# Low-Level File I/O in C

# System Calls for Low-Level File I/O

## open()

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int flags, mode_t mode);
```

Used to open a file for reading, writing, appending, etc.

Returns a file descriptor (small positive integer). Returns -1 on error.

`path`: Path to file to open.

`flags`: What is to be done by the `open`. Multiple flags are specified by **bit** or'ing (`|`):

- `O_RDONLY` --- Open for reading only.
- `O_WRONLY` --- Open for writing only.
- `O_CREAT` --- Create file if it does not exist.
- `O_TRUNC` --- Truncate size to 0.
- ...

`mode`: Permissions with which to create a file. Used only when creating a file on an open for write. Multiple modes are specified by bit or'ing:

- `S_IRUSR` --- Read for owner.
- `S_IWUSR` --- Write for owner.
- `S_IRGRP` --- Read for group.

- `S_IROTH` --- Read for other.
- ...

Typical read call:

```
int rfd;

if ((rfd = open(argv[1], O_RDONLY, 0)) < 0)
   pdie("Open failed");
```

Typical write call:

```
int wfd;

if ((wfd = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC,
                S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) <
0)
   pdie("Open failed");
```

# Sample Program

```
/
************************************************************
***********
 * fileio.c
 *
 * This program demonstrates how to do low-level file I/O
in C.  This
 * program implements a simple version of Unix's cp
command.  Two
 * filename are expected on the command line.  The first
file is the
 * name of the file to be copied, the second is the file to
be created.

************************************************************
***********/
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>


/* Prototypes. */
void pdie(const char *);
void die(const char *);


#define BUFFER_SIZE 1024   /* Size of the read/write
buffer. */


/
**********************************************************
***********
 * main

**********************************************************
***********/

int main(int argc, char* argv[])
{
   int rfd;   /* Read file descriptor. */
   int wfd;   /* Write file descriptor. */
   char buffer[BUFFER_SIZE];   /* Read/Write buffer. */
   char *bp;   /* Pointer into write buffer. */
   int bufferChars;   /* Number of bytes remaining to be
written. */
   int writtenChars;  /* Number of bytes written on last
write. */

   if (argc != 3)
   {
      printf("Two filenames expected.\n");
```

```
      exit(1);
   }

   /* Open file to be copied. */
   if ((rfd = open(argv[1], O_RDONLY, 0)) < 0)
      pdie("Open failed");

   /* Open file to be created. */
   if ((wfd = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH))
< 0)
      pdie("Open failed");

   while (1)
   {
      /* Normal case --- some number of bytes read. */
      if ((bufferChars = read(rfd, buffer, BUFFER_SIZE)) >
0)
      {
         bp = buffer;   /* Pointer to next byte to write.
*/

         /*
            Since we can't guarantee that all the bytes
will be written
            in a single write(), this code must be written
such that
            several write()'s can possibly be called.
         */
         while (bufferChars > 0)
         {
            if ((writtenChars = write(wfd, bp,
bufferChars)) < 0)
               pdie("Write failed");

            bufferChars -= writtenChars;   /* Update. */
            bp += writtenChars;
         }
      }
      else if (bufferChars == 0)   /* EOF reached. */
         break;
```

```
      else   /* bufferChars < 0 --- read failure. */
         pdie("Read failed");
   }

   close(rfd);
   close(wfd);

   return 0;
}


/
************************************************************
***********
 * pdie --- Print error message, call perror, and die.

************************************************************
***********/

void pdie(const char *mesg) {

   perror(mesg);
   exit(1);
}


/
************************************************************
***********
 * die --- Print error message and die.

************************************************************
***********/

void die(const char *mesg) {

   fputs(mesg, stderr);
   fputc('\n', stderr);
   exit(1);
}
```