while opening or accessing them, this error occurs.
   For example: If we are trying to access a device driver using its path but in actual, it has been removed from the system.
7. Argument list too long: This error generally occurs when we work with a large number of files.
   For example, If we need to get a count of no. of files in a directory (consisting of a large number of files) that starts with the string - "Scaler", then due to limited buffer space, it will show an error message - "Argument list too long" as no. of files in that directory will be equal to the arguments list.

8. Exec format error: This error occurs when we try to execute a file that is not executable or has an invalid executable file format.
9. Bad file number: This error generally occurs when we try to write to a file which is opened for read-only purposes.
10. No child process: If a process has no further sub-process or child process, then the code returns -1 value, and we get no child process error message.

11. Try again: An attempt to create a process fails when there are no more process slots or not enough memory available. We then get a try error again.
12. Out of memory: This error occurs when there is not enough memory available to execute a process.
13. Permission denied: This error occurs when we try to read from a file that is not opened. It suggests that an attempt was made to access a file in a way that is incompatible with the file's attributes.

Example:

```
#include <stdio.h>

#include <errno.h>

int main ()

{

   FILE *fp;

   fp = fopen ("File.txt", "r");

   printf (" Value of errno: %d\n ", errno);

   return 0;

}


Output:

Value of errno: 2
```

3. USING PERROR()

If we do not keep a check on errors, then it may result in either termination of the program or it may result in giving incorrect outputs. These errors can also change the logical flow of the code. Therefore, it is very important for programmers to keep an eye on unchecked errors if they are present in the code. Below are some functional methods of error handling in C Library that are helpful while performing file operations:

- perror() function stands for print error and when called by the user, it displays a message describing about the most recent error that occured in the code. It returns the string you pass to it, followed by a colon, a space, and then the textual representation of the current errno value.
- perror() function is contained in stdio.h header file.

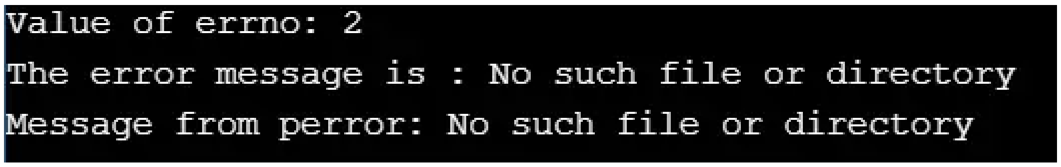
Syntax:

Void perror (const char *str) where str is a string containing a message to be printed.

Example:

```
#include <stdio.h>

#include <errno.h>

#include <string.h>

int main ()

{

   FILE *fp;

   fp = fopen (" File.txt ", "r");

   printf ("Value of errno: %d\n", errno);

   printf ("The error message is : %s\n", strerror
(errno));

   perror ("Message from perror");

   return 0;

}
```

Output:



```
Value of errno: 2
The error message is : No such file or directory
Message from perror: No such file or directory
```

4. USING STRERROR()

strerror() function is contained in a string.h header file.

It takes 'errno' as an argument and returns the string error message of the currently passed errno. It returns a pointer to the textual representation of the current errno value.

Syntax: char *strerror (int errnum), where errnum is the error number.

Here, errnum is the error number (errno value), using which a respective error message will be displayed accordingly.

C Program to illustrate the use of strerror() function: In this program with the help of strerror() function, we are displaying an error message using errno value when there is no file available naming "test.txt".

Example:

After opening the file in read only mode using file pointer fp, we check that if fp is NULL i.e. the file doesn't exists and then we pass errno as parameter in strerror() function that will print respective error message.

```
#include <errno.h>

#include <stdio.h>

#include <string.h>



int main(){

   FILE* fp;

   fp = fopen("test.txt","r");



   if(fp==NULL){        //Error handling in case if the file doesn't exist

      printf("Error: %s\n",strerror(errno));

      //errno passed as an argument to display a respective order
error message

   }

   fclose(fp);

   return 0;

}
```

Output:

Assuming "test.txt" file doesn't exists: (Output):

```
Error: No such file or directory
```

SELF-ASSESSMENT QUESTIONS - 2

5. The information that needs to be passed in when a function is called is_____.

6. The main() function doesn't return any value. (True/False)

5. SUMMARY

In C language when we call a function, a variable called errno is assigned with a numeric value and we can use that to identify the type of error if encountered while writing the code. When we perform operations of file handling in C, there are some common errors that we may encounter such as reading a file that doesn't even exists or using a file that has not been opened. To avoid errors while performing file operations, we have some helpful functional methods for error handling in C language: perror(), strerror(), feof(), and Exit status.

6. TERMINAL QUESTIONS

1. How does dynamic memory allocation in C differ from static memory allocation, and what advantages does it offer to C programmers?
2. What strategies can C programmers employ to effectively handle errors in their programs, considering the language's lack of built-in error-handling mechanisms?
3. How does the perror() function aid in error handling within C programs, and what information does it provide to the programmer?
4. Explain the significance of using the strerror() function in C error handling. Provide an example scenario where strerror() would be beneficial.

7. ANSWERS TO SELF ASSESSMENT QUESTIONS

1. Preprocessor directive
2. False
3. True
4. Text
5. False
6. False
7. remove
8. unlink.
9. #if

8. ANSWERS TO TERMINAL QUESTIONS

1. Dynamic memory allocation in C allows memory to be allocated at runtime, offering flexibility and efficient memory usage compared to static allocation. It enables C programmers to adapt to varying memory needs during program execution.
2. C programmers can utilise return values from functions and the global variable errno to detect and respond to errors. By checking return values for -1 or NULL and referencing errno for specific error codes, programmers can implement error-handling routines tailored to their applications.

3. The perror() function prints an error message describing the most recent error that occurred, along with the textual representation of the current errno value.
4. The strerror() function returns the string error message corresponding to the given errno value. For example, when attempting to open a non-existent file, strerror() could provide the error message "No such file or directory," aiding in debugging and error reporting.