Linux File I/O: Additional Topics

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

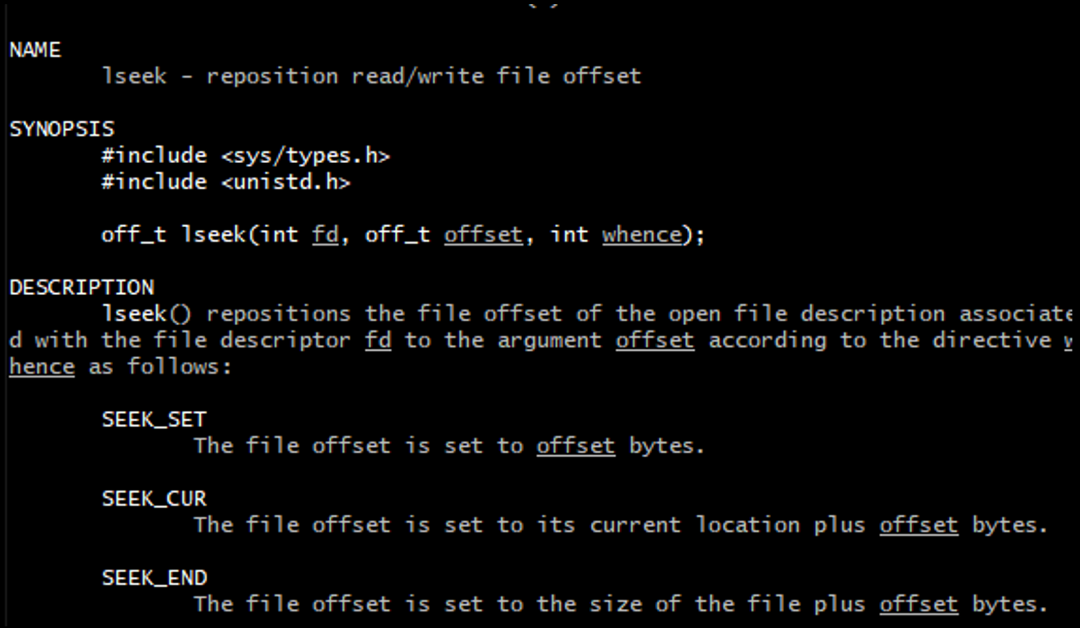
Access files positionally

Introduce Sparse Files

Introduce Multiplexed I/O

Reading Positionally

- SEEK_SET
  - Use the specified offset
- SEEK_CUR
  - Increment or decrement
- SEEK_END
  - Use EOF



```
NAME
       lseek - reposition read/write file offset

SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

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

DESCRIPTION
       lseek() repositions the file offset of the open file description associate
d with the file descriptor fd to the argument offset according to the directive w
hence as follows:

       SEEK_SET
              The file offset is set to offset bytes.

       SEEK_CUR
              The file offset is set to its current location plus offset bytes.

       SEEK_END
              The file offset is set to the size of the file plus offset bytes.
```

Alternative Positional Read/write

```
ssize_t pread (int fd, void *buf, size_t count, off_t pos);
```

```
ssize_t pwrite (int fd, const void *buf, size_t count, off_t pos);
```

- What happened to the origin argument?
  - Not used, always use SEEK_SET equivalent.
- Why use these over lseek()?
  - Avoids race conditions when multiple threads access a file (doesn't manipulate file position.)

Sparse Files

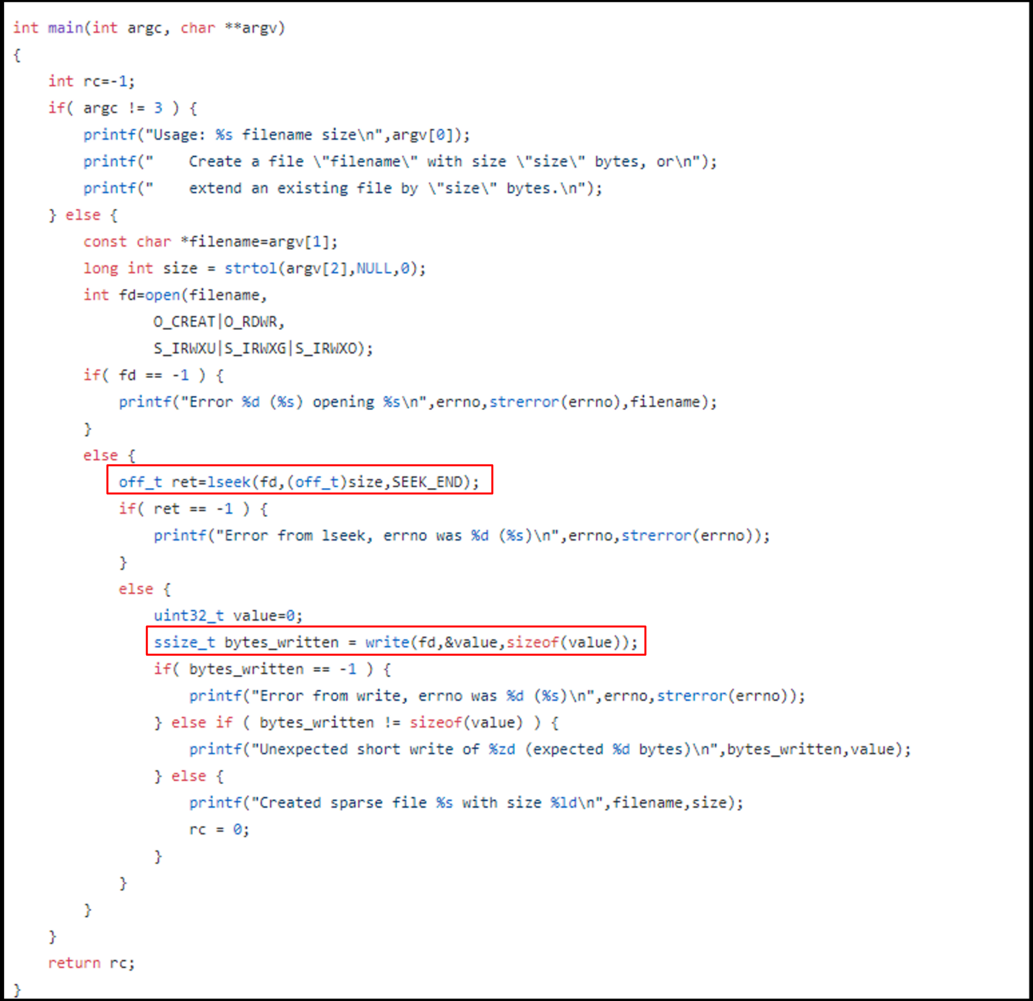
- lseek(fd, (off_t) 1000000000, SEEK_END);
- write(fd,&value,sizeof(value));
- What does this do?
- Extends a file 1GB past the end of the current file end location.

Sparse Files

- What if there’s less than 1GB of disk available?
  - Doesn’t use disk space
    - Linux will create a “hole” to represent the space in the file.
  - Reads of the hole will return 0s
  - Referred to as a “sparse file”

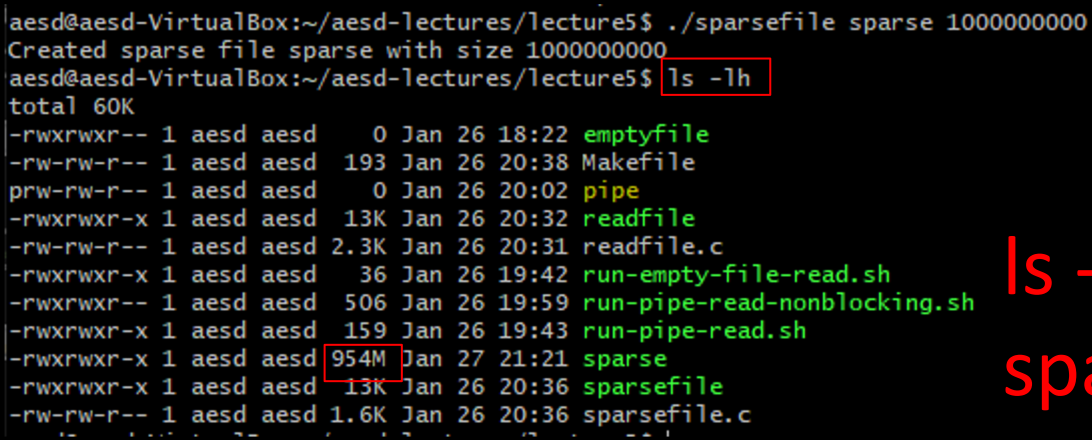
Sparse Files

- Seek relative to the end of the file
- Write a single 0 value



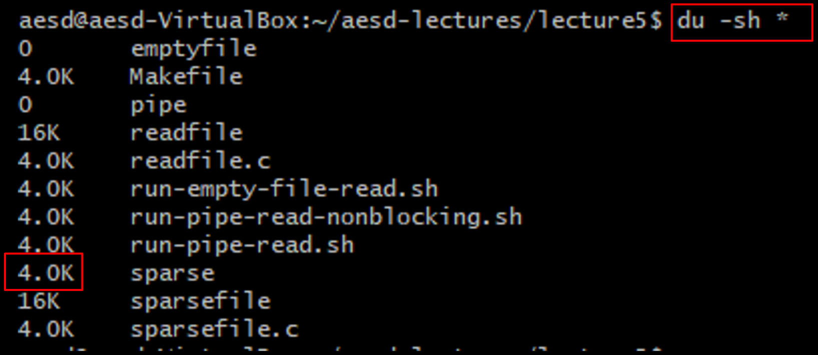
```
int main(int argc, char **argv)
{
    int rc=-1;
    if( argc != 3 ) {
        printf("Usage: %s filename size\n",argv[0]);
        printf("    Create a file \"filename\" with size \"size\" bytes, or\n");
        printf("    extend an existing file by \"size\" bytes.\n");
    } else {
        const char *filename=argv[1];
        long int size = strtol(argv[2],NULL,0);
        int fd=open(filename,
                O_CREAT|O_RDWR,
                S_IRWXU|S_IRWXG|S_IRWXO);
        if( fd == -1 ) {
            printf("Error %d (%s) opening %s\n",errno,strerror(errno),filename);
        }
        else {
            off_t ret=lseek(fd,(off_t)size,SEEK_END);
            if( ret == -1 ) {
                printf("Error from lseek, errno was %d (%s)\n",errno,strerror(errno));
            }
            else {
                uint32_t value=0;
                ssize_t bytes_written = write(fd,&value,sizeof(value));
                if( bytes_written == -1 ) {
                    printf("Error from write, errno was %d (%s)\n",errno,strerror(errno));
                } else if ( bytes_written != sizeof(value) ) {
                    printf("Unexpected short write of %zd (expected %d bytes)\n",bytes_written,value);
                } else {
                    printf("Created sparse file %s with size %ld\n",filename,size);
                    rc = 0;
                }
            }
        }
    }
    return rc;
}
```

Sparse Files



```
aesd@aesd-VirtualBox:~/aesd-lectures/lecture5$ ./sparsefile sparse 1000000000
Created sparse file sparse with size 1000000000
aesd@aesd-VirtualBox:~/aesd-lectures/lecture5$ ls -lh
total 60K
-rwxrwxr-- 1 aesd aesd    0 Jan 26 18:22 emptyfile
-rw-rw-r-- 1 aesd aesd  193 Jan 26 20:38 Makefile
prw-rw-r-- 1 aesd aesd    0 Jan 26 20:02 pipe
-rwxrwxr-x 1 aesd aesd  13K Jan 26 20:32 readfile
-rw-rw-r-- 1 aesd aesd 2.3K Jan 26 20:31 readfile.c
-rwxrwxr-x 1 aesd aesd   36 Jan 26 19:42 run-empty-file-read.sh
-rwxrwxr-- 1 aesd aesd  506 Jan 26 19:59 run-pipe-read-nonblocking.sh
-rwxrwxr-x 1 aesd aesd  159 Jan 26 19:43 run-pipe-read.sh
-rwxrwxr-x 1 aesd aesd 954M Jan 27 21:21 sparse
-rwxrwxr-x 1 aesd aesd  13K Jan 26 20:36 sparsefile
-rw-rw-r-- 1 aesd aesd 1.6K Jan 26 20:36 sparsefile.c
```

ls -h shows 945M for sparse file size



```
aesd@aesd-VirtualBox:~/aesd-lectures/lecture5$ du -sh *
0       emptyfile
4.0K    Makefile
0       pipe
16K     readfile
4.0K    readfile.c
4.0K    run-empty-file-read.sh
4.0K    run-pipe-read-nonblocking.sh
4.0K    run-pipe-read.sh
4.0K    sparse
16K     sparsefile
4.0K    sparsefile.c
```

du -sh shows 4k for sparse file size

Multiplexed I/O

- How to handle input from more than one file descriptor in your application?
    - Create a threads servicing each descriptor using blocking reads.
    - Use nonblocking IO (O_NONBLOCK)
        - Polling implementation

Multiplexed I/O

- Better solution than polling: Use multiplexed IO
    - Sleep until one of a set of file descriptors is available.
    - Watch several file descriptors at once
  - select()
  - pselect()
  - poll() - preferred mechanism