


Error Handling

Issues and Ideas



Netprog: Error Handling 1

System Calls and Errors

- In general, systems calls return a negative number to indicate an error.
 - We often want to find out what error.
 - Servers generally add this information to a log.
 - Clients generally provide some information to the user.

Netprog: Error Handling 2

`extern int errno;`

- Whenever an error occurs, system calls set the value of the global variable **`errno`**.
 - You can check `errno` for specific errors.
 - You can use support functions to print out or log an ASCII text error message.

Netprog: Error Handling 3

When is `errno` valid?

- `errno` is valid only after a system call has returned an error.
 - System calls don't *clear* `errno` on success.
 - If you make another system call you may lose the previous value of `errno`.
 - `printf` makes a call to `write!`

Netprog: Error Handling

4

Error codes

```
#include <errno.h>
```

Error codes are defined in `errno.h`

<code>EAGAIN</code>	<code>EBADF</code>	<code>EACCESS</code>
<code>EBUSY</code>	<code>EINTR</code>	<code>EINVAL</code>
<code>EIO</code>	<code>ENODEV</code>	<code>EPIPE</code>
...		

Netprog: Error Handling

5

Support Routines

```
void perror(const char *string);
```

↑
In `stdio.h`

```
char *strerror(int errnum);
```

↑
In `string.h`

Netprog: Error Handling

6

General Strategies

- Include code to check for errors after every system call.
- Develop "wrapper functions" that do the checking for you.
- Develop layers of functions, each hides some of the error-handling details.

Netprog: Error Handling

7

Example wrapper

```
int Socket( int f,int t,int p) {  
    int n;  
    if ( (n=socket(f,t,p)) < 0 ) {  
        perror("Fatal Error");  
        exit(1);  
    }  
    return(n);  
}
```

Netprog: Error Handling

8

What is fatal?



- How do you know what should be a fatal error (program exits)?
 - Common sense.
 - If the program can continue – it should.
- Example – if a server can't create a socket, or can't bind to it's port - there is no sense continuing...

Netprog: Error Handling

9

Wrappers are great!

- Wrappers like those used in the text can make code much more readable.
- There are always situations in which you cannot use the wrappers
 - Sometimes system calls are "interrupted" (EINTR) – this is not always a fatal error !

Netprog: Error Handling

10

Word of Caution

- If you use the code from the book for your projects, you must understand it!
- The library of code used in the text is extensive:
 - Wrappers call custom error handling code.
 - Custom error handling code make assumptions about having other custom library functions.
 - ...

Netprog: Error Handling

11

Another approach

- Instead of simple wrapper functions, you might develop a *layered system*.
- The idea is to "hide" the `sockaddr` and error handling details behind a few custom functions:

```
int tcp_client(char *server, int port);
int tcp_server(int port);
```

Netprog: Error Handling

12

Layers and Code Re-use

- Developing general functions that might be re-used in other programs is obviously "a good thing".
- Layering is beneficial even if the code is not intended to be re-used:
 - hide error-handling from "high-level" code.
 - hide other details.
 - often makes debugging easier.

Netprog: Error Handling

13

The Best Approach to handling errors



- There is no *best approach*.
- Do what works for you.
- Make sure you check *all* system calls for errors!!!!
 - Not checking can lead to security problems!
 - Not checking can lead to bad grades on homework projects!

Netprog: Error Handling

14
