Micro Assignment - 04

Universal I/O in UNIX

Systems calls that make I/O refer to open files using file descriptors (small positive integers). File descriptors provide a unifying abstraction for I/O in UNIX like operating systems. They are used for regular files, sockets, pipes, FIFOs, terminals, and devices. All UNIX processes by default have access to three special file descriptors.

File descriptor	**Purpose**	**POSIX name**	***stdio* stream**
0	standard input	`STDIN_FILENO`	*stdin*
1	standard output	`STDOUT_FILENO`	*stdout*
2	standard error	`STDERR_FILENO`	*stderr*

To create or open a file use the following system call.

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

int open(const char *pathname, int flags, ... /* mode_t mode */);
                        Returns file descriptor on success, or −1 on error
```

Figure shows an example code fragment that uses the open() system for opening files under different situations.

On success open() returns the lowest unused index from the file descriptor table.

```
/* Open existing file for reading */

fd = open("startup", O_RDONLY);
if (fd == -1)
    errExit("open");

/* Open new or existing file for reading and writing, truncating to zero
   bytes; file permissions read+write for owner, nothing for all others */

fd = open("myfile", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
if (fd == -1)
    errExit("open");

/* Open new or existing file for writing; writes should always
   append to end of file */

fd = open("w.log", O_WRONLY | O_CREAT | O_TRUNC | O_APPEND,
                   S_IRUSR | S_IWUSR);
if (fd == -1)
    errExit("open");
```

Figure 1: Code fragment showing the use of open().

Reading From A File

The *read()* system call reads data from the open file referred to by the descriptor *fd*.

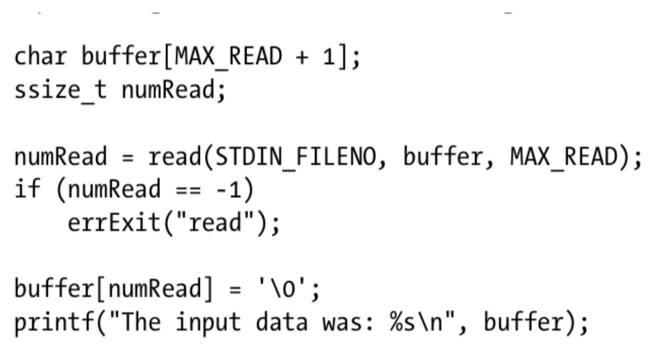
```
#include <unistd.h>

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

Returns number of bytes read, 0 on EOF, or –1 on error

The *count* argument specifies the maximum number of bytes to read. (The *size_t* data type is an unsigned integer type.) The *buffer* argument supplies the address of the memory buffer into which the input data is to be placed. This buffer must be at least *count* bytes long.

A call to read() can return with less than the requested number of characters because the reading location was already close to the end of the file.

Figure shows an example usage of the read() system call.



```
char buffer[MAX_READ + 1];
ssize_t numRead;

numRead = read(STDIN_FILENO, buffer, MAX_READ);
if (numRead == -1)
    errExit("read");

buffer[numRead] = '\0';
printf("The input data was: %s\n", buffer);
```

Figure 2: Example usage of the read() system call.

Writing To A File

The *write()* system call writes data to an open file.

```
#include <unistd.h>

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

Returns number of bytes written, or –1 on error

The arguments to *write()* are similar to those for *read()*: *buffer* is the address of the data to be written; *count* is the number of bytes to write from *buffer*; and *fd* is a file descriptor referring to the file to which data is to be written.

Changing File Offset

The *offset* argument specifies a value in bytes. (The *off_t* data type is a signed integer type specified by SUSv3.) The *whence* argument indicates the base point from which *offset* is to be interpreted, and is one of the following values:

`SEEK_SET`
: The file offset is set *offset* bytes from the beginning of the file.

`SEEK_CUR`
: The file offset is adjusted by *offset* bytes relative to the current file offset.

`SEEK_END`
: The file offset is set to the size of the file plus *offset*. In other words, *offset* is interpreted with respect to the next byte after the last byte of the file.

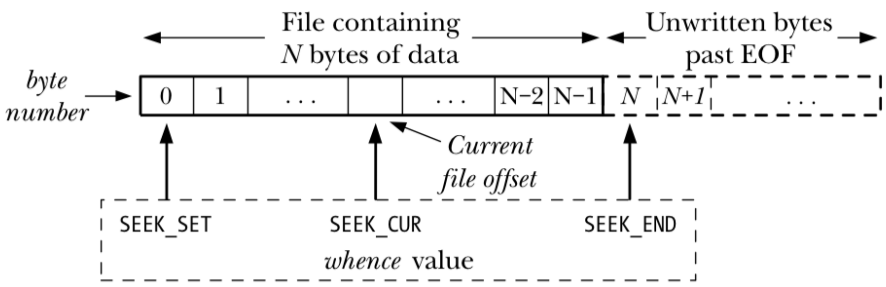
For each open file, the kernel records a *file offset*, sometimes also called the *read-write offset* or *pointer*. This is the location in the file at which the next *read()* or *write()* will commence. The file offset is expressed as an ordinal byte position relative to the start of the file. The first byte of the file is at offset 0.

The file offset is set to point to the start of the file when the file is opened and is automatically adjusted by each subsequent call to *read()* or *write()* so that it points to the next byte of the file after the byte(s) just read or written. Thus, successive *read()* and *write()* calls progress sequentially through a file.

The *lseek()* system call adjusts the file offset of the open file referred to by the file descriptor *fd*, according to the values specified in *offset* and *whence*.

```
#include <unistd.h>

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

Returns new file offset if successful, or –1 on error



What Do You Need To Do?

Read the HANDOUT and understand the important concepts.

Write a simple program to write an array of integers to a file. You need to treat the file as a randomly addressable stream (hint: use the lseek() system call). Your program should be able to read the integers in any given order. That is, given the index location you should be able to seek to that location and read the integer out.