CSE 333 – SECTION 2

POSIX I/O Functions

Overview

- STDIO vs. POSIX Functions
- Errors and Error codes
- UNIX System I/O calls
- Example program
- Section Exercise

STDIO vs. POSIX Functions

- User mode vs. Kernel mode.
- STDIO library functions – *fopen, fread, fwrite, fclose,* etc. with FILE* pointers.
- POSIX functions – *open, read, write, close,* etc. with integer file descriptors.
- POSIX file descriptors: Input – 0; Output – 1; Error – 2.
- FDs – index for an entry in a table with details of open files.

Why learn these functions?

- They are unbuffered. You can implement different buffering/caching strategies on top of read/write.
- There is no equivalent of fread/fwrite for network and other I/O devices.
- More explicit control since read and write functions are system calls and you can directly access system resources.

Errors

- When an error occurs, the error number is stored in “errno”, which is defined under errno.h
- View/Print details of the error using perror() and errno.
- POSIX functions have a variety of error codes to represent different errors.

System I/O calls

- Opening a file

```
#include <sys/file.h> //can be replaced by <fcntl.h>
int open(char* filename, int flags, int mode);
```

Returns an integer which is the file descriptor.
Returns -1 if there is a failure.

`filename:` A string representing the name of the file.
`flags:` An integer code describing the access.

O_RDONLY -- opens file for read only
O_WRONLY – opens file for write only
O_RDWR – opens file for reading and writing
O_APPEND --- opens the file for appending
O_CREAT -- creates the file if it does not exist

`mode`: File protection mode. Ignored if O_CREAT is not specified.

System calls continued

- Reading from a file.

```
#include <sys/types.h> // or #include <unistd.h>
size_t read(int fd, char *buffer, size_t bytes);
```

fd: file descriptor.

buffer: address of a memory area into which the data is read.

bytes: the maximum amount of data to read from the stream.

The return value is the actual amount of data read from the file.

- Writing to a file

```
size_t write(int fd, char *buffer, size_t bytes);
```

- Closing a file

```
int close(int fd);
```

Error codes for read errors

- **EBADF -** *fd* is not a valid file descriptor or is not open for reading.
- **EFAULT -** *buf* is outside your accessible address space.
- **EINTR -** The call was interrupted by a signal before any data was read.
- **EISDIR -** *fd* refers to a directory.

System calls continued

- Accessing directories.

  Header file: `#include <sys/dir.h>`

- Opening a directory.

  DIR *opendir(char* dir_name);

- Opens a directory given by dir_name and provides a pointer DIR* to access files within the directory.

System calls continued

- Reading a directory file.

```
int readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result);
```

- returns 0 on success.
- A NULL pointer is returned in *result when the end of the directory is reached.

```
struct dirent {
 u_long d_ino; /* i-node number for the dir entry */
 u_short d_reclen; /* length of this record */
 off_t d_off ; /* offset to the next dirent*/
 unsigned char d_type; /* type of file; not supported by all file system types */
 char d_name[MAXNAMLEN+1] ; /* directory entry name */
};
```

Reading N bytes from a file

```
#include <errno.h>
#include <unistd.h>
...
   char *buf = ...;
   int bytes_read = 0;
   int result = 0;
   while (bytes_read < N) {
         result = read(fd, buf + bytes_read, N - bytes_read);
         if (result == -1) {
                  if (errno != EINTR)) {
                           // a real error happened, return an error result
                  }
                  // EINTR happened, do nothing and loop back around
                  continue;
         }
         bytes_read += result;
   }
   buf[N] = '\0';
```

Section Exercise

- Find a partner if you wish.
- Write a C program that does the following
  - Given a command line argument, if it is an ordinary file, print its contents to stdout.
  - If not, or some other error occurs, print an informative error message using perror().
  - Similar to cat.
  - You must use the POSIX functions to open, close, read and write.