# CS450

# Programming Assignmet 4

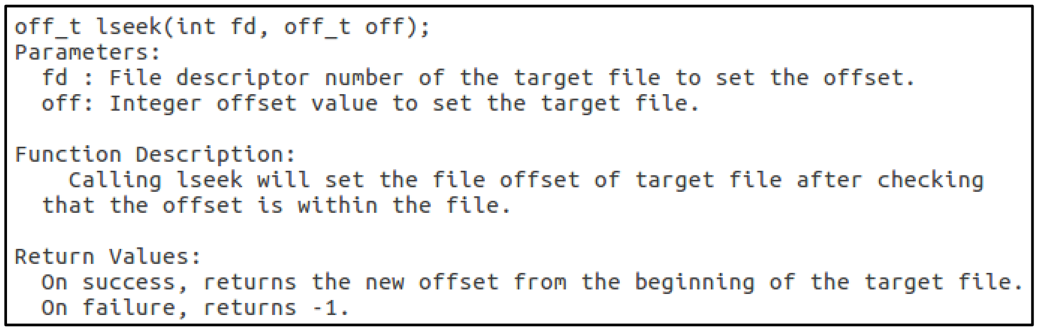
November 28, 2017

Ling Xing Zheng
A20343483

Venkata Krishna Chaitanya Polavarapu
A20378279

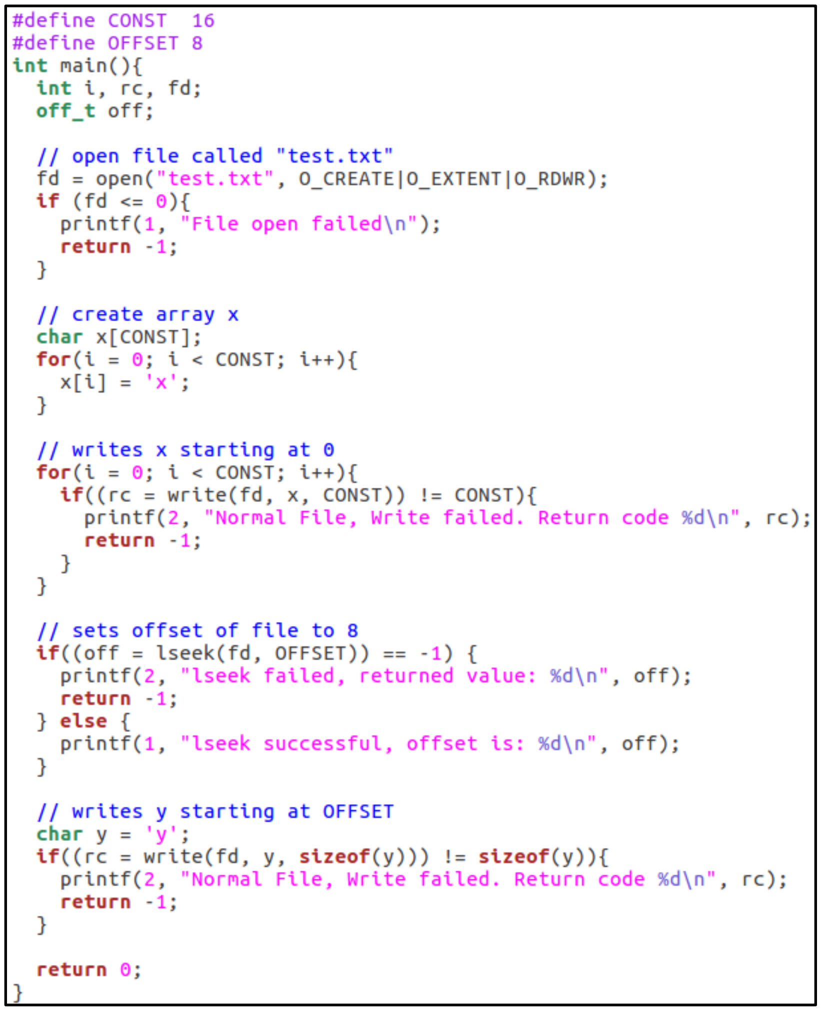
## Manual Page: lseek()

The system call **lseek()** sets the offset of the given file descriptor **fd** with the given offset **off**, as described below:



```
off_t lseek(int fd, off_t off);
Parameters:
  fd : File descriptor number of the target file to set the offset.
  off: Integer offset value to set the target file.

Function Description:
    Calling lseek will set the file offset of target file after checking
  that the offset is within the file.

Return Values:
  On success, returns the new offset from the beginning of the target file.
  On failure, returns -1.
```

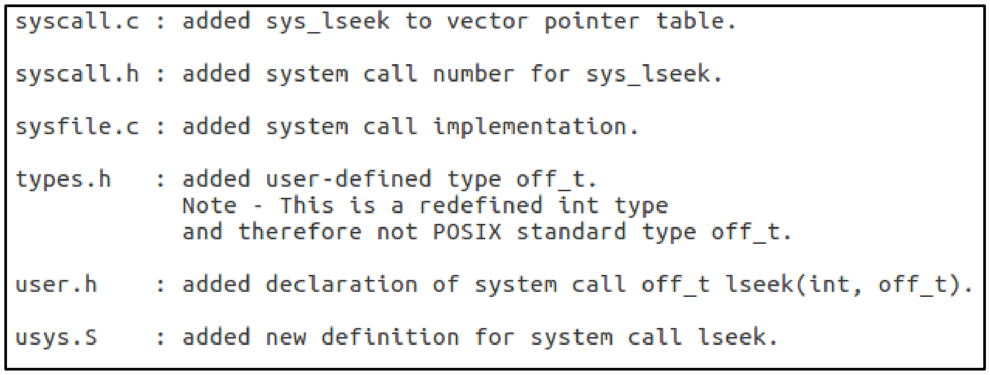
The following is an example code on how to use the syatem call. Note that an open file must exist in a process to utilize **lseek()**.



```
#define CONST  16
#define OFFSET 8
int main(){
  int i, rc, fd;
  off_t off;

  // open file called "test.txt"
  fd = open("test.txt", O_CREATE|O_EXTENT|O_RDWR);
  if (fd <= 0){
    printf(1, "File open failed\n");
    return -1;
  }

  // create array x
  char x[CONST];
  for(i = 0; i < CONST; i++){
    x[i] = 'x';
  }

  // writes x starting at 0
  for(i = 0; i < CONST; i++){
    if((rc = write(fd, x, CONST)) != CONST){
      printf(2, "Normal File, Write failed. Return code %d\n", rc);
      return -1;
    }
  }

  // sets offset of file to 8
  if((off = lseek(fd, OFFSET)) == -1) {
    printf(2, "lseek failed, returned value: %d\n", off);
    return -1;
  } else {
    printf(1, "lseek successful, offset is: %d\n", off);
  }

  // writes y starting at OFFSET
  char y = 'y';
  if((rc = write(fd, y, sizeof(y))) != sizeof(y)){
    printf(2, "Normal File, Write failed. Return code %d\n", rc);
    return -1;
  }

  return 0;
}
```

## Implementation:

The following files were modified to implement the new system call **lseek()**.



```
syscall.c : added sys_lseek to vector pointer table.

syscall.h : added system call number for sys_lseek.

sysfile.c : added system call implementation.

types.h   : added user-defined type off_t.
            Note - This is a redefined int type
            and therefore not POSIX standard type off_t.

user.h    : added declaration of system call off_t lseek(int, off_t).

usys.S    : added new definition for system call lseek.
```

- Modify user.h to add function declaration
  ```
  Off_t lseek(int, off_t);
  ```
  **User.h** must included to call system call functions.

- Include
  ```
  #define SYS_lseek 22
  ```
  In **syscall.h** as the system call number for **sys_lseek()**.

- In **syscall.c**, add **sys_lseek()** to the vector array of function pointers.

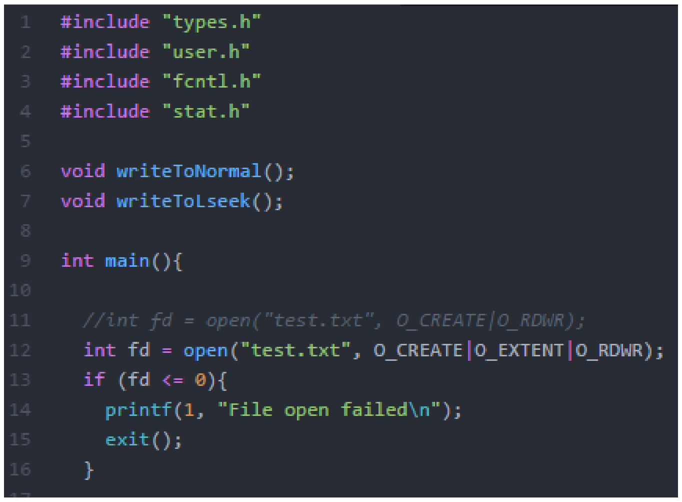
- **sys_lseek()** will be implementated in **sysfile.c**, where the file descriptor is used to obtain the pointer to the file's struct which can then be used to modify the offset.

- Finally, include
  ```
  SYSCALL(lseek)
  ```
  to **usys.S** to call **int** whenever **lseek** is called and hands control over to the kernal.

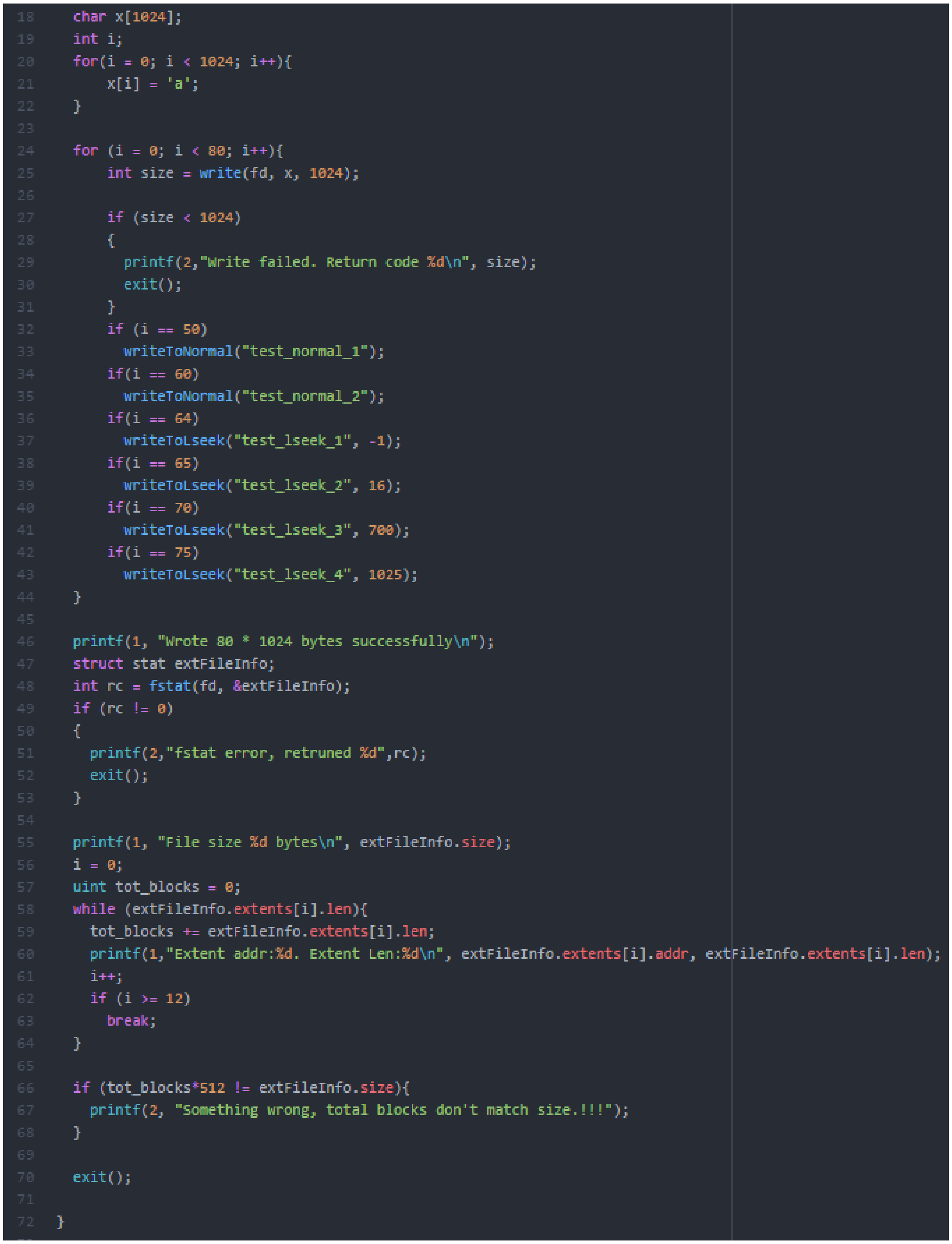
- Optionally, **off_t** can be user-defined in **types.h** using **int**. Otherwise, **lseek()** can be implemented using **uint** type.

## Testing:

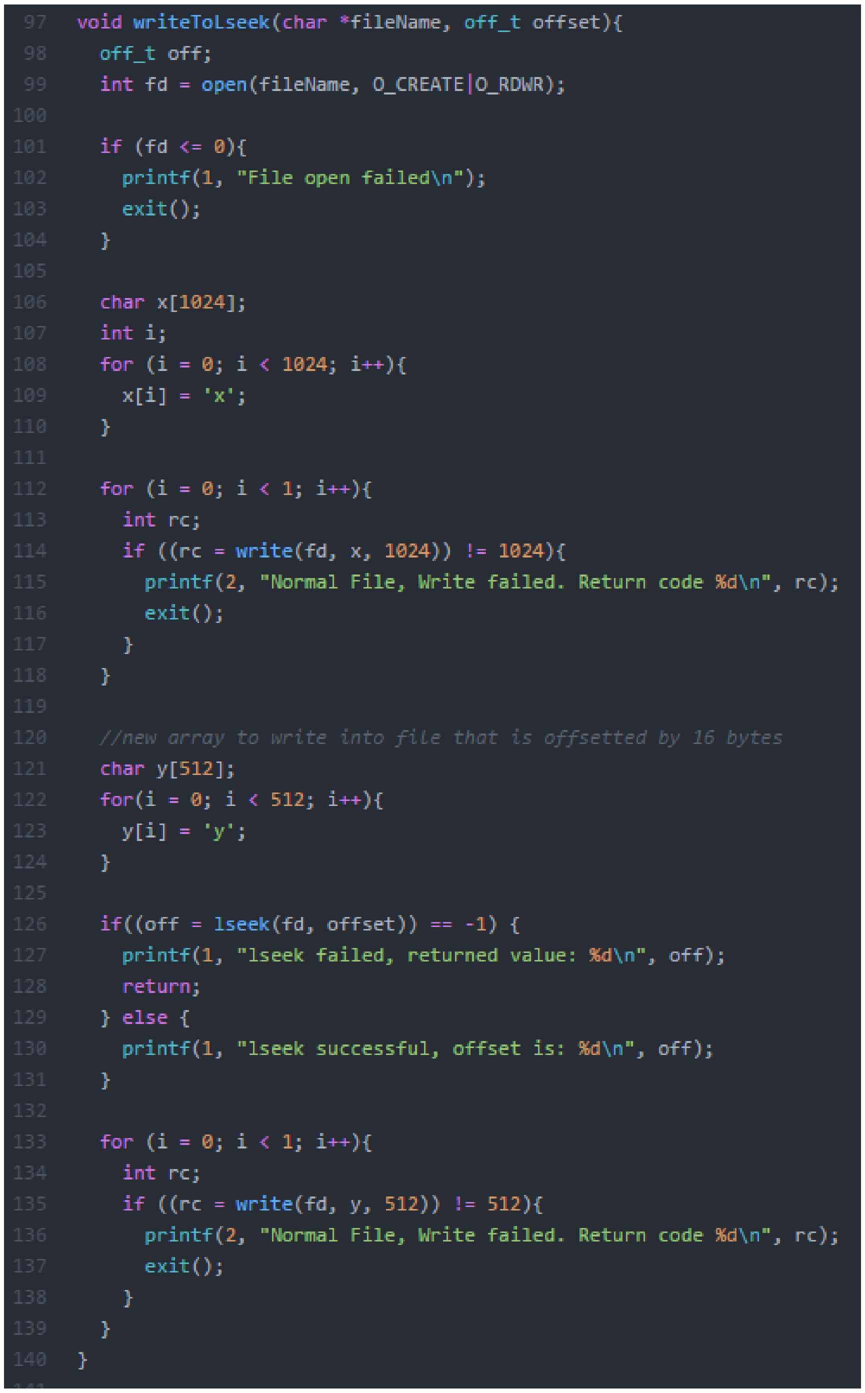
A user program called **assign4.c** was created and was later modified to test the functionality of the new extent-based file system along with the newly implemented **lseek** system call. The program was executed from xv6's shell.



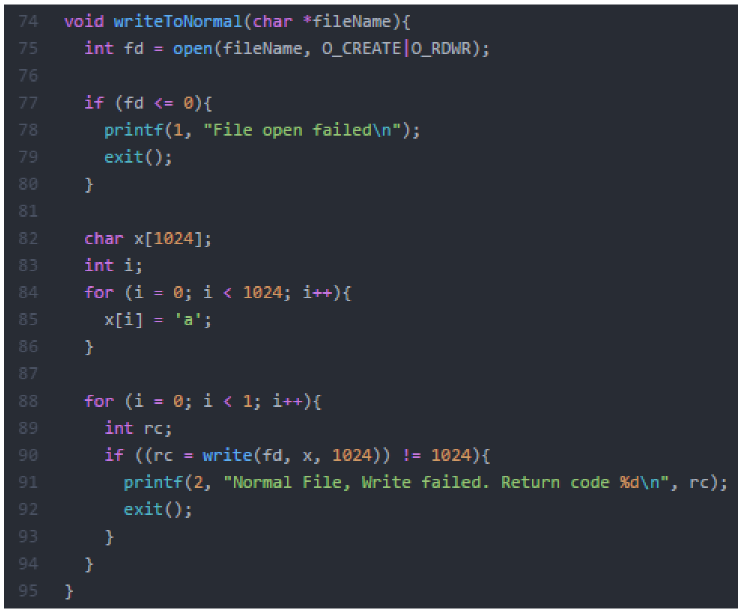
```
#include "types.h"
#include "user.h"
#include "fcntl.h"
#include "stat.h"

void writeToNormal();
void writeToLseek();

int main(){

  //int fd = open("test.txt", O_CREATE|O_RDWR);
  int fd = open("test.txt", O_CREATE|O_EXTENT|O_RDWR);
  if (fd <= 0){
    printf(1, "File open failed\n");
    exit();
  }
```



```
  char x[1024];
  int i;
  for(i = 0; i < 1024; i++){
      x[i] = 'a';
  }

  for (i = 0; i < 80; i++){
      int size = write(fd, x, 1024);

      if (size < 1024)
      {
        printf(2,"Write failed. Return code %d\n", size);
        exit();
      }
      if (i == 50)
        writeToNormal("test_normal_1");
      if(i == 60)
        writeToNormal("test_normal_2");
      if(i == 64)
        writeToLseek("test_lseek_1", -1);
      if(i == 65)
        writeToLseek("test_lseek_2", 16);
      if(i == 70)
        writeToLseek("test_lseek_3", 700);
      if(i == 75)
        writeToLseek("test_lseek_4", 1025);
  }

  printf(1, "Wrote 80 * 1024 bytes successfully\n");
  struct stat extFileInfo;
  int rc = fstat(fd, &extFileInfo);
  if (rc != 0)
  {
    printf(2,"fstat error, retruned %d",rc);
    exit();
  }

  printf(1, "File size %d bytes\n", extFileInfo.size);
  i = 0;
  uint tot_blocks = 0;
  while (extFileInfo.extents[i].len){
    tot_blocks += extFileInfo.extents[i].len;
    printf(1,"Extent addr:%d. Extent Len:%d\n", extFileInfo.extents[i].addr, extFileInfo.extents[i].len);
    i++;
    if (i >= 12)
      break;
  }

  if (tot_blocks*512 != extFileInfo.size){
    printf(2, "Something wrong, total blocks don't match size.!!!");
  }

  exit();

}
```



```c
void writeToLseek(char *fileName, off_t offset){
  off_t off;
  int fd = open(fileName, O_CREATE|O_RDWR);

  if (fd <= 0){
    printf(1, "File open failed\n");
    exit();
  }

  char x[1024];
  int i;
  for (i = 0; i < 1024; i++){
    x[i] = 'x';
  }

  for (i = 0; i < 1; i++){
    int rc;
    if ((rc = write(fd, x, 1024)) != 1024){
      printf(2, "Normal File, Write failed. Return code %d\n", rc);
      exit();
    }
  }

  //new array to write into file that is offsetted by 16 bytes
  char y[512];
  for(i = 0; i < 512; i++){
    y[i] = 'y';
  }

  if((off = lseek(fd, offset)) == -1) {
    printf(1, "lseek failed, returned value: %d\n", off);
    return;
  } else {
    printf(1, "lseek successful, offset is: %d\n", off);
  }

  for (i = 0; i < 1; i++){
    int rc;
    if ((rc = write(fd, y, 512)) != 512){
      printf(2, "Normal File, Write failed. Return code %d\n", rc);
      exit();
    }
  }
}
```



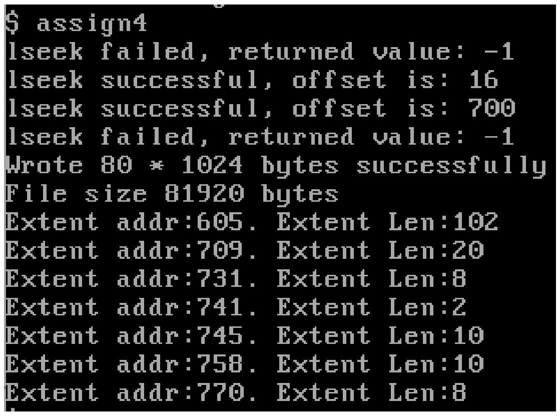
```c
void writeToNormal(char *fileName){
  int fd = open(fileName, O_CREATE|O_RDWR);

  if (fd <= 0){
    printf(1, "File open failed\n");
    exit();
  }

  char x[1024];
  int i;
  for (i = 0; i < 1024; i++){
    x[i] = 'a';
  }

  for (i = 0; i < 1; i++){
    int rc;
    if ((rc = write(fd, x, 1024)) != 1024){
      printf(2, "Normal File, Write failed. Return code %d\n", rc);
      exit();
    }
  }
}
```

**Test Case 1: negative offset**
This test case checks the functionality of **lseek()** when a negative offset is introduced. Unlike **lseek()** from **stdio.h**, this implementation does not deal with negative offsets and will therefore return -1 for error.

**Test Case 2: small offset**
This test case checks small offsets relative to the file size. In this case, the offset of the file will be set and the value returned to user program.

**Test case 3: large offset**
This test case checks larger offsets. Similarly, the system call will set and return the new offset of the file.

**Test case 4: larger offset (greater than the file size)**
This test case will determine that the offset is too large (and will therefore introduce holes in the file) and will return -1 to indicate error.



```
$ assign4
lseek failed, returned value: -1
lseek successful, offset is: 16
lseek successful, offset is: 700
lseek failed, returned value: -1
Wrote 80 * 1024 bytes successfully
File size 81920 bytes
Extent addr:605. Extent Len:102
Extent addr:709. Extent Len:20
Extent addr:731. Extent Len:8
Extent addr:741. Extent Len:2
Extent addr:745. Extent Len:10
Extent addr:758. Extent Len:10
Extent addr:770. Extent Len:8
```

Figure: output of assign4.c test program