Library Functions

Standard Libraries

- Any system call is *not* part of the C language definition
- Such system calls are defined in *libraries*, identified with the suffix **.a**
- Libraries typically contain many **.o** object files
- To create your own library archive file:
  ar crv mylib.a *.o
- Disregard "**ranlib**" command in Wang, p311 (no longer needed)
- Look in **/usr/lib** and **/usr/local/lib** for most system libraries
- Can list all .o files in an archive use "**ar t /usr/lib/lib.a**"
- More useful to see all the function names:
  /usr/ccs/bin/nm /usr/lib/libc.a | grep FUNC

Standard Libraries (cont)

- By default, gcc links **/usr/lib/libc.a** to all executables
- Typing "**man 3 intro**" will give a list of most of the standard library functions
- Any other libraries must be explicitly linked by referring to the absolute pathname of the library, or preferably by using the "**-l**" gcc switch:
  gcc *.o /usr/lib/libm.a -o mathExamples
  gcc *.o -lm -o mathExamples
- These **.a** files are also sometimes referred to as *static libraries*
- Often you will find for each system **.a** file a corresponding **.so** file, referred to as a *shared object* (not needed for this course)
- Advantage of shared objects: smaller executable files (library functions loaded at run time)

Standard Libraries: Example

```
#include <stdio.h>
/* #include <math.h> */
int main( void )
{
   printf( "Square root of 2 is %f\n", sqrt(2) );
   return( 0 );
}
```

- May get various problems/errors when you compile with:
  1) **gcc example.c -o example**
  2) **gcc example.c -m -o example**
  3) **gcc example.c -m -o example** # with math.h included

Files and Directories

- Disk drives divided into partitions
- Each partition contains a filesystem (type **df** for a listing of filesystems *mounted* on any given computer)
- Filesystems are mounted onto existing filenames (Fig 8.4, p.241)
- Each filesystem has a boot block, a super block, an *ilist* containing ***inodes*** (short for index nodes), directory blocks, and data blocks
- An inode contains all the information about a file: type, time of last modification/write/access, uid/gid of creator, size, permissions, etc.
- Directories are just lists of inodes (2 files automatically created with mkdir: "**.**" (inode of directory) and "**..**" (inode of parent directory)
- See figure 8.3 (page 240) for an example.

Example: argc/argv

```
#include <stdio.h>
#include <sys/stat.h>
int main( int argc, char *argv[])
{
   if( argc == 2 )
   {
      struct stat buf;
      if( stat( argv[1], &buf ) != -1 )
         printf( "file %s has size %d\n", argv[1],
                                            buf.st_size );
   }
   return( 0 );
}
```

Miscellaneous

- **fopen**/**fread**/**fwrite**/**fclose**, etc. are implemented in terms of low-level *non*-standard i/o functions **open**/**read**/**write**/**close**, etc.
- There are 3 types of buffering:
  - fully buffered (or *block buffered*):
    - actual physical i/o takes place only when buffer is filled
  - line buffered:
    - actual i/o takes place when a newline (**\n**) is encountered
  - unbuffered:
    - output as soon as possible
- All files are normally block buffered, except *stdout* (line buffered only if it refers to a terminal), and *stderr* (always unbuffered)
- Can use **fflush()** to force a buffer to be cleared

S-125

Advanced Library Functions

S-126

String/Character Handling

- All "str" functions require input strings be terminated with a null byte
- Some of the most common ones:
  strlen, **strcpy**, **strcmp**, **strcat**
- **strtok** used for extracting "tokens" from strings
- **memcpy** not just for strings!
- **strncmp** allows limits to be placed on length of strings, other 'n' string functions
- Some function for testing/converting single characters:
  isalpha, **isdigit**, **isspace**
  toupper, **tolower**
  atoi, atol

S-127

Storage Allocation

- Dynamic memory allocation (very important for many C programs):
  malloc, **calloc**, **free**, **realloc**
- An (incomplete) example:

```
#include <stdio.h>
#include <stdlib.h>
struct xx *sp;
sp = (struct xx *) malloc( 5 * sizeof(struct xx) );
if( sp == (struct xx *) NULL )
{
   fprintf( stderr, "out of storage\n" );
   exit( -1 );
}
```

S-128

Date and Time Functions

- **clock_t, clock(), time_t, time()**
- Most UNIX time functions have evolved from various sources, and are sometimes inconsistent, referring to time as one of:
  - the number of seconds since Jan 1, 1970 (or Jan 1, 1900)
  - the number of clock ticks since Jan 1, 1970 (or Jan 1, 1900)
  - the broken down structure "**struct tm**" (see **/usr/include/time.h**)
  - the broken down structure "**struct timeval**" (see **/usr/include/sys/time.h**)
- Some are intended for time/date, whereas others are intended for measuring elapsed time

S-129

Variable Arguments

- An under-used but very powerful feature
- **printf()** is an example where the number and types of arguments can differ from invocation to invocation
- **/usr/include/stdarg.h** provides definitions of:
  - a special type named **va_list**
  - three macros to implement variable arguments:
    - **va_start**
    - **va_end**
    - **va_arg**
- Another useful function is "**vfprintf**", as shown in the next slide

S-130

Variable Arguments

- A very useful example:

```
#include <stdarg.h>
void Abort( char *fmt, ... )
{
   va_list args;
   va_start( args, fmt );
   fprintf( stderr, "\n\t" );
   vfprintf( stderr, fmt, args );
   fprintf( stderr, "\n\n" );
   va_end( args );
   exit( -1 );
}
```

S-131

Environment Interfacing

- Reading environment variables:

```
getenv( “PATH” );
```

- Executing a “**$SHELL**” shell command:

```
fflush( stdout );
system( “ls -atl” );
```

- Can also execute a system call and have its output sent to a pipe instead of stdout: (w*e’ll talk more about pipes in chapter 12)*

```
FILE *pipe;
pipe = popen( “ls -atl”, “r” );
...
pclose( pipe );
```

S-132