CS240: Programming in C

Lecture 15: Unix interface: low-level interface

PURDUE UNIVERSITY

Streams Recap

- Higher-level interface, layered on top of the primitive file descriptor facilities.
- More powerful set of functions for performing actual input and output operations than the corresponding facilities for file descriptors.
- It is implemented in terms of file descriptors
  - the file descriptor can be extracted from a stream and then perform low-level operations directly on the file descriptor
  - a file can be open as a file descriptor and then make a stream associated with that file descriptor.

Text Stream I/O Write

```
#include <stdio.h>
int fputc (int c, FILE *stream);
```

- Writes the character c, cast to an unsigned char, to stream.
- Return the character written as an unsigned char cast to an int or EOF on error.

```
int fputs(const char *s, FILE *stream);
```

- Writes the string s to stream, without '\0'.
- Returns a non - negative number on success, or EOF on error.

Text Stream I/O Write - Example

```
int main() {
  FILE *f;
  int n;

  f = fopen("myfile", "w+");
  n = fputs("Let's just write something\n", f);
  fprintf(stderr, "fputs returned: %d\n", n);
  fclose(f);

  return 0;
}
```

Binary Stream I/O - Write

```
#include <stdio.h>
size_t fwrite (const void *ptr, size_t size,
   size_t nmemb, FILE *stream);
```

- Writes nmemb elements of data, each size bytes long, to the stream pointed to by stream, obtaining them from the location given by ptr.
- Returns the number of items successfully written. If an error occurs, or the end-of-file is reached, the return value is a short item count (or zero).

Binary Stream I/O Write - Example

```
int main() {
  FILE *f;
  int n, v[3]={1, 2, 3};

  f = fopen("myfile", "wb+");

  /* writes 3 ints in file f */
  n = fwrite(v, sizeof(int), 3, f);

  fprintf(stderr, "fwrite returned: %d\n", n);
  fclose(f);

  return 0;
}
```

Binary Stream I/O - Read

```
#include <stdio.h>
size_t  fread (void *ptr, size_t size, size_t
   nmemb, FILE *stream);
```

- Reads nmemb elements of data, each size bytes long, from the stream pointed to by stream, storing them at the location given by ptr.
- Returns the number of items successfully read. If an error occurs, or the end-of-file is reached, the return value is a short item count (or zero).
- Does not distinguish between end-of-file and error, use feof and ferror to determine which occurred.

Binary Stream I/O Read - Example

```
int main() {
  FILE *f;
  int n, v[3];

  f = fopen("myfile", "rb");

  /* read 3 int from file f */
  n = fread(v, sizeof(int), 3, f);

  fprintf(stderr, "fwrite returned: %d\n", n);
  fclose(f);

  return 0;
}
```

File descriptor revisited

- A handle to access a file (or I/O device), like the file pointer in streams
- It is a small non-negative integer used in same open / read-write / close paradigm
- Returned by the open system call; all active opens have distinct fd's
- Once a file is closed, fd can be reused
- Same file can be opened several times, and be associated with multiple fd's

Example

```
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

int main() {
  int  f1, f2;
  int n;
  char buf[100];

  f1 = open("log1", O_RDONLY);
  f2 = open("log2", O_RDONLY);
  fprintf(stderr, "Log1 file descriptor is: %d\n", f1);
  fprintf(stderr, "Log2 file descriptor is: %d\n", f2);
  close(f1);
  close(f2);

  f2 = open("log2", O_RDONLY);
  fprintf(stderr, "Opening again log2, notice the new file descriptor: %d\n", f2);
  close(f2);

  return 0;
}
```

Standard Unix fds revisited

- fd 0 same as stdin
- fd 1 same as stdout
- fd 2 same as stderr
- Predefined, automatically created when program starts
- Can be closed

Low-level functions

- #include <unistd.h>
- int open(const char *pathname, int flags);
- int open(const char *pathname, int flags, mode_t mode);

- int creat(const char *pathname, mode_t mode);
- Flags: O_RDONLY, O_WRONLY or O_RDWR bitwise OR with O_CREAT, O_EXCL, O_TRUNC, O_APPEND, O_NONBLOCK, O_NDELAY

- int close(int fd);

FD IS an INT (file descriptor) not a FILE* !!!!!

Return values

- Note that they do not take a FILE*, but a int.
- On success they return the file descriptor
- On error, they return -1

Read/Write

- #include <unistd.h>
- ssize_t read(int fd, void *buf, size_t count);
- ssize_t write(int fd, const void *buf, size_t count);
- fd is a descriptor, _not_ FILE pointer
- Returns number of bytes transferred, or -1 on error
- Normally waits until operation is enabled (e.g., there are bytes to read), except under O_NONBLOCK and O_NDELAY (in which case, returns immediately with ‘‘try again’’ error condition)

Binary Stream I/O Write - Example

```
int main() {
  int f;
  int n, v[3]={1, 2, 3};

  f = open("myfile", O_RDWR );

  /* writes 3 ints in file f */
  n = write(f, v, sizeof(int)*3);

  fprintf(stderr, "write returned: %d\n", n);
  close(f);

  return 0;
}
```

Binary Stream I/O Read - Example

```
int main() {
  int f;
  int n, v[3];

  f = open("myfile", O_RDONLY);

  /* read 3 int from file f */
  n = read(f, v, sizeof(int)*3);

  fprintf(stderr, "read returned: %d\n", n);
  close(f);

  return 0;
}
```

Listing the files in a directory

```
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>

int main () {
   DIR *dp;
   struct dirent *ep;

   dp = opendir ("./");
   if (dp != NULL) {
         while (ep = readdir (dp))
            puts (ep->d_name);
         closedir (dp);
   }
   else
    perror ("Couldn't open the current directory");

   return 0;
}
```

Readings and exercises for this lecture

Read man/info pages for all the functions mentioned in the lecture

Code all the examples in the lecture.