

Kulvir Singh

19BCE2074

CSE2005 – Operating Systems Lab

Assessment 1

a)Shell Programming

Handling Command Line Arguments

pwd COMMAND

This command prints the location of your current working directory.

ls COMMAND

ls is used to print contents of a directory, by default it lists contents of current working directory(PWD).

cd COMMAND

cd command is used to move one directory up.

cat COMMAND

It is used to print the contents of a file to the screen(stdout more precisely).

cp COMMAND

cp is used to copy files and directories .

mv COMMAND

mv command is used to move or rename directories and files.

rm COMMAND

rm command is used to remove directories or files.

mkdir COMMAND

mkdir is used to make a new directory in linux

rmdir COMMAND

rmdir is used to remove a directory.

ln COMMAND

This command is used to make link between files and directories.

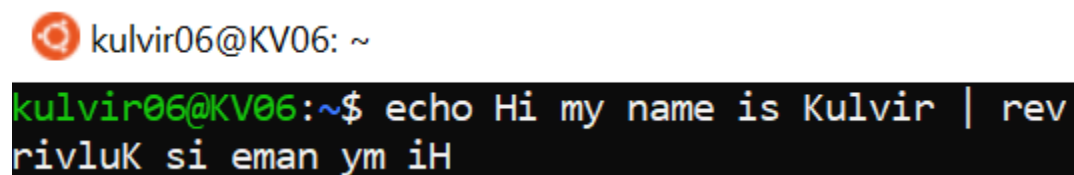
String Reversal

Code:

```
# string reversal
```

```
echo Hi my name is Kulvir | rev
```

Output:

A terminal window with a dark background. The prompt is 'kulvir06@KV06: ~'. The command 'echo Hi my name is Kulvir | rev' is entered, and the output 'rivluK si eman ym iH' is displayed.

```
kulvir06@KV06: ~$ echo Hi my name is Kulvir | rev  
rivluK si eman ym iH
```

If-Else, Nested If-Else, Switch Cases

Code:

```
#if else
```

```
a=6
```

```
b=77
```

```
if [ $a == $b ]
```

then


echo "a is equal to b"

else

echo "a is not equal to b"

fi

Output:

 kulvir06@KV06: ~

```
kulvir06@KV06:~$ a=6
kulvir06@KV06:~$ b=77
kulvir06@KV06:~$ if [ $a == $b ]
> then
> echo "a is equal to b"
> else
> echo "a is not equal to b"
> fi
a is not equal to b
kulvir06@KV06:~$
```

Code:

#if else nested

x=40

y=20

z=30

if [\$x > \$y]

then

if [\$x > \$z]

then

echo "x is the greatest"

else

```
    echo "z is the greatest"
fi
else
    if [ $y > $z ]
    then
        echo "y is the greatest"
    else
        echo "z is the greatest"
    fi
fi
```

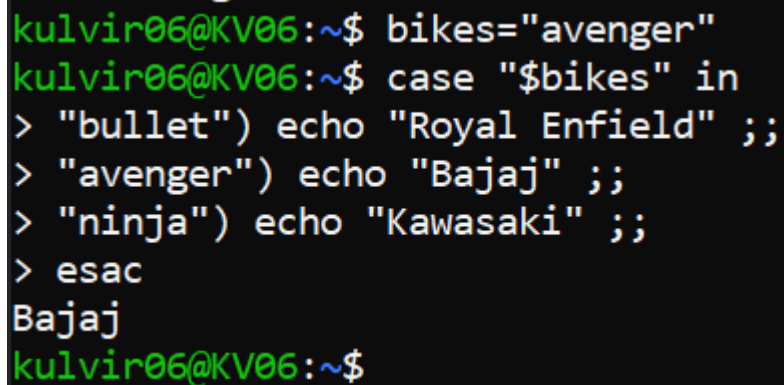
Output:

```
kulvir06@KV06:~$ x=40
kulvir06@KV06:~$ y=20
kulvir06@KV06:~$ z=30
kulvir06@KV06:~$ if [ $x > $y ]
> then
>   if [ $x > $z ]
>   then
>     echo "x is the greatest"
>   else
>     echo "z is the greatest"
>   fi
> else
>   if [ $y > $z ]
>   then
>     echo "y is the greatest"
>   else
>     echo "z is the greatest"
>   fi
> fi
x is the greatest
kulvir06@KV06:~$
```

Code:

```
#switch case  
bikes="avenger"  
case "$bikes" in  
    "bullet") echo "Royal Enfield" ;;  
    "avenger") echo "Bajaj" ;;  
    "ninja") echo "Kawasaki" ;;  
esac
```

Output:



```
kulvir06@KV06:~$ bikes="avenger"  
kulvir06@KV06:~$ case "$bikes" in  
> "bullet") echo "Royal Enfield" ;;  
> "avenger") echo "Bajaj" ;;  
> "ninja") echo "Kawasaki" ;;  
> esac  
Bajaj  
kulvir06@KV06:~$
```

b)Parent child process creation using fork() and exec() system call

Checking the process identifier

Code:

```
#include<stdio.h>  
#include<stdlib.h>  
#include<unistd.h>  
  
int main() {
```

```
printf("\n\n****OUTPUT****\n\n");

int pid, pid1, pid2;

pid = fork();

if( pid==0 ) {
    sleep(3);
    printf("child[1] --> process id = %d and parent process id = %d\n", getpid(), getppid());
}
else {
    pid1 = fork();

    if( pid1==0 ){
        sleep(2);
        printf("child[2] --> process id = %d and parent process id = %d\n", getpid(), getppid());
    }
    else {
        pid2 = fork();

        if( pid2==0 )
            printf("child[3] --> process id = %d and parent process id = %d\n", getpid(), getppid());
        else {
            sleep(5);
            printf("parent --> process id = %d\n", getpid());
        }
    }
}
}
```

```
}
```

Output:

```
****OUTPUT****

child[3] --> process id = 22529 and parent process id = 22526
child[2] --> process id = 22528 and parent process id = 22526
child[1] --> process id = 22527 and parent process id = 22526
parent --> process id = 22526

...Program finished with exit code 0
Press ENTER to exit console. □
```

Assigning new task to the child

Code:

```
#include<stdio.h>

#include<sys/types.h>

void ChildProcess();

void ParentProcess();

int main() {

    printf("\n\n****OUTPUT****\n\n");

    pid_t pid;

    pid = fork();

    if( pid==0 ) { ChildProcess(); }

    else { ParentProcess(); }
```

```
}  
  
void ChildProcess() { printf("This is the Child Process\n"); }  
  
void ParentProcess() { printf("This is the Parent Process\n"); }
```

Output:

```
****OUTPUT****  
  
This is the Parent Process  
This is the Child Process  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Providing the pathname and program name to exec()

Code:

```
#include<stdio.h>  
  
#include<stdlib.h>  
  
#include<unistd.h>  
  
int main()  
{  
    printf("\n\n****OUTPUT****\n\n");  
    char *args[]={"/EXEC",NULL};  
    execv(args[0],args);  
    printf("Ending-----");  
    return 0;  
}
```


Output:

```
****OUTPUT****

Ending-----

...Program finished with exit code 0
Press ENTER to exit console.
```

Synchronizing parent and child process using wait

Code:

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

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

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

```
int main()
```

```
{
```

```
    printf("\n\n****OUTPUT****\n\n");
```

```
    if ( fork() == 0 )
```

```
        printf("Child Process: hello from child\n");
```

```
    else
```

```
    {
```

```
        printf("Parent Process: hello from parent\n");
```

```
        printf("***Waiting for Child Process to execute***\n");
```

```
        wait(NULL);
```

```
    printf("Child Process: child has terminated\n");  
}  
return 0;  
}
```

Output:

```
****OUTPUT****  
  
Parent Process: hello from parent  
***Waiting for Child Process to execute***  
Child Process: hello from child  
Child Process: child has terminated  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

c)Process and Thread Management

Code:

```
#include <stdio.h>  
  
#include <string.h>  
  
#include <pthread.h>  
  
int i = 6;  
  
void* Thread( void* p ) {  
    printf("Value received as argument in starting routine: ");
```

```

    printf("%d\n", * (int*)p);
    return &i;
}

int main() {

    pthread_t id;
    int j = 1;
    pthread_create(&id, NULL, Thread, &j);

    int* ptr;

    pthread_join(id, (void**)&ptr);
    printf("Value received by parent from child: ");
    printf("%d\n", *ptr);
}

```

Output:

```

Value received as argument in starting routine: 1
Value received by parent from child: 6

...Program finished with exit code 0
Press ENTER to exit console.

```

d) Write a program to create a thread to find the factorial of a natural number 'n'.

Code:

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

```
int factorial = 1;
```

```
void* Thread( void* p ) {
    int x = * (int*)p;
    for( int i=1; i<=x; i++ ) {
        if( x%i==0 ) {
            factorial *= i;
        }
    }
    return &factorial;
}
```

```
int main() {
    int num;
    printf("enter a number\n");
    scanf("%d",&num);

    pthread_t id;
    pthread_create(&id, NULL, Thread, &num);

    int* ptr;
    pthread_join(id, (void**)&ptr);
    printf("Factorial = %d",*ptr);
}
```

```
    return 0;
}
```

Output:

```
enter a number
6
Factorial = 36

...Program finished with exit code 0
Press ENTER to exit console.
```

e) Assume that two processes named client and server running in the system. It is required that these two processes should communicate with each other using shared memory concept. The server writes alphabets from a..z to the shared memory .the client should read the alphabets from the shared memory and convert it to A...Z. Write a program to demonstrate the above mentioned scenario.

Code:

```
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
#include<sys/types.h>

void Client();
void Server();

int ch[26];
int main() {
    printf("\n\n****OUTPUT****\n\n");
    pid_t pid;
    pid = fork();
    if( pid==0 ) { Client(); }
    else {
```

```

        printf("Data stored by server in memory ----->\n");
        Server();
        printf("Client accessing the same memory location\n");
        wait(NULL);
    }
}

void Server() {
    int k = 97;
    for(int i=0;i<26;i++) {
        ch[i] = k;
        k++;
        printf("%c",ch[i]);
    }
    printf("\n");
}

void Client() {
    printf("Data modified by client----->\n");
    int k=65;
    for(int i=0;i<26;i++) {
        ch[i] = k;
        k++;
        printf("%c",ch[i]);
    }
    printf("\n");
}

```

Output:

```
****OUTPUT****
```

```
Data stored by server in memory ----->
```

```
abcdefghijklmnopqrstuvwxyz
```

```
Client accessing the same memory location
```

```
Data modified by client----->
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
...Program finished with exit code 0
```

```
Press ENTER to exit console.█
```

f) The Collatz conjecture concerns what happens when we take any positive integer n and apply the following algorithm: $n = n/2$, if n is even $n = 3 \times n + 1$, if n is odd the conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1. For example, if $n = 35$, the sequence is 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1. Write a C program using the `fork()` system call that generates this sequence in the child process. The starting number will be provided from the command line. For example, if 8 is passed as a parameter on the Command line, the child process will output 8, 4, 2, 1. Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence. Have the parent invoke the `wait()` call to wait for the child process to complete before exiting the program

Code:

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

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

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

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

```

void ChildProcess();

int main( int args ) {
    printf("\n\n****OUTPUT****\n\n");
    pid_t pid;
    pid = fork();
    if( pid==0 ) { ChildProcess( args ); }
    else {
        printf("Waiting for child process to complete\n");
        wait(NULL);
    }
}

void ChildProcess( int n ) {
    while( n!=1 ) {
        if( n%2 == 0 ) {
            printf("%d ", n);
            n = n/2;
        }
        else {
            printf("%d ", n);
            n = 3*n + 1;
        }
    }
    printf("1\n");
}

```

Output:


```
****OUTPUT****

Waiting for child process to complete
8 4 2 1

...Program finished with exit code 0
Press ENTER to exit console.□
```

g) Write a multithreaded program that calculates various statistical values for a list of numbers. This program will be passed a series of numbers on the command line and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value. For example, suppose your program is passed the integers 90 81 78 95 79 72 85 , the program will report the average value as 82. The minimum value as 72. The maximum value as 95. The variables representing the average, minimum, and maximum values will be stored globally. The worker threads will set these values, and the parent thread will output the values once the workers have exited.

Code:

```
#include <stdio.h>

#include <string.h>
#include <pthread.h>

int avg, max, min;

void* avgThread( int p[] ) {
    int length = sizeof(p);
    int sum = 0;
    for(int i=0; i<length; i++)
        sum += p[i];
    avg = sum/(length+1);
```

```

    return &avg;
}

void* maxThread( int p[] ) {
    int length = sizeof(p);
    for(int i=0; i<length; i++)
        if(p[i]>max)
            max = p[i];
    return &max;
}

void* minThread( int p[] ) {
    min = p[0];
    int length = sizeof(p);
    for(int i=0; i<length; i++)
        if(p[i]<min)
            min = p[i];
    return &min;
}

int main(int args[]) {
    pthread_t id,id1,id2;

    pthread_create(&id, NULL, avgThread, &args);
    pthread_create(&id1, NULL, maxThread, &args);
    pthread_create(&id2, NULL, minThread, &args);

    int* ptr;
    int* ptr1;
    int* ptr2;

    pthread_join(id, (void**)&ptr);
    pthread_join(id1, (void**)&ptr1);
    pthread_join(id2, (void**)&ptr2);

    printf("Average = %d\n", *ptr);
    printf("Maximum = %d\n", *ptr1);
    printf("Minimum = %d\n", *ptr2);
}

```

Output:

Average = 82

Maximum = 95

Minimum = 72

...Program finished with exit code 0

Press ENTER to exit console.