CSE333 SECTION 6

GNU C Library

- “de facto” standard C library
- Contains a bunch of header files and APIs to do various tasks

- Don’t need to memorize everything
- Do need to know what if there’s an API that can do X

- Source available at: http://www.gnu.org/software/libc/download.html

Error Reporting

- Most library functions return a special value to indicate that they have failed.
  - typically -1, a null pointer, or a constant such as EOF that is defined for that purpose.
- To find out what kind of error it was, you need to look at the error code stored in the variable errno

Errno

Variable: *volatile int* **errno**

- Contains the system error number. You can change the value of errno.
- Initially set to zero at program startup is zero
- Many library functions are guaranteed to set it to certain nonzero values when they encounter certain kinds of errors
- Not changed when library function succeed
  - the value of errno after a successful call is not necessarily zero,

Error Codes

- Macro: *int* **EPERM**
- Operation not permitted

- Macro: *int* **ENOENT**
- No such file or directory

- Macro: *int* **ESRCH**
- No process matches the specified process ID.

- Macro: *int* **EINTR**
- Interrupted function call

- ……..

Error Messages

- Function: *char* * **strerror** *(int errnum)*
- maps the error code specified by the errnum argument to a descriptive error message string. The return value is a pointer to this string.

- Function: *void* **perror** *(const char *message)*
- Prints an error message to the stream stderr
- If *message* is either a null pointer or an empty string, perror just prints the error message corresponding to errno

Program Arguments

- The system starts a C program by calling the function main

int main (int argc, char *argv[])

- Argc: number of command line arguments
- Argv: a vector of C strings; its elements are the individual command line argument strings.
- The file name of the program being run is also included in the vector as the first element
- A null pointer always follows the last element: argv[argc] is this null pointer.

Parsing Program Arguments

Function: int getopt (int argc, char *const *argv, const char *options)

- Gets the next option argument from the argument list specified by the argv and argc arguments.
- Options: a string that specifies the option characters that are valid for this program.

Return value:

- The option character for the next command line option. Sets optarg if the option has an argument
- -1, when no more option arguments are available
- '?' for unknown option character or a missing option argument. Sets the external variable optopt to the actual option character.

Getopt Example

```
% testopt
aflag = 0, bflag = 0, cvalue = (null)

% testopt -a -b
aflag = 1, bflag = 1, cvalue = (null)

% testopt -ab
aflag = 1, bflag = 1, cvalue = (null)

% testopt -c foo
aflag = 0, bflag = 0, cvalue = foo

% testopt arg1
aflag = 0, bflag = 0, cvalue = (null)
Non-option argument arg1
```

```
while ((c = getopt (argc, argv, "abc:")) != -1)
  switch (c)
    {
    case 'a':
      aflag = 1;
      break;
    case 'b':
      bflag = 1;
      break;
    case 'c':
      cvalue = optarg;
      break;
    case '?':
      if (optopt == 'c')
        fprintf (stderr, "Option -%c requires an
argument.\n", optopt);
      else if (isprint (optopt))
        fprintf (stderr, "Unknown option `-%c'.\n",
optopt);
      else
        fprintf (stderr,
                 "Unknown option character `\\x%x'.\n",
                 optopt);
      return 1;
    default:
      abort ();
    }
```

Environment Variables

- When a program is executed, it receives information about the context in which it was invoked in two ways.
  - Program arguments: pass command-line arguments specific to the particular program being invoked
  - Environment variables: information that is shared by many programs, changes infrequently, and that is less frequently used

$export NAME=VALUE

$echo $NAME

- Programs executed from the shell inherit all of the environment variables from the shell.

Environment Variables

Standard environment variables include:

- HOME: user's home directory, or initial default working directory.
- LOGNAME: name that the user used to log
- PATH: a sequence of directory names which is used for searching for a file
- TERM: specifies the kind of terminal that is receiving program
- TZ: specifies the time zone

Environment Access

- The value of an environment variable can be accessed with the getenv function

Function: char * getenv (const char *name)

- This function returns a string that is the value of the environment variable name

Example