

```
/* 11) Consider the last 100 bytes as a region. Write a C/C++ program to check whether the
region is locked or not. If the region is locked, print pid of the process which has
locked. If the region is not locked, lock the region with an exclusive lock, read
the last 50 bytes and unlock the region. */
```

```
// When multiple processes want to share some resource, it becomes essential to provide
// some form of mutual exclusion, so that only one process at a time can access the resource
```

```
// One of mechanism - File locking locks an entire file, Record locking allows a process to
// lock a specific portion of a file
```

```
// A file cannot have multiple exclusive locks at any time
```

```
// fcntl API for file locking : impose read or write locks on either a segment or an entire file
```

```
// All file locks set by a process will be unlocked when the process terminates
```

```
// The child process created through fork will NOT INHERIT the file lock
```

```
// Other API used: open , lseek , read , close
```

```
// lseek API allows a process to perform random access of data on any opened file
```

```
// Flow of implementation :
```

```
// Input : File name as command line argument
```

```
// Output :
```

```
// if last 100 bytes is locked in the file then details of Process which has already
// acquired the lock is printed
```

```
// else
```

```
// lock last 100 bytes , read last 50 bytes, display content, then unlock region
```

```
/* if a valid file name has been passed as command line argument
```

```
fork a child;
```

```
set the values for exclusive locking last 100 bytes;
```

```
try getting a exclusive lock;
```

```
if unable to get the lock
```

```
then check for process which has got the lock, print its details and return
```

```
if acquiring of lock was successful
```

```
set the file descriptor to read from last 50th byte; read last 50 bytes;
```

```
set the values for unlocking the last 100 bytes; unlock the region and return */
```

```
#include<stdio.h>
```

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

```
#include<fcntl.h>
```

```
#include<unistd.h>
```

```
int main( int argc , char *argv[] ) // Check if file name is passed as command line argument
```

```
{ // As fork is used - to get separate outputs, set stdout as buffered, setbuf(stdout, temp);
```

```
char temp[1000]; setbuf(stdout,temp);
```

```
struct flock fvar;//flock structure variable
```

```
int fdesc; //file descriptor
```

```
char buf; //buffer for reading content from file
```

```
int rc; //variable for count of bytes read by read API
```

off_t offset; //variable to hold return value from lseek API

pid_t pid=fork(); // unconditional fork , can print pid and parent pid for clarity
// let scheduler decide order of execution of parent and child, can be four possibilities

fdesc = open (argv[1],O_RDWR); // If open call fails, print error message, return

offset = lseek (fdesc , -100 , SEEK_END); // If lseek fails, print error message, return

fvar.l_type = F_WRLCK; //set a write exclusive lock on specified region
fvar.l_whence = SEEK_CUR;
fvar.l_start = 0; //start from SEEK_CUR current file descriptor location
fvar.l_len = 100; //lock 100 bytes from SEEK_CUR

if(fcntl(fdesc, F_SETLK , &fvar) == -1)// Check whether the region is locked
{ // fcntl unsuccessful, region is locked, print pid of the process which acquired locked
printf("\n ----- \n File has been locked by : \n");
while(fcntl(fdesc,F_GETLK,&fvar) != -1 && fvar.l_type != F_UNLCK)
{
printf("\n File : %s is locked by process with pid : %u", argv[1],fvar.l_pid);
printf(" from %ld th byte in file for %ld ", fvar.l_start, fvar.l_len);
printf(" number of bytes , for %s \n",(fvar.l_type == F_WRLCK ? "write" : "read"));

if(!fvar.l_len) **break**;

fvar.l_start +=fvar.l_len ;
fvar.l_len = 0;

}

else

{// fcntl successful, as region was not locked hence process was able to get exclusive
printf("\n ----- \n");
printf("\n\n File : %s was not locked and acquiring of Exclusive Lock was", argv[1]);
printf(" successful By Process ID : %u \n", getpid());

offset = lseek (fdesc , -50 , SEEK_END);// read the last 50 bytes, print error if lseek fails

printf("\n\n Last 50 bytes of file : %s = \n", argv[1]);

while((rc = read(fdesc, &buf , 1)) > 0) printf("%c", buf);

offset = lseek (fdesc , -100 , SEEK_END); // and unlock the region
fvar.l_type = F_UNLCK ;//unlocks a specific region
fvar.l_whence = SEEK_CUR;
fvar.l_start = 0;
fvar.l_len = 100;

fcntl(fdesc,F_SETLKW,&fvar); // print error if fcntl fails

printf("\n File Unlocked successfully\n\n");
}

return 0;

}

/* Output:

File : File Locking OSandUSP 11.c was not locked and acquiring of Exclusive Lock was successful By Process ID : 6922

Last 50 bytes of file : File Locking OSandUSP 11.c =
ed successfully\n\n");

```
    }  
    return 0;  
}
```

File Unlocked successfully

File has been locked by :

File : File Locking OSandUSP 11.c is locked by process with pid : 6922 from 4626 th byte in file for 100 number of bytes , for write

*/