**Operating Systems(19CS2106S)**

**Lab – 4**

1. **Runs two programs in a pipeline Child runs cat, parent runs tr**

**Program:**

Pipe.c

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdarg.h>

void quit (char \*message, int exit\_status)

{

printf(" %s",message);

exit(exit\_status);

}

int main()

{

int fd[2]; /\* To be filled up by pipe() \*/

if (pipe(fd < 0) /\* Now have four descriptors for pipe \*/

printf("os lead");

switch (fork())

{

case -1: quit("fork", 2);

case 0: close(fd[0]); /\* CHILD - Close read end first\*/

dup2(fd[1], STDOUT\_FILENO); /\*Connect stdout to write end \*/

close(fd[1]); /\* and close original descriptor \*/

execlp("cat", "cat", "os.txt", (char \*) 0);

quit("cat", 3);

default:close(fd[1]); /\*PARENT - Close write end first \*/

dup2(fd[0], STDIN\_FILENO); /\*Connect stdin to read end\*/

close(fd[0]); /\* and close original descriptor \*/

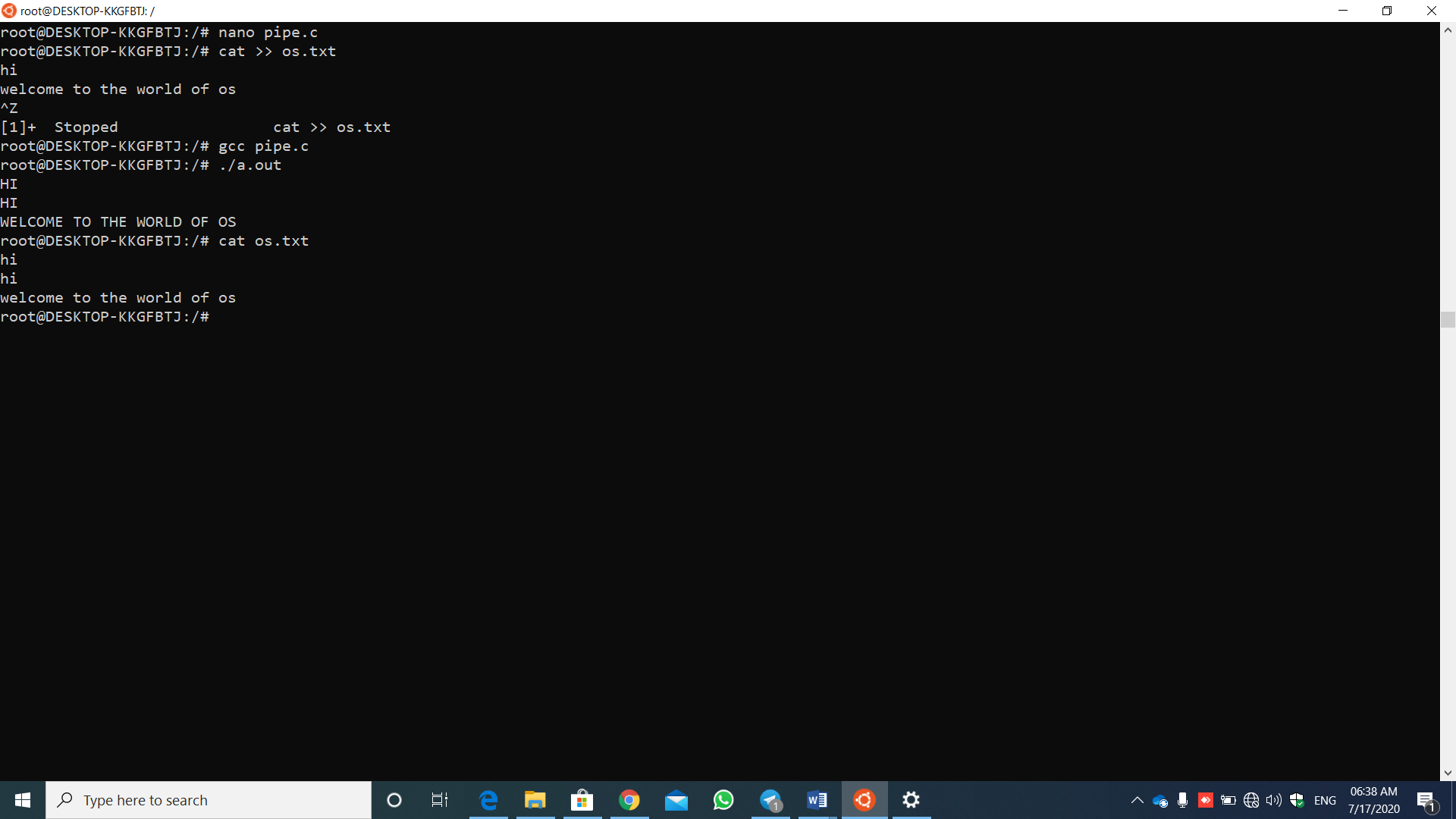
execlp("tr", "tr", "[a-z]","[A-Z]", (char \*) 0);

quit("tr", 4);

}

}

**Output:**



**Steps:**

1.create a file having c program in it by using any editor (parent file).

2.create one more file by using any editor and place some content in it (child file).

3.execute the c program with gcc command (gcc filename.c)

4. (./a.out ) and we get the final output which is present in the child file

**Commands**:

1. exit() - terminates the process normally.

Syntax : void exit ( int status );

2.File descriptor

File descriptor is integer that uniquely identifies an open file of the process.

**Read from stdin => read from fd 0** : Whenever we write any character from keyboard, it read from stdin through fd 0.  
**Write to stdout => write to fd 1** : Whenever we see any output to the video screen, it’s written to stdout in screen through fd 1.  
**Write to stderr => write to fd 2** : We see any error to the video screen, it is also from that file write to stderr in screen through fd 2.

3.dup2()

Syntax: int dup2(int oldfd, int newfd);

**oldfd:** old file descriptor

**newfd** new file descriptor which is used by dup2() to create a copy.

1. close() - close a file descriptor

Synatx : int close(int *fd*);

### 5.execlp -- Overlay Calling Process and Run New Program

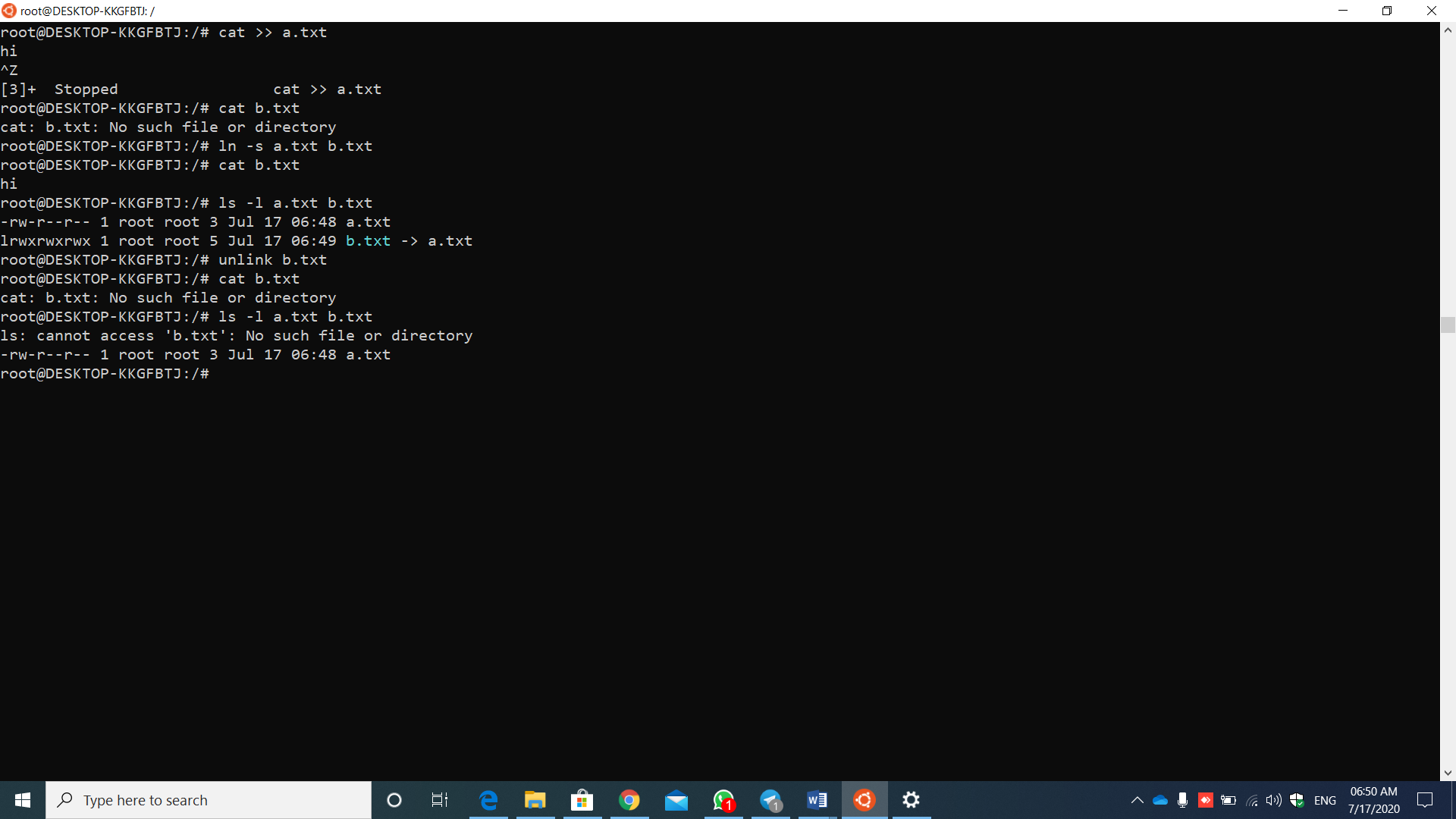
Syntax : int execlp(const char \*path, const char \*arg0, ..., NULL);

Reference Links:

<https://docs.python.org/2.0/lib/os-fd-ops.html>

3. **Create the filename "another.txt" and link it to the other file. Later delete it using unlink**

**Program & output**:



Steps:

1.create a file and place some content in it.

2. now check by placing some other file which does not exist.

3.now link the file we created and the file which does not exist.

4.now check the output of the file which does not exist. As we created the link between the existed file and unexisted file the data in the existence file will come and place in the unexistence file.

5. now check the path by using ls command

6.unlink the file which has been linked by usimg unlink command

7. now check the path by using ls command

8. it should show no such file or directory for the unexistence file and it should show the path for the existence file.

Reference Links:

Link Command:

[https://www.geeksforgeeks.org/soft-hard-links-unixlinux/#:~:text=A%20link%20in%20UNIX%20is,to%20the%20same%20file%2C%20elsewhere.](https://www.geeksforgeeks.org/soft-hard-links-unixlinux/#:~:text=A link in UNIX is,to the same file, elsewhere.)

Unlink command:

<https://www.tutorialspoint.com/unix_system_calls/unlink.htm>

4. **Unlinking an opened file**

unlink.c

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

int main(argc, argv)

int argc;

char \*argv[];

{

int fd;

char buf[10241];

struct stat statbuf;

if(argc !=2) /\* need a parameter \*/

exit(1);

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

if(fd ==-1) /\* open fails \*/

exit(1);

if(unlink(argv[1])==-1) /\* unlink file just opened \*/

exit(1);

if(stat(argv[1], &statbuf)==-1) /\* stat the file by name\*/

printf("stat %s fails as it should\n", argv[1]);

else

printf("stat %s succeeded!!!!\n", argv[1]);

if(fstat(fd, &statbuf)==-1)

{ /\* stat the file by fd \*/

printf("fstat %s fails!!!\n", argv[1]);

}

else

{

printf("fstat %s succeeds as it should\n", argv[1]);

}

while(read(fd,buf,sizeof(buf)>0)) /\* read open/unlinked file \*/

{

printf("%1024s", buf); /\* prints 1K byte field \*/

}

}

Steps:

1.create a file having c code in it

2. run it using the gcc command(compile)

3.execute it using (./a.out) command

4.we get the status of stat and fstat

Finally we get the data as output by having empty spaces in between and where the data is present means there will be no linked to the data.

Commands:

Unlink() - **int unlink(const char \****pathname***);**

**unlink**() deletes a name from the filesystem. If that name was the last link to a file and no processes have the file open the file is deleted and the space it was using is made available for reuse

read() - read from a file descriptor

Stat(),fstat() - get file status

Synatx:

**int stat(const char \****pathname***, struct stat \****statbuf***);**

**int fstat(int** *fd***, struct stat \****statbuf***);**

Output:

