File input/output
Software Technologies - Lecture 2

Saurabh Barjatiya

International Institute Of Information Technology, Hyderabad

28 December, 2009

Contents

Introduction

Buffered input/output is very useful when programs input/output block size is very small in comparison to Operating Systems block size. Normally operating systems would read / write data in blocks (say 4KB). In case the programs reads or writes only one record (about 100 bytes) at a time, then about 40 system calls are made to read / write 4KB of data, which could have been done in one system call. In such cases Buffered I/O is very useful to avoid penalty incurred in context switch which happens when we invoke system calls.

Also functions used in buffered input/output are very similar to common functions used in console input/output making them very easy and intuitive to learn and use.

Common functions

fopen

Header file: stdio.h

Declaration:

```
FILE *fopen(const char *path, const char *mode);
```

Here, path can be normal C-style null terminated string containing either relative or absolute path and mode can be one of ‘r’, ‘r+’, ‘w’, ‘w+’, ‘a’ and ‘a+’.

Common functions

fprintf

Header file: stdio.h

Declaration:

```
int fprintf(FILE *stream, const char *format, ...);
```

Here 'stream' is FILE pointer returned by call to fopen function. We can also use the default streams stdout and stderr apart from streams that we open. Format specifier and arguments are exactly same as normal 'printf' function.

Common functions

fscanf

Header file: stdio.h

Declaration:

```
int fscanf(FILE *stream, const char *format, ...);
```

Here 'stream' is FILE pointer returned by call to fopen function. We can also use the default stream stdin apart from streams that we open. Format specifier and arguments are exactly same as normal 'scanf' function.

Common functions

fclose

Header file: stdio.h

Declaration:

```
int fclose(FILE *fp);
```

Here 'fp' is FILE pointer returned by call to fopen function. It is important to close files opened for writing so that file buffers which are not yet flushed / written gets written before the program ends.

Common functions

fflush

Header file: stdio.h

Declaration:

```
int fflush(FILE *stream);
```

Here 'stream' is FILE pointer returned by call to fopen function. This function is important if we want to ensure that all user-space buffers are synced to kernel buffers for the output stream. This helps in avoiding data-loss when using buffered I/O and program terminates unexpectedly (due to signals).

Common functions

fseek

Header file: stdio.h

Declaration:

```
int fseek(FILE *stream, long offset, int whence);
```

Here 'stream' is FILE pointer returned by call to fopen function. Here offset is long value and hence this function is useful only when we are dealing with small files (size ¡ 2 GB). For files greater than 2 GB we have to use 64-bit versions of these functions.

'whence' can be 'SEEK_SET', 'SEEK_CUR' or 'SEEK_END' to indicate that offset is with respect to start of file, current position in file or end of file.

Contents

Introduction

Un-buffered input/output is very useful when programs input/output block size is almost equal or larger in comparison to Operating Systems block size. By using unbuffered input/ouput we avoid one more layer of buffering which is not useful when amount of data to be transferred is large.

Un-buffered input/output is very useful when progrmas are piped and output of one program is used as input in other program. In such cases buffering may cause delays or even halt the programs breaking the entire functionality. These functions are also useful in binary input/output.

Common functions

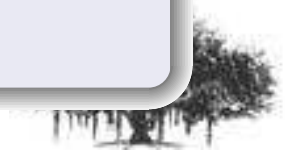
open

Header file: sys/types.h, sys/stat.h, fcntl.h

Declaration:

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

'open' function call on successful completion should return a non-negative file descriptor to file. 'flags' must contain one of the access modes 'O_RDONLY', 'O_WRONLY', or 'O_RDWR'. Optionally 'flags' can also contain creation flags like 'O_CREAT', 'O_EXCL', 'O_NOCTTY', and 'O_TRUNC'. 'mode' can be used to set permissions on newly created files.



Common functions

read

Header file: unistd.h

Declaration:

```
ssize_t read(int fd, void *buf, size_t count);
```

'read' function call on successful completion returns number of bytes read from given file descriptor. The number of bytes can be zero or at most equal to count. The bytes read are stored in array buf and hence buf must be of at least count bytes long. The buf array is not null terminated, so if it ASCII input/output then programmer must null terminate buffer before using it as normal C style string.

Common functions

write

Header file: unistd.h

Declaration:

```
ssize_t write(int fd, const void *buf, size_t count);
```

'write' function call on successful completion returns number of bytes written into file descriptor. The bytes to write are read from 'buf' and at most count bytes are written. Programmer must ensure that 'buf' is at least count bytes long to avoid writing of junk data into file-descriptor and to avoid segmentation faults.

Common functions

close

Header file: unistd.h

Declaration:

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

'close' function calls simply closes the stream associated with given file descriptor.

Common functions

lseek

Header file: sys/types.h, unistd.h

Declaration:

```
off_t lseek(int fildes, off_t offset, int whence);
```

'lseek' function calls helps in jumping to specified location in stream associated with given file descriptor. Here 'fildes' is file descriptor of stream in which we want to seek location, 'offset' is offset with respect to 'whence' and 'whence' is one of 'SEEK_SET', 'SEEK_CUR' and 'SEEK_END'.