(slips-1,5,13)—1. Write a program that demonstrates the use of nice () system call. After a child process is

// started using fork (), assign higher priority to the child using nice () system call.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/time.h>
int main()
{
   int child_id=fork();
   int nice_value=-10;
   if(child_id==0)
   {
      printf("child process id %d is running…\n",getpid());
      if(nice(nice_value)==-1)
      {
         perror("nice faild");
         Exit(1);
      }
   }
   Else
   {
      printf("parent process id %d is running..\n",getpid());
      sleep(2);
      printf("parent process id %d is finished…\n",getpid());
   }
   Return 0;
}
```

//(slips-3) Q. 1 Creating a child process using the command exec(). Note down process ids of the parent // and the child processes, check whether the control is given back to the parent after the child // process terminates.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main()
{
   int child_id=fork();
   if(child_id==0)
   {
      printf("child process id:%d\n",getpid());
      printf("child process: Executed 'ls'command \n");
      execlp("ls",(char*)NULL);
      perror("execlp()");
      exit(1);
   }
   else
   {
      printf("parent process id: %d\n",getpid());
      printf("parent process: waiting for child process to finish\n");
      wait(NULL);
      printf("parent process: child has finished\n");
   }
   return 0;
}
```

(slips-2,11) Q.1 Create a child process using fork(), display parent and child process id. Child process will

// display the message “Hello World” and the parent process should display “Hi”.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
   int child_pid=fork();
   if(child_pid==0)
   {
      printf(“child process pid:%d\n”,getpid());
      printf(“Hello World\n”);
   }
   else if(child_pid>0)
   {
      printf(“parent process pid:%d\n”,getpid());
      printf(“Hi\n”);
   }
   else
   {
      perror(“fork”);
      exit(EXIT_FAILURE);
   }
   return 0;
}
```

(slips-6,14,16) Q.1 Write a program to find the execution time taken for execution of a given set of instructions

// (use clock() function)

```
#include<stdio.h>
#include<time.h>
int main()
{
   int start_time=clock();

   for(int i=0;i<1000000;i++)
   {
      //instruciton
   }
   int end_time=clock();
   double execution_time=(double)(end_time-start_time)/CLOCKS_PER_SEC;
   printf("Execution time: %lf seconds\n",execution_time);
   return 0;
}
```

(slips-4,10,12) Q.1 Write a program to illustrate the concept of orphan process (Using fork() and sleep()

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
   int child_id=fork();

   if(child_id==0)
   {
      printf("child process pid %d is running…\n",getpid());
      sleep(5);
      printf("child procees pid %d has finished…\n",getpid());
   }
   else
   {
      printf("parent process pid %d is running…\n",getpid());
      sleep(2);
      printf("parent process pid %d is finished…\n",getpid());
   }
   return 0;
}
```

(slips- 7, 9, 15, 19, 20) Q.1 Write a program to create a child process using fork().The parent should goto sleep state and // child process should begin its execution. In the child process, use execl() to execute the "ls"// command.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main()
{
   int child_id=fork();
   if(child_id==0)
   {
      printf("Child process is executed 'ls' cmd…\n");
      execl("/bin/ls","ls",NULL);
      perror("execl");
      exit(1);
   }
   else
   {
      printf("parent process in sleep mode…\n");
      sleep(2);
      printf("parent process wake up in sleep \n");
   }
   return 0;
}
```