Assignment - I

**Q1. Write a program that does the following :**

**a) It should accept two command line arguments.**

**(These two are file names. One is the *source file* and the another is the *destination file*)**

**b) It creates a pipe.**

**c) It creates a child process**

**d) Now the parent reads the data in the source file, sends it over the pipe. The child reads the data, over the pipe and writes it in the destination file. Whenever there is no more data to read, the child exits. The destination file becomes a copy of the source file. (Do error handlings and proper indentation of your source code).**

**Syntax:**

#include<stdio.h>

#include<string.h>

#include<unistd.h>

#include<sys/types.h>

#include<stdlib.h>

int main(int argc , char \*argv[])

{

if(3 != argc)

{

printf("Wrong no. of arguments\n");

exit(0);

}

FILE \*source,\*destination;

if(NULL == (source=fopen(argv[1],"r")))

{

printf("Error in opening file1\n");

exit(0);

}

if(NULL == (destination=fopen(argv[2],"w")))

{

printf("Error in opening file2\n");

exit(0);

}

pid\_t pid;

int fd[2] , i=0;

if(-1 == pipe(fd))

{

printf("Error in pipe creation\n");

}

pid = fork();

if(pid<0)

{

printf("fork fail\n");

}

if(pid>0)

{

char c, ch;

while((ch = fgetc(source)) != EOF)

{

fputc(ch, destination);

}

fclose(source);

fclose(destination);

printf("Content of file (%s) is \n",argv[2]);

if(NULL == (destination=fopen(argv[2],"r")))

{

printf("Error in opening file2\n");

exit(0);

}

c=getc(destination);

while(c != EOF)

{

putchar(c);

c=getc(destination);

}

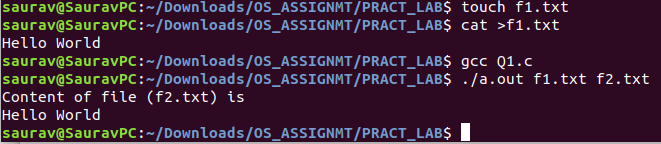
fclose(destination);

}

return 0;

}

**Output:**



**Q2. Write a shell script that changes (means rename,  *program\_1.c* needs to be renamed to *program\_1.cpp*) all .c files to *.cpp* files in your current directory. Finally it should also print the number of renamings done as a summary.**

**(Use the two extension names as command line input and do proper error handlings too. Do not use the "find" command in your script.)**

**Syntax:**

#!/bin/bash

n=1

for file in \*.cpp

do

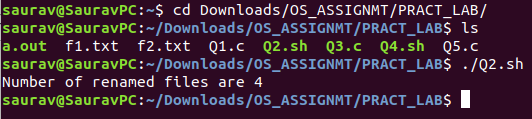
mv "$file" "${file%.cpp}.c"

(( n++ ))

done

echo "Number of renamed files are $n"

**Output:**



**Q3. Write a program which will concatenate "n" files given as command line arguments in *reverse order.***

**(Say your final executable is myconcat and you have invoked it as "*myconcat f1 f2 f3 f4*". *Here f4 will contain the data of f3,f2 and f1 in that order.***

**You need to implement error handling that is say if any of the file in the sequence is not existing or there is any other other, you need to report that.**

**If the program is invoked with two arguments say "myconcat f1 f2", f2 will be a copy of f1.**

**If the program is invoked with one argument say "myconcat f1", display contents of f1 on screen.**

**If the program is invoked with no arguments (that is no file name), it gives an error message and exit.**

**You need to use system calls like open, read, write, close.**

**Syntax:**

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<fcntl.h>

#include<unistd.h>

#define BUFFER\_SIZE 9999

int main(int argc , char \*argv[])

{

if(argc<2)

{

printf("Wrong no. of arguments\n\n");

exit(0);

}

int fd[argc-1], i=0, j=0;

char buffer[BUFFER\_SIZE];

ssize\_t in,out;

if(argc==2)

{

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

in = read(fd[argc-1],&buffer,BUFFER\_SIZE);

printf("%s\n",buffer);

exit(0);

}

for(i=1;i<=argc-2;i++)

{

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

if(-1==fd[i-1])

{

printf("Error in opening file '%s'\n",argv[i]);

exit(0);

}

}

fd[argc-2]=open(argv[i],O\_WRONLY);

for(i=argc-3;i>=0;i--)

{

in = read(fd[i],&buffer,BUFFER\_SIZE);

out = write(fd[argc-2],&buffer,(ssize\_t)in);

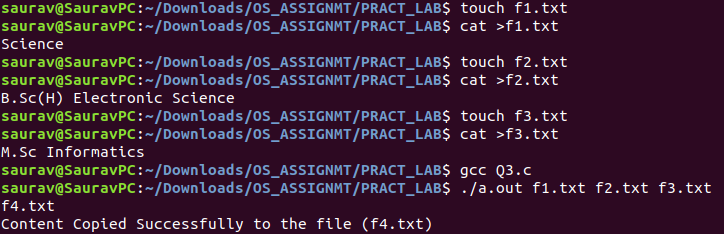
}

printf("Content Copied Successfully to the file (%s) \n",argv[4]);

return 0;

}

**Output:**



**Q4. Write a shell script to find the names of all "zero" (0) byte files in your current directory and move them to a folder called "ZERO". If the folder "*ZERO" exists then do not create it* else create *the folder "ZERO" in your current directory.***

**(If no zero byte file is there in the current directory, then an appropriate message should be provided and the folder "ZERO" need not to be created, if it does not exists)**

**Syntax:**

#!/bin/bash

directory="ZERO"

command=$(find . -type f -size 0 | cut -d "/" -f2;)

# Check if any files having size 0 exists?

if [[ $command ]]; then

# Files having size 0 bytes exists

if [ -d "$directory" ]; then

echo "Found directory $directory"

for i in $command; do

mv $i $directory && echo "moved $i to $directory/$i"

done

else

echo "$directory directory do not exist!!"

echo "----------------"

echo "Creating directory...."

mkdir $directory

echo "Moving files.."

for i in $command; do

mv $i $directory && echo "moved $i to $directory/$i"

done

fi

elif [ $? != 0 ] ; then

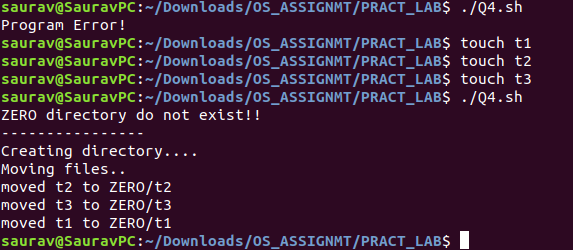
echo "Program Error!"

else

echo "No files having file size 0 could be found"

fi

**Output:**



**Q5. Write a program, that will accept a command line arg, which tells how many processes to create. In your code you need to create that many number of processes. Say you have entered 10 as the command line argument. So you need to create 10 processes (a loop should be there in the main program).Each child process should immediately exit with value say 0 to 9. The main program (the parent process) should wait for the termination of all 10 children and it should print the process identifier of the children and the value with which, the individual children exited.**

**Syntax:**

#include<stdio.h>

#include<stdlib.h>

#include<fcntl.h>

#include<errno.h>

#include<sys/types.h>

#include<unistd.h>

#include<sys/wait.h>

int main(int argc, char\* argv[])

{

int i,pid;

if(argc < 2)

{

printf ("Error in arguments\n");

exit(0);

}

for(i = 0; i < atoi(argv[1]); i++)

{

pid = fork();

if(pid < 0)

{

printf("Error");

exit(1);

}

else if (pid == 0)

{

printf("Child (%d): %d\n", i + 1, getpid());

exit(0);

}

else

{

wait(NULL);

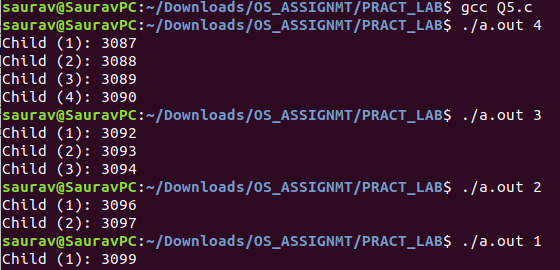
}

}

return 0;

}

**Output:**



Assignment 2

**Q1. Write a shell script that, that will traverse a directory recursively and will print the "absolute path names of all the regular files and their sizes present in the directory and all other subdirectories inside it".**

**Syntax:**

#!/bin/bash

if [[ $1 ]]; then # any thing other than Zero

cd $1 && find $PWD -type f

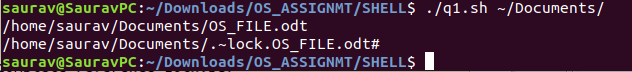
else

echo "Invalid Path Entered"

echo "Usage: ./filename path"

fi

**Output:**

****

**Q2. Write a shell script that will take a backup of all files in a directory and its subdirectories, that has been modified within the last 24 hours. The script will create a backup directory in the parent directory with name backup\_today's date, where it will copy all the files.**

**(The value 24 is for an example, you can use a command line parameter for this)**

**Syntax:**

#!/bin/bash

date=$(date +%F)

date="backup-$date"

echo "Creating Directory"

mkdir ../$date

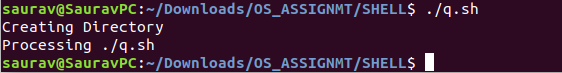
find -mmin -60 | grep -e "./" | while read -r line ;do

echo "Processing $line"

cp $line ../$date -rf

done

**Output:**



**Q3. Write a shell script that will check the top 10 CPU using processes. If any of these processes has a CPU usage more than 40%, the script will kill those processes.**

**Syntax:**

#!/bin/bash

top10process="$(ps -e -o %cpu -o %p -o comm | sort -r |

sed -n '1!p' | head -n 10)"

num1=40.0

cpu\_util="$(echo "$top10process" | awk '{print $1}')"

counter=1

delProcess=" "

for i in $cpu\_util; do

if (( $(echo "$i > $num1" |bc -l) )); then

echo "----------------------------------------------------"

echo "Found the following process with CPU utilization $i:"

echo "CPU% PID PName"

echo "$(echo "$top10process" | sed -n "$counter p")"

echo "Killing the found processes:

$(echo "$top10process" | sed -n "$counter p" | awk '{print $3}')"

delProcess=$(echo "$top10process" | sed -n "$counter p" | awk '{print $33

}')

sudo killall "$delProcess"

echo "----------------------------------------------------"

fi

let counter=counter+1

done

**Output:**

Q3(s).PNG

**Q4. Write a shell script that changes (means rename, myfile.doc needs to be renamed to myfile.txt) all .doc files to .txt files in your current directory and in all subdirectories.**

**(Use the two extension names as command line input)**

**Syntax:**

# Shell script to change file extensions.

#!/bin/bash

if [[ $1 && $2 ]]; then

for x in \*.$1; do

t=${x%.$1}.$2

mv $x $t && echo "changed $x -> $t"

done

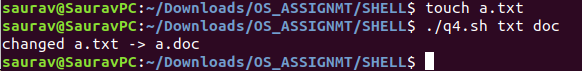
else

echo "Please enter args correctly"

echo "Usage ./filename ext1 ext2"

fi

**Output:**



**Q5. Write a shell script to find the total size of all files, for a particular user or all users (if no command line argument is supplied) (You may like to use the program that you wrote in Question 1. This is an useful problem for the administrator, who would like to see if some user has used much of the disk space)**

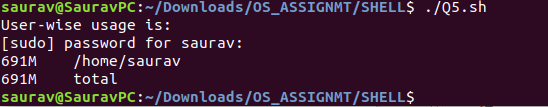
**Syntax:**

#!/bin/bash

echo "User-wise usage is:"

sudo du -shc /home/\* | sort

**Output:**



Assignment 3

**Q1. Write a shell script that changes (means rename, myfile.doc needs to be renamed to myfile.txt) all .doc files to .txt files in your current directory. Finally it should also print the number of renaming done as a summary.**

**(Use the two extension names as command line input)**

**Syntax:**

#!/bin/bash

num = 1

if [[ $1 && $2 ]];

then

for x in \*.$1;

do

t=${x%.$1}.$2

mv $x $t && echo "changed $x -> $t"

(( num++ ))

done

else

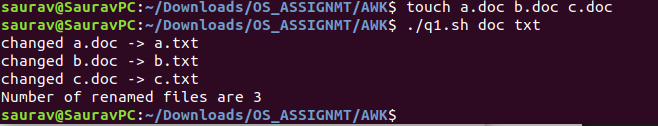
echo "Please enter args correctly"

echo "Usage ./filename ext1 ext2"

fi

echo "Number of renamed files are $num"

**Output:**



**Q2. Write a shell script to find the total size of all regular files in your current directory.**

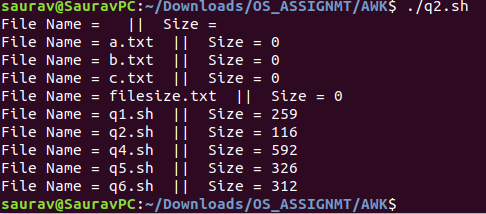
**Syntax:**

#!/bin/bash

ls -l>filesize.txt

awk '{print "File Name = " $9 " " " || " " Size = " $5 " "bytes}' filesize.txt

**Output:**



**Q4. Write a shell script that accepts a list of filenames as argument and reverses the lines (last line becomes the first line) in the files. Suppose you invoke the script as *myrev file1.txt file2.txt file3.txt,* the out put should be *file1.txt\_rev, file2.txt\_rev, file3.txt\_rev.* The new files file1.txt\_rev, file2.txt\_rev, file3.txt\_rev are to be created and they contain the lines reversed. (Do not use the tac command and write your own logic to reverse the lines)**

**Syntax:**

#!/bin/bash

if [ $# -eq 1 ]; #if passing only one file then print the data of that file then

then

cat $1

elif [ $# -eq 2 ]; #if passed two files then copy the contents of file1 into file 2 and print then

then

cp $1 $2

cat $2

else #copy the contents of all the files into last files and print

touch new

for var in $@

do

cat $var >> new

done

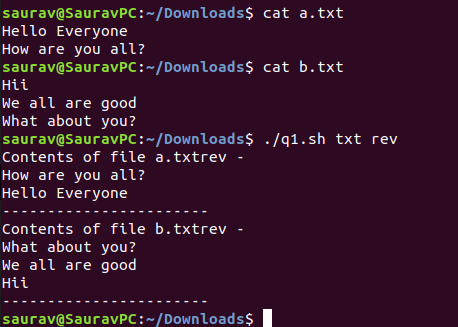
fi

echo "Content of NEW File :- "

awk '{a[i++]= $0 } END {for(j=i-1;j>=0;j--) print a[j];}' new

#One by one line is read and stored in an array. At the end, the array is printed in the reverse order by starting from the last index.

**Output:**



**Q5. Consider a text file named say "city", which contains a list of city names like "Delhi", "Kolkata", "Mumbai", "New York", "North Carolina" and so on. Please note that *each line is* containing one city name and city names can have space in beteen like "North Carolina". Write a shell script which will make another file say "my\_city", where you have lines like. "I would like to visit. New York".**

**It means you need to add "I would like to visit" before each city name in the new file. Do this only using shell constructs.**

**Syntax:**

#!/bin/bash

echo "Content of "city" file :-"

cat city.txt

touch my\_city

for var in $@

do

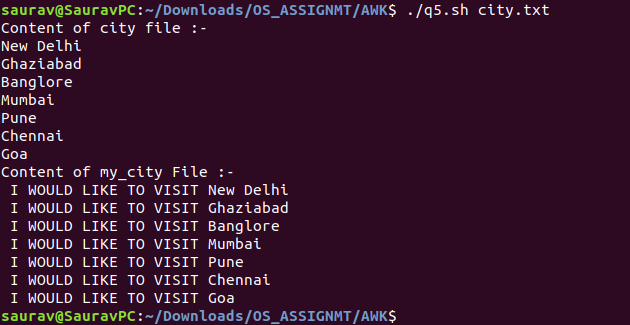
cat $var >> my\_city

done

echo "Content of my\_city File :- "

awk '{a[i++]= $0 } END {for(j=0;j<i;j++) print " I WOULD LIKE TO VISIT " a[j];}' my\_city

#One by one line is read and stored in an array. At the end, the array is printed.

**Output:**

**Q6. Write an awk script (do not use wc) to find the total number of *words, characters a and lines in a file.***

**Syntax:**

#!/bin/bash

BEGIN{print"Record\t\tCharacters\tWords\n"}

#BODY section

{

count=length($0)

print("-",NR,":\t\t",count,":\t\t",NF," ",$0)

words+=NF

Total\_count+=count

}

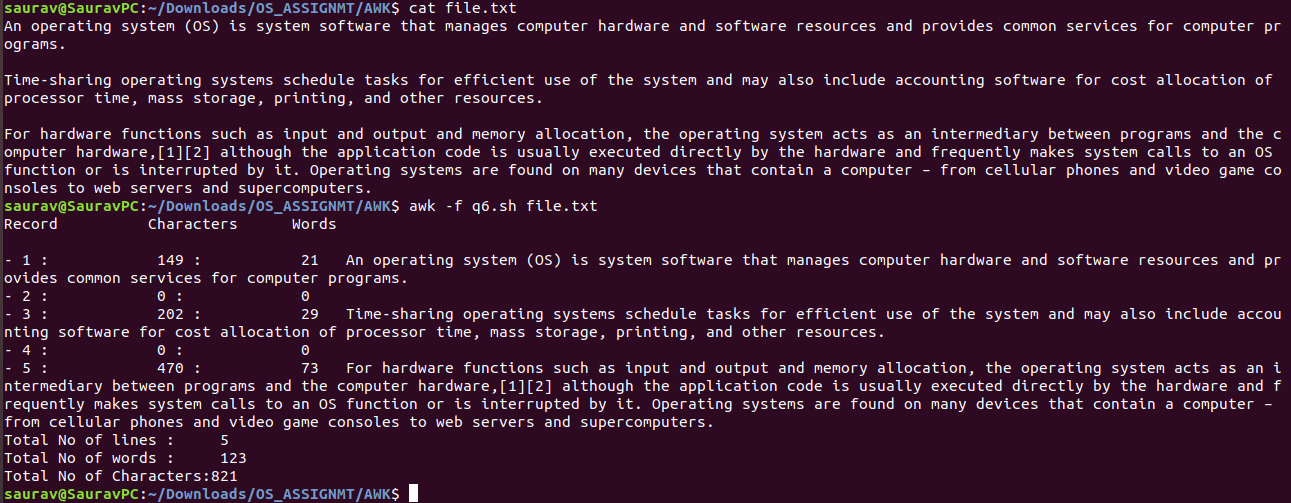
END{

print("Total No of lines :\t" NR)

print("Total No of words :\t" words)

print("Total No of Characters:" Total\_count)

}

 **Output:**