Unnamed Pipes:

Header files used :

#include<stdio.h> which is most commonly used header file in C program

#include<stdlib.h> which is used for process control and memory allocation

#include<unistd.h> is used for POSIX operating system API for function definitions like fork()

#include<sys/wait.h> and #include<errno.h> is used for pipes

#include<math.h> used for square root function

System calls used:

fork() : Fork system call is used for creating a new process, which is called child process with same PID which runs concurrently with the process that makes the fork() call (the parent process).

pipe(): The pipe() system call is used to open file descriptors, which are used to communicate between different Linux processes. It is used for inter-process communication(ipc) in linux.

read(): It attempts to read up to count bytes from file descriptor into the buffer starting at buf

write(): It writes up to count bytes from the buffer starting at buf to the file descriptor fd.

close(): It closes the file descriptor, so that it no longer refers to any file and may be reused. Any record locks held on the file it was associated with, and owned by the process, are removed.

Functions used:

Standard_deviation :

This function calculated standard deviation which is square of sum of squares of deviations from the average(mean).

Commands:

$ gcc -o Q1 Q1.c -lm

$ ./Q1

Here -lm is to compile as per the specification in the math function squareroot(sqrt)

Named Pipes:

Header files used :

#include<stdio.h> which is most commonly used header file in C program

#include<stdlib.h> which is used for process control and memory allocation

#include<sys/ipc.h> is used for messages and shared memory. Here for some of functions like key_t this is used.

#include<sys/msg.h> is the header file containing many system calls for message like mkfifo, msgsnd, msgrcv etc.,

#include<math.h> used for square root function

System calls used:

mkfifo(): Creates a new FIFO special file named by the pathname pointed to by path. The file permission bits of the new FIFO shall be initialized from mode.(Basically Pipe is created using this)

ftok(): It is used to generate a unique key.

msgget(): either returns the message queue identifier for a newly created message queue or returns the identifiers for a queue which exists with the same key value.

msgsnd(): Data is placed on to a message queue by calling msgsnd().

msgrcv(): messages are retrieved from a queue.

msgctl(): It performs various operations on a queue. Generally it is use to destroy message queue.

Commands:

$ mkfifo pipe1

$mkfifo pipe2

$mkfifo pipe3

(This is for creating three pipes for communicating between processes.)

$gcc -o process1 process1.c

$./process1

$gcc -o process2 process2.c -lm

$./process2

(Here -lm is used for supporting math function square root sqrt)

$gcc -o process3 process3.c

$./process3

Shared Memory:

Header files used:

#include<stdio.h> which is most commonly used header file in C program

#include<stdlib.h> which is used for process control and memory allocation

#include<sys/ipc.h> is used for messages and shared memory. Here for some of functions like key_t this is used.

#include<sys/shm.h> is used for system calls responsible for shared memory.

System Calls used:

ftok(): It is used to generate a unique key.

shmget(): **int shmget(key_t,size_tsize,intshmflg);**

Upon successful completion, shmget() returns an identifier for the shared memory segment.

shmat(): Before you can use a shared memory segment, you have to attach yourself to it using shmat().

void *shmat(int shmid ,void *shmaddr ,int shmflg);
shmid is shared memory id.
shmaddr specifies specific address to use but we should set it to zero and OS will automatically choose the address.

shmdt(): When you're done with the shared memory segment, your program should detach itself from it using shmdt().
int shmdt(void *shmaddr);

shmctl(): When you detach from shared memory, it is not destroyed. So, to destroy shmctl() is used.
shmctl(int shmid,IPC_RMID,NULL);

Commands :

$ gcc -o process1 process1.c

$./process1

$gcc -o process2 process2.c

$./process2

$gcc -o process3 process3.c

$./process3