**Advanced Lab –2:**

**STEPS TO EXECUTE THIS PROGRAM IN XV6**

1. **TOUCH COMMAND IN XV6:**

**STEP1: Open Vi Editor**

Syntax : vi touchex.c

**STEP2: Type the below code(Press ‘i’ to enter into insert mode)**

|  |
| --- |
| #include"types.h" |
| #include "user.h" |
| #include "fcntl.h" |
| #include "fs.h" |
|  |
| int main(int argc,char \*argv[]) |
| { |
| if(argc<2) |
| { |
| printf(1,"Usage: touch [files]...\n"); |
| exit(); |
| } |
| int i,err; |
| for(i=1;i<argc;i++) |
| { |
| if((err=open(argv[i],O\_CREATE|O\_RDWR)) < 0) |
| { |
| printf(1,"touch: error where creating %s\n",argv[i]); |
| exit(); |
| } |
| close(err); |
| } |
| exit(); |

}

**STEP 3: Press Esc : wq to save and quit from the editor after typing the program.**

**STEP 4: Open Makefile**

Syntax: vi Makefile

**STEP 4: IN Makefile program do the following changes in two sections:**

In the **Makefile**, there are two places in which we need to put entries. Find the place with some lines like the following. We have to add a line as shown below to notify about our new program. **UPROGS= \**

**\_cat\**

**\_echo\**

**\_forktest\**

**\_grep\**

**\_init\**

**\_kill\**

**\_ln\**

**\_ls\**

**\_mkdir\**

**\_rm\**

**\_sh\**

**\_stressfs\**

**\_usertests\**

**\_wc\**

**\_zombie\**

**\_touchex\**

Similarly, find the place with the lines like below. Add an entry as shown to indicate that we have a program called **my.c** there.

**EXTRA=\ mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\ ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\ touchex.c\**

**printf.c umalloc.c\ README dot-bochsrc \*.pl toc.\* runoff runoff1 runoff.list\ .gdbinit.tmpl gdbutil\**

Now, our Makefile and our user program is ready to be tested. Enter the following commands to compile the whole system.

Syntax:

make clean

make

Now, start xv6 system on QEMU and when it booted up, run ls command to check whether our program is available for the user.

Syntax:

make qemu-nox

$ls

Check whether touchex is listed in the output.If yes then use that as a command.

$touchex f1.txt

$ls

The output should show f1.txt in the list of files.

Useful touch command examples in Linux

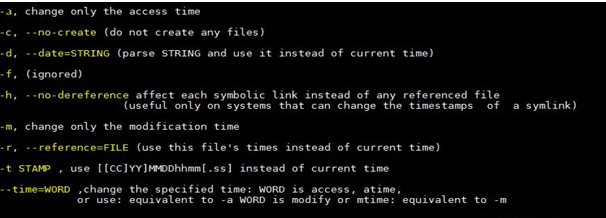
Touch command is used to create empty files and also changes the timestamps of existing files on Unix & Linux System.

Changing timestamps here means updating the access and modification time of files and directories.

Let’s have a look on the syntax and options used in touch command,

Syntax: # touch {options} {file}

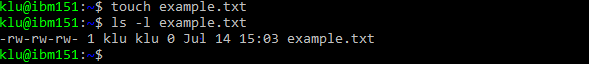
Options used in touch command,



In this article we will walk through 9 useful touch command examples in Linux,

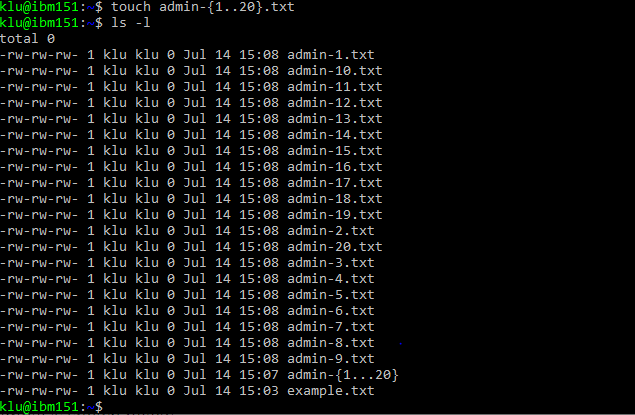
**Example: 1 Create an empty file using touch**

To create an empty file using touch command on Linux systems, type touch followed by the file name, example is shown below



**Example:2 Create empty files in bulk using touch**

There can be some scenario where we have to create lots of empty files for some testing, this can be easily achieved using touch command,



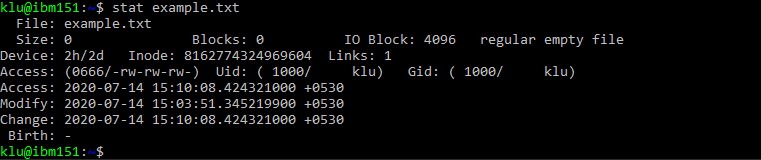
In the above example we have created 20 empty files with name admin-1.txt to admin-20.txt, you can change the name and numbers based on your requirements.

**Example:3 Change / Update access time of a file and directory**

Let’s assume we want to change access time of a file called “**example.txt**“, to do this use ‘**-a**‘ option in touch command followed by file name, example is shown below,



Now verify whether access time of a file has been updated or not using ‘stat’ command



**2) TAIL COMMAND IN XV6:**

**STEP1: Open Vi Editor**

Syntax : vi tailex.c

**STEP2: Type the below code(Press ‘i’ to enter into insert mode)**

|  |
| --- |
|  |
|  | #include "stat.h"  #include “types.h” |
|  | #include "user.h" |
|  | #include "fcntl.h" |
|  |  |
|  | char buf [1024]; |
|  |  |
|  | void tail (int fd, char \*name, int line) { |
|  | int i, n; //here the size of the read chunk is defined by n, and i is used to keep a track of the chunk index |
|  | int l=0; // here total lines are defined by l, and the character count in the string is defined by c |
|  | int count = 0; |
|  |  |
|  | int temp = open ("temporary", O\_CREATE | O\_RDWR); // creating a temporary file to store the data/ user input in it then print it |
|  |  |
|  | while((n = read(fd, buf, sizeof(buf))) > 0 ) |
|  | { |
|  | write (temp, buf, n); // writing the n chunks of data from buffer to the temp file |
|  |  |
|  | for(i=0;i<=n ;i++) |
|  | { |
|  | if(buf[i] != '\n') // checking for end of line, if so then conitnue |
|  | { |
|  | continue; |
|  | } |
|  | else // else count the number of lines in the file |
|  | { |
|  | l++; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | close (temp); |
|  |  |
|  | if (n < 0) |
|  | { |
|  | printf(1, "tail: read error\n"); |
|  | exit(); |
|  | } |
|  |  |
|  | temp = open ("temporary", 0); // opening the file reading the file from the beginning |
|  |  |
|  | while((n = read(temp, buf, sizeof(buf))) > 0 ) |
|  | { |
|  | for (i = 0; i<n; i++) |
|  | { |
|  | if (count >= (l - line)) // to check if the counter value exceeds the last n lines to be printed or not, if yes then print those lines |
|  | { |
|  | printf(1,"%c",buf[i]); |
|  | } |
|  | else if (l < line) // to check i fthe total number of lines in the file are less than the requested number of lines |
|  | { |
|  | printf(1,"%c",buf[i]); |
|  | } |
|  | else if (buf[i] == '\n') // to check for end of line in the buffer, to increase the count value |
|  | { |
|  | count++; |
|  | } |
|  | } |
|  | } |
|  | close (temp); |
|  |  |
|  | unlink("temporary"); // delete the file before closing the function |
|  | } |
|  |  |
|  |  |
|  | int |
|  | main(int argc, char \*argv[]) { |
|  | int i; |
|  | int fd = 0; // when the file is not specified, then it will take input from the user |
|  | int x = 10; // will read the last 10 lines by default |
|  | char \*file; // pointer to the name of the file |
|  | char a; |
|  |  |
|  | file = ""; // in the case when no file name is specified, it will take input from the user |
|  |  |
|  | if (argc <= 1) |
|  | { |
|  | tail(0, "", 10); // handles the default case of taking input from user and printing only last 10 lines |
|  | exit(); |
|  | } |
|  |  |
|  | else { |
|  | for (i = 1; i < argc; i++) |
|  | { |
|  | a = \*argv[i]; // assigns the char value of the argv to the var a |
|  |  |
|  | if (a == '-') |
|  | { // it means that -NUM is provided, hence limited number of lines are to be printed |
|  | argv[i]++; |
|  | x = atoi(argv[i]++); |
|  | } |
|  |  |
|  | else |
|  | { // if a !='-' then it implies that number of lines are not defined and hence default lines will print |
|  | if ((fd = open(argv[i], 0)) < 0) |
|  | {// this will execute if the file is unable to open |
|  | printf(1, "tail: cannot open %s\n", argv[i]); |
|  | exit(); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | tail(fd,file, x); |
|  | close(fd); |
|  | exit(); |
|  |  |
|  | } |
|  | } |

**STEP 3: Press Esc : wq to save and quit from the editor after typing the program.**

**STEP 4: Open Makefile**

Syntax: vi Makefile

**STEP 4: IN Makefile program do the following changes in two sections:**

In the **Makefile**, there are two places in which we need to put entries. Find the place with some lines like the following. We have to add a line as shown below to notify about our new program. **UPROGS= \**

**\_cat\**

**\_echo\**

**\_forktest\**

**\_grep\**

**\_init\**

**\_kill\**

**\_ln\**

**\_ls\**

**\_mkdir\**

**\_rm\**

**\_sh\**

**\_stressfs\**

**\_usertests\**

**\_wc\**

**\_zombie\**

**\_tailex\**

Similarly, find the place with the lines like below. Add an entry as shown to indicate that we have a program called **my.c** there.

**EXTRA=\ mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\ ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\ tailex.c\**

**printf.c umalloc.c\ README dot-bochsrc \*.pl toc.\* runoff runoff1 runoff.list\ .gdbinit.tmpl gdbutil\**

Now, our Makefile and our user program is ready to be tested. Enter the following commands to compile the whole system.

**Syntax:**

make clean

make

Now, start xv6 system on QEMU and when it booted up, run ls command to check whether our program is available for the user.

**Syntax:**

make qemu-nox

**$ls**

Check whether touchex is listed in the output.If yes then use that as a command.

**$tailex f1.txt**

**Output:**

Last 10 lines of the file f1.txt will be displayed by default

**STEPS TO EXECUTE THIS PROGRAM IN UNIX PROGRAMMING**

**(REFERENCE TEXT BOOK: SUMITABHA DAS)**

1. **lseek: Positioning the Offset Pointer(17.8)**

**lseek** doesn’t do any physical I/O. It simply moves the file offset pointer to a specified

point where the next I/O operation will take place. Here’s its syntax:

off\_t lseek(int *fildes*, off\_t *offset*, int *whence*);

The *offset* and *whence* arguments together control the location of the file’s offset pointer.

*offset* signifies the position (positive or negative) of this pointer relative to *whence*, which

can take one of three values:

SEEK\_SET Offset pointer set to beginning of file.

SEEK\_END Offset pointer set to end of file.

SEEK\_CUR Offset pointer remains at current location

With some restrictions, *offset* can be a positive or negative integer, so it is represented

by a signed data type.

For instance,

lseek(fd, 10, SEEK\_CUR) fd *obtained from prior* open

moves the pointer forward by 10 characters from its current position, and

lseek(fd, -10, SEEK\_END) *Negative offset*

sets the pointer 10 characters before EOF.

You can’t have a negative *offset* with *whence*set to SEEK\_SET, but strangely enough you can have a positive *offset* with *whence* atSEEK\_END.

In this case, the pointer moves beyond EOF, thus creating a *sparse* file—also

called a file with a “hole.” Sparse files find use in database applications, but our next

program also moves the offset pointer beyond EOF.

**lseek** returns the position of the pointer in bytes from the beginning of the file.

This value can be used to determine the size of the file:

size = lseek(fd, 0, SEEK\_END); *This returns the file size*

Unlike **read** and **write**, which work with practically all file types, **lseek** works only

with those files that are capable of “seeking.” It doesn’t work with the terminal file or with a socket or pipe, and it is mainly used for disk files.

Example on lseek:

#include<sys/types.h>

#inlclude<sys/stat.h>

#include<fcntl.h>

#include<unistd.h>

int main()

{

int n,f,f1;

char buff[10];

f=open(“f1.txt”,O\_RDWR);//1234567890abcdefghijx1x2x3x4x5

read(f,buff,10);//12345678904

write(1,buff,10);

lseek(f,10,SEEK\_CUR);

read(f,buff,10);//x1x2x3x4x5

write(1,buff,10);

}

Execution:

1. open vi editor type the above program
2. Compile: gcc lseekex.c
3. ./a.out
4. **reverse\_read.c: Reading a File in Reverse(17.8.1)**

You can’t read a file from the end to the beginning using the standard UNIX utilities

(except **perl**), but using **lseek** in a C program, **reverse\_read.c**, you can.

You have to first move the file pointer to one character beyond EOF. Then use a loop to

move the pointer back by two positions every time a character is read.

Hence, we use a single-character buffer with **read**

and **write**. While **read** advances the pointer one byte forward, the next **lseek** takes it

back by two bytes.

#include <fcntl.h>

#include <unistd.h> /\* For STDOUT\_FILENO \*/

#include <stdio.h>

int main(int argc, char \*\*argv) {

int size, fd;

char buf; /\* Single-character buffer \*/

char \*mesg = "Single filename required\n";

if ((fd = open(argv[1], O\_RDONLY)) == -1)

perror("open");

lseek(fd, 1, SEEK\_END); /\* Pointer taken to EOF + 1 first \*/

while (lseek(fd, -2, SEEK\_CUR) >= 0) { /\* and then back by

two bytes \*/

if (read(fd, &buf, 1) != 1)

perror("read");

if (write(STDOUT\_FILENO, &buf, 1) != 1)

perror("write");

}

close(fd); /\* Can have error here too \*/

exit(0); /\* exit doesn't return - hence no error \*/

}

Execution:

1. open vi editor type the above program
2. Compile: gcc reverse\_read.c
3. ./a.out f1.txt
4. **Run-Time Stack Usage in 32-Bit GCC**

**(Reference Book: K.C Wang System Programming in Unix/Linux)**

