

BCSE303P- Operating Systems Lab

ASSESSMENT-2

File operations and fork() system calls

Name: Jefferson David Kingston

Register Number: 21bce1882

Lab Slot: L15+L16

Faculty:Dr. Anandan P

1. Develop a C program to get the input from standard input device and print the same on the standard output device.

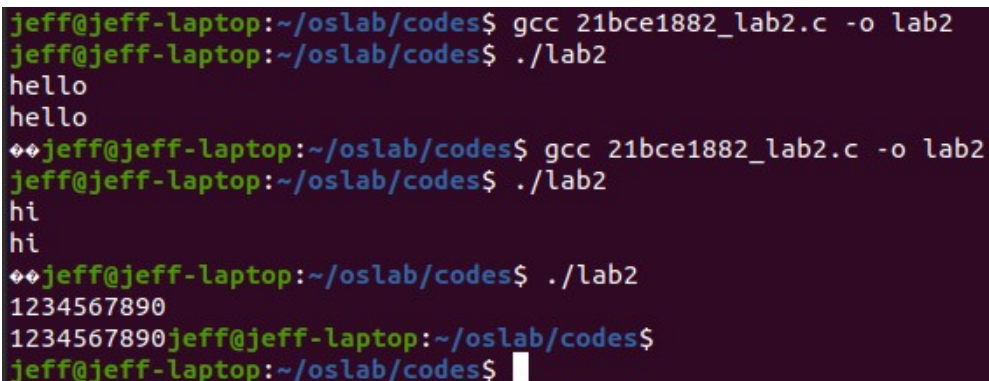
Code:

```
#include <stdio.h>

#include <unistd.h>

int main(){
    int count,count1;
    char buff[20];
    count = read(0,buff,10);
    count=write(1,buff,10);
}
```

output:



```
jeff@jeff-laptop:~/oslab/codes$ gcc 21bce1882_lab2.c -o lab2
jeff@jeff-laptop:~/oslab/codes$ ./lab2
hello
hello
♦♦jeff@jeff-laptop:~/oslab/codes$ gcc 21bce1882_lab2.c -o lab2
jeff@jeff-laptop:~/oslab/codes$ ./lab2
hi
hi
♦♦jeff@jeff-laptop:~/oslab/codes$ ./lab2
1234567890
1234567890jeff@jeff-laptop:~/oslab/codes$
jeff@jeff-laptop:~/oslab/codes$
```

2. Develop a C program to perform the following file operations – Create , Write, Read, Copy.

Code:

```
#include <unistd.h>

#include <stdio.h>

#include <sys/types.h>

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

```
#include <fcntl.h>

#include <string.h>


int main(){
char buff[11];
int fd= open("foo.txt",O_WRONLY|O_CREAT);
int c= write(fd,"Hello-World",11);
fd=open("foo.txt",O_RDONLY);
read(fd,buff,c);
printf("Message in the file foo.txt is :%s\n",buff);
close(fd);
fd=open("foo.txt",O_RDONLY);
read(fd,buff,c);
close(fd);
int fd1=open("foo1.txt",O_WRONLY|O_CREAT);
write(fd1,buff,c);
close(fd1);
fd=open("foo.txt",O_RDONLY);
read(fd,buff,c);
printf("Message written in new file: %s",buff);
}
```

Output:

```
jeff@jeff-laptop:~$ gcc copy.c -o cop
jeff@jeff-laptop:~$ ./cop
Message in the file foo.txt is :Hello-World
Message written in new file: Hello-Worldjeff@jeff-laptop:~$ echo 21bce18882
21bce18882
jeff@jeff-laptop:~$ █
```

3.

Develop a C program to create four child processes. Print the parent ID in all child processes along with its own ID. Print all Child IDs in parent process along with its own ID.

Code:

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

int main()
{
printf("21bce1882-Jefferson\n");
for(int i=0;i<4;i++)
{

if(fork() == 0)
{
printf("Child pid %d from Parent pid %d\n",getpid(),getppid());
exit(0);
}

}
```

```
}  
}
```

output:

```
jeff@jeff-laptop:~/oslab/codes$ ./chi  
21bce1882-Jefferson  
Child pid 6470 from Parent pid 6469  
Child pid 6471 from Parent pid 6469  
Child pid 6472 from Parent pid 6469  
Child pid 6473 from Parent pid 6469  
jeff@jeff-laptop:~/oslab/codes$
```

4.

Develop a C program to print even numbers between 0 and 10 within Child process & print odd numbers between 0 and 10 within Parent process.

Code:

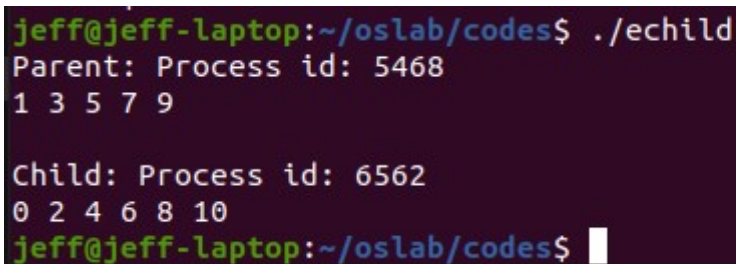
```
#include<stdio.h>  
#include<unistd.h>  
#include<sys/types.h>  
#include<stdio.h>  
int main()  
{  
int pid;  
pid=fork();  
if(pid==0)  
{  
printf("Child: Process id: %d\n",getpid());  
for(int i=0;i<=10;i++)  
{  
(i%2==0)?printf("%d ",i):printf("");  
}
```

```

printf("\n");
}
else if(pid>0)
{
printf("Parent: Process id: %d\n",getppid());
for(int i=0;i<=10;i++)
{
(i%2!=0)?printf("%d ",i):printf("");
}
printf("\n\n");
}
return 0;
}

```

Output:



```

jeff@jeff-laptop:~/oslab/codes$ ./echild
Parent: Process id: 5468
1 3 5 7 9

Child: Process id: 6562
0 2 4 6 8 10
jeff@jeff-laptop:~/oslab/codes$

```

5. Develop a C program to perform 2x2 matrix addition in child process and determine whether the given number is prime or not in parent process.

Code:

```

#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdio.h>
void parent()
{

```

```

int flag=1,n;
scanf("%d",&n);
for(int i=1;i<n/2;i++)
{
if(n%i==0)
{
flag=0;
break;
}
}

(flag==1)?printf("The number given is a prime number\n"):printf("The number given
is not a prime number\n");
}

void child()
{
int a[2][2],b[2][2],sum[2][2];
for(int i=0;i<2;i++)
{
for(int j=0;j<2;j++)
{
scanf("%d",&a[i][j]);
}
}

for(int i=0;i<2;i++)
{
for(int j=0;j<2;j++)
{
scanf("%d",&b[i][j]);
}
}

```

```
}  
for(int i=0;i<2;i++)  
{  
for(int j=0;j<2;j++)  
{  
sum[i][j]=a[i][j]+b[i][j];  
}  
}  
for(int i=0;i<2;i++)  
{  
for(int j=0;j<2;j++)  
{printf("%d\t",sum[i][j]);  
}  
printf("\n");  
}  
}  
int main()  
{  
int pid;  
pid=fork();  
if(pid==0)  
{  
child();  
}  
if(pid>0)  
{  
parent();  
printf("Parent: Process id: %d\n\n",getppid());
```

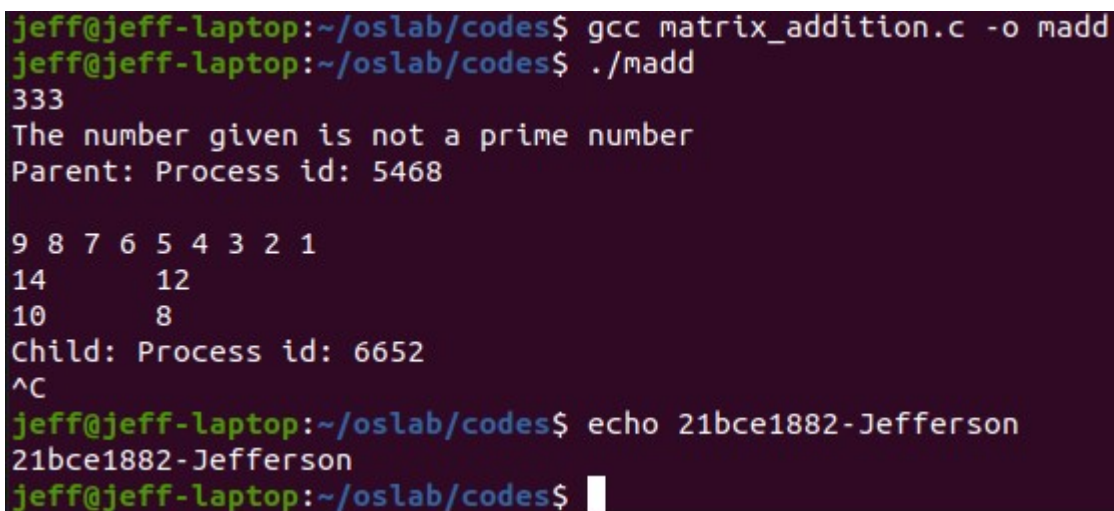


```

}else{
printf("Child: Process id: %d\n",getpid());
}
if(pid>0)
{
parent();
printf("Parent: Process id: %d\n\n",getppid());
}
return 0;}

```

output:



```

jeff@jeff-laptop:~/oslab/codes$ gcc matrix_addition.c -o madd
jeff@jeff-laptop:~/oslab/codes$ ./madd
333
The number given is not a prime number
Parent: Process id: 5468

9 8 7 6 5 4 3 2 1
14      12
10      8
Child: Process id: 6652
^C
jeff@jeff-laptop:~/oslab/codes$ echo 21bce1882-Jefferson
21bce1882-Jefferson
jeff@jeff-laptop:~/oslab/codes$ █

```

6. Develop a C program to create a process hierarchy as shown below.

Code:

```

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main() {
int P1,P2,P3,P4,P5;
int pid1=fork();
if(pid1==0)

```

```
{
printf("P2 of child process id %d under %d\n",getpid(),getppid());
}
else
{
int pid2=fork();
if(pid2==0)
{
int pid3=fork();
if(pid3==0)
{
printf("P4 of child process id %d under %d\n",getpid(),getppid());
}
else
{
int pid4=fork();
if(pid4==0)
{
printf("P5 of child process id %d under %d\n",getpid(),getppid());
}
else
{
printf("P3 of child process id %d under %d\n",getpid(),getppid());
}
}
}
else
{
printf("P1 parent process id :%d\n",getpid());
}
}
return 0;
```

}

Output:

```
jeff@jeff-laptop:~/oslab/codes$ gcc process_hierarchy.c -o ph
jeff@jeff-laptop:~/oslab/codes$ ./ph
P1 parent process id :6779
P2 of child process id 6780 under 6779
P4 of child process id 6782 under 6781
P5 of child process id 6783 under 6781
P3 of child process id 6781 under 1510
jeff@jeff-laptop:~/oslab/codes$ echo 21bce1882-Jefferson
21bce1882-Jefferson
jeff@jeff-laptop:~/oslab/codes$
```