Laboratory Assignments 4
Subject: Design of Operating Systems
Subject code: CSE 4049

Assignment 4: Familiarization with Process Management in Unix environment.

Objective of this Assignment:

- To trace the different states of a process during its execution.
- To learn the use of different system calls such as (fork(),vfork(),wait(),execl()) for process handling in Unix environment.

1. Write a C program to create a child process using fork() system call. The child process will print the message "Child" with its process identifier and then continue in an indefinite loop. The parent process will print the message "Parent" with its process identifier and then continue in an indefinite loop.
   a) Run the program and trace the state of both processes.
   b) Terminate the child process. Then trace the state of processes.
   c) Run the program and trace the state of both processes. Terminate the parent process. Then trace the state of processes.
   d) Modify the program so that the parent process after displaying the message will wait for child process to complete its task. Again run the program and trace the state of both processes.
   e) Terminate the child process. Then trace the state of processes.
2. Trace the output of the following codes:

a)
int main( )
{
if(fork()==0)
printf("1");
else
printf("2");
printf("3");
return 0;
}
	b)
	int main( )
	{
	if(vfork()==0)
	{
	printf("1");
	_exit(0);
	}
	else
	printf("2");
	printf("3");
	}

c)
```
int main( )
{
 pid_t pid;
 int i=5;
 pid=fork();
 i=i+1;
if(pid= =0)
{
     printf("Child: %d",i);
 }
 else

 {
     wait(NULL);
     printf("Parent: %d",i);
  }
  return 0;
}
```

d)
```
int main( )
{
 pid_t pid;
 int i=5;
 pid=vfork();
 i=i+1;
 if(pid==0)
 {
     printf("Child: %d",i);
     _exit(0);
 }

 else
{
     printf("Parent: %d",i);
 }
 return 0;
}
```

e)
```
int main( )
{
 pid_t pid;
 int i=5;
 pid=fork();
 if(pid= =0)
 {
    i=i+1;
    printf("Child: %d",i);
 }
  else
 {
   wait(NULL);
   printf("Parent: %d",i);
  }
  return 0;
 }
```

f)
```
int main( )
{
  pid_t pid;
  int i=5;
   pid=vfork();
   if(pid==0)
   {
      i=i+1;
      printf("Child: %d",i);
      _exit(0);
   }
   else
   {
      printf("Parent: %d",i);
   }
   return 0;
  }
```

g)
```
int main( )
{
 int i=5;
 if(fork( )==0)
 {
      printf("Child: %d",i);
 }
 else
 {
     printf("Parent: %d",i);
 }
 return 0;
}
```

h)
```
int main( )
{
 int i=5;
 if(vfork( )==0)
 {
     printf("Child: %d",i);
     _exit(0);
 }
 else
 {
     printf("Parent: %d",i);
 }
  return 0;
}
```

i)
```
int main( )
{
 if(fork( )==0)
 {
      printf("1");
  }
  else
  {
       wait(NULL);
       printf("2");
       printf("3");
    }
    return 0;
  }
```

j)
```
int main( )
{
 if(vfork( )==0)
 {
     printf("1");
     _exit(0);
 }
  else
 {
     printf("2");
    printf("3");
  }
  return 0;
 }
```

k)
```
int main( )
{
pid_t c1;
int n=10;
c1=fork( );
if(c1==0)
{
      printf(" Child\n");
      n=20;
      printf("n=%d \n",n);
 }
 else
{
       wait(NULL);
       printf("Parent\n");
       printf("n=%d \n",n);
  }
   return 0;
}
```

l)
```
int main( )
{
  pid_t c1;
  int n=10;
  c1=vfork( );
  if(c1==0)
  {
        printf(" Child\n");
        n=20;
        printf("n=%d \n",n);
        _exit(0);
   }
   else
   {
         printf("Parent\n");
         printf("n=%d \n",n);
   }
    return 0;
  }
```

m)
```
int main( )
{
 int i=5;
 fork();
 i=i+1;
 fork();
 fprintf ( stderr,"% d",i);
 return 0;
}
```

n)
```
int main( )
{
pid_t pid;
int i=5;
pid=vfork();
if(pid==0)
{
      printf("Child: %d",i);
      _exit(0);
 }
  else
  {
        i=i+1;
        printf("Parent: %d",i);
   }
   return 0;
```

	}

o)
int main( )
{
int i=5;
if(fork()==0)
i=i+1;
else
i=i-1;
fprintf(stderr,"%d",i);
return 0;
}
	p)
	int main( )
	{
	int i=5;
	if(vfork()==0)
	{
	i=i+1;
	_exit(0);
	}
	else
	i=i-1;
	fprintf(stderr,"%d",i);
	return 0;
	}

q)
int main( )
{
int j,i=5;
for(j=1;j<3;j++)
{
if(fork()==0)
{
i=i+1;
break;
}
else
wait(NULL);
}
fprintf(stderr,"%d",i);
return 0;
}
	r)
	int main( )
	{
	int j,i=5;
	for(j=1;j<3;j++)
	{
	if(fork()!=0)
	{
	i=i-1;
	break;
	}
	}
	fprintf(stderr,"%d",i);
	return 0;
	}

s)
int main( )
{
if(fork() == 0)
if(fork())
printf("1\n");
return 0;
}
	t)
	void fun1(){
	fork();
	fork();
	printf("1\n");
	}
	int main() {
	fun1();
	printf("1\n");
	return 0;
	}

3. Write a C program that will create three child process to perform the following operations respectively:
    - First child will copy the content of file1 to file2
    - Second child will display the content of file2
    - Third child will display the sorted content of file2 in reverse order.
    - Each child process being created will display its id and its parent process id with appropriate message.
    - The parent process will be delayed for 1 second after creation of each child process. It will display appropriate message with its id after completion of all the child processes.

4. Write a C program that will create a child process to generate a Fibonacci series of specified length and store it in an array. The parent process will wait for the child to complete its task and then display the Fibonacci series and then display the prime Fibonacci number in the series along with its position with appropriate message.