1. The below given code demonstrate to observe the list of descriptors opened for a process. The given code conatins a **scanf(``%d'',&var)** before the last **return 0;** statement. Compile and run the code, but do not supply any value for the **scanf()**. At this point the code will be a blocking read. Now open a new terminal and run the command **ls /proc/PID/fd**, where PID is the process ID of your process running and has printed on the blocking read terminal.

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>
#include<errno.h>
int main()
{
 int fd,var;
 printf("PID=%ld\n",(long)getpid());
 fd=open("read.c",O_RDONLY);
 fd=open("read.c",O_RDONLY);
 scanf("%d",&var);
 return 0;
}
```

PROGRAM:-

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>
#include<errno.h>
int main() {
int fd, var;
printf("PID=%1d\n"(long)getpid());
fd=open("read.c"O_RDONLY);
fd-open ("read.c",O_RDONLY);
scanf("%d",&var);
return 0;
}
```

OUTPUT:-

```
coldwater@sushovan:~/Desktop/UNP/1941012580/Assignment6$ gcc A6Q1.c
coldwater@sushovan:~/Desktop/UNP/1941012580/Assignment6$ ./a.out
PID=2229
```

2. Check out the list of file descriptors are opened for the following code snippet:

```
int main(){
 int fd;
 FILE *myfp,*fp;
 printf("PID=%ld\n",(long)getpid());
 myfp=fopen("T1.dat","w");
 fp=fopen("T2.txt","w");
 if(myfp==NULL){
   return 1;
 }
 if(fp==NULL){
   return 2;
 }
 fd=open("T3.c",O_RDONLY);
 fd=open("T4.c",O_RDONLY);
 while(1);
 return 0;
}
```

PROGRAM:-

```
#include<stdio.h>
#include<unistd.hy
#include<fentl.h>
#include<sys/types.h›
#include<errno.h›
int main {
int fd;
FILE *myfp, *fp;
brintf("PID=%ld\n(long) getpid());
```

```
myfp=fopen("T1.dat,"w");
fp=fopen ("T2. txt", "w");
if (myfp==NULL){
return 1;
}
}
```

OUTPUT:-

```
coldwater@sushovan:~/Desktop/UNP/1941012580/Assignment6$ gcc A6Q2.c
coldwater@sushovan:~/Desktop/UNP/1941012580/Assignment6$ ./a.out
PID=3730
```

3. Lets us consider the below code segment to open a file using **file pointer**.

```
FILE *myfp;
myfp=fopen("Test.dat","w");
if(myfp==NULL){
    return 1;
}
fprintd(myfp,"File pointer is a handle to handle");
```

The **FILE** structure is allocated by **fopen** function call. The **FILE** structure contains a buffer and a file descriptor (*Refer page number 122, section 4.6.2 of the **USP** text book for schematic diagram*). Run the below code to display the **file descriptor** created internally because of the file pointer to perform IO. In some sense the file pointer is a handle to a handle.

```
#include<stdio.h>
int main()
{
    FILE *myfp;
    int fd;
    myfp=fopen("Trial.txt","w");
    if(myfp==NULL){
        perror("Opening Error");
        return 1;
    }
    /* To get the file descriptor value */

    fd=fileno(myfp);    /* fileno() is a library function */

    printf("File descriptor=%d\n",fd);
    return 0;
}
```

PROGRAM:-

```
#include<stdio.h>
#include<unistd.h>
#include<fentl.h>
#include<sys/types.h>
#include<errno.h>
int main(){
FILE *myfp;
int fd;
myfp=fopen("Trial.txt","w");
if(myfp==NULL){
perror ("Opening Error") ;
return 1;
}
fd-fileno (myfp);
print ("File descriptor=%d\n",fd);
return 0;
}
```

OUTPUT:-

```
coldwater@sushovan:~/Desktop/UNP/1941012580/Assignment6$ gcc A6Q3.c
coldwater@sushovan:~/Desktop/UNP/1941012580/Assignment6$ ./a.out
PID=609
```

4. Findout the output of the given code snippet:

```
int main()
{
 printf("stdin file descriptor No.: %d\n",STDIN_FILENO);
 printf("stdout file descriptor No.: %d\n",STDIN_FILENO);
 printf("stderr file descriptor No.: %d\n",STDIN_FILENO);
 printf("Standard file descriptors using FILE pointers:\n");
 printf("stdin file descriptor No.: %d\n",fileno(stdin));
 printf("stdout file descriptor No.: %d\n",fileno(stdout));
 printf("stderr file descriptor No.: %d\n",fileno(stderr));
 return 0;
}
```

PROGRAM:-

```
#include<stdio.h>
#include<unistd.h>
#include<fentl.h>
#include<sys/types.h>
#include<errno.h>
int main(){
printf("stdin file descriptor No.: %d\n", STDIN FILENO);
printf("stdout file descriptor No.: %d\n", STDIN FILENO);
printf("stderr file descriptor No.: %a \n",STDIN FILENO);
printf("Standard file descriptors using FILE pointers: \n");
printf("stdin file descriptor No.: %d\n",fileno(stdin));
printf("stdout file descriptor No.: %d\n", fileno(stdout));
printf("stderr file descriptor No.: %\n", fileno(stderr));
return 0;
}
```

OUTPUT:-

```
coldwater@sushovan:~/Desktop/UNP/1941012580/Assignment6$ gcc A6Q4.c
coldwater@sushovan:~/Desktop/UNP/1941012580/Assignment6$ ./a.out
PID=247
```

5. Consider the given code snippet to generate few **fd** values. If the **fd** numbers are printed only odd values, then state the answers for even **fds**.

```
int main()
{
    FILE *myfp;
    int fd,i;
    for(i=0;i<16;i++){
        fd=open("anyExistingFilename",O_RDONLY);
        if(fd==-1){
            perror("Opening error");
            return 1;
       }
       printf("FD number=%d\n",fd);
       myfp=fopen("anyExistingFilewname","r");
       if(myfp==NULL){
            printf("File opening error");
            return 2;
     }
    }
    return 0;
}
```

PROGRAM:-

```
#include<stdio.h>
#include <sys/select.h>
#include<sys/time.h>
int main() {
fd_set r;
FD_ZERO(fd_set *fdset);
FD_SET(int fd, fd_set *fdset );
FD_SET(int fd, fd_set *fdset);
FD_SET(int fd, fd_set *fdset );
select(int maxfdf 1,fd_set *readset, fd_set *writeset, fd_set
*exceptset, NULL);
return 0;
}
```

6. Consider the given code snippet to generate few **fd** values values. If the **fd** values are printed only even values, then state the answers about odd number **fds**.

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/stat.h>
int main()
{
    FILE *myfp;
    int fd,i;
    for(i=0;i<16;i++){
      myfp=fopen("anyExistingFilewname","r");
      if(myfp==NULL){
           printf("File opening error");
           return 1;
       }
       fd=open("anyExistingFilename",O_RDONLY);
       if(fd==-1){
           perror("Opening error");
           return 2;
       }
      printf("FD number=%d\n",fd);
    }
    return 0;
}
```

PROGRAM:-

```
#include<stdio.h>
#include <sys/select.h>
#include<sys/time.h>
fd_set w;
struct timeval timeout;
timeout.tv_sec = 5;
timeout.tv_usec = 0;
FD_ZERO(fd_set *fdset);
FD_SET(int fd, fd_set *fdset );
FD_SET(int fd, fd_set *fdset);
FD_SET(int fd, fd_set *fdset );
select(int maxfdf 1,fd_set *readset, fd_set *writeset, fd_set
*exceptset, const struct timeval*timeout);
return 0;
}
```

7. Fillout the parameters of select function call to monitor any of the descriptors in the set $\{1, 4, 5\}$ are ready for reading and the select call will return when one of the specified descriptors is ready for IO (i.e. a case of **wait forever**).

```
#include<stdio.h>
#include<sys/select.h>
#include<sys/time.h>
int main()
{
  fd_set r;
  FD_ZERO(_______);
  FD_SET(____, ____);
  FD_SET(____, ____);
  FD_SET(____, ____);
  select(____, _____, _____, _____, NULL);
  return 0;
}
```

PROGRAM:-

```
#include<stdio.h>
#include <sys/select.h>
#include<sys/time.h>
 int main(){
 fd_set e;
 struct timeval timeout;
 timeout.tv_sec = 5;
 timeout.tv_usec = 0;
 FD_ZERO(fd_set *fdset);
 FD_SET(int fd, fd_set *fdset );
```

```
FD_SET(int fd, fd_set *fdset);
FD_SET(int fd, fd_set *fdset );
select(int maxfdf 1,fd_set *readset, fd_set *writeset, fd_set
*exceptset, const struct timeval*timeout);
return 0;
}
```

8. Fillout the parameters of select function call to monitor any of the descriptors in the set {2, 7, 10} are ready for writing and the select call will **wait up to a fixed amount of time** - return when one of the descriptor is ready for IO, but do not wait beyond the number of seconds and microseconds specified in the **timeval** structure pointed to by the **timeout** argument.

```
#include<stdio.h>
#include<sys/select.h>
#include<sys/time.h>
int main()
{
  fd_set w;
  struct timeval timeout;
  timeout.tv_sec = ____;
  timeout.tv_usec = ____;
  FD_ZERO(_______);
  FD_SET(____, ____);
  FD_SET(____, ____);
  FD_SET(____, ____);
  select(____, _____, _____, _____, ____);
  return 0;
}
```

PROGRAM:-

```
#include<stdio.h>
#include <sys/select.h>
#include<sys/time.h>
int main(){
fd_set e;
struct timeval timeout;
timeout.tv_sec = 5;
void timeout.tv_usec = 0;
FD_ZERO(fd_set *fdset);
FD_CLR(int fd, fd_set *fdset);
return 0;
}
```

9. Fillout the parameters of select function call to monitor any of the descriptors in the set {1, 4} have an exception condition pending and the select call will **not wait at all** -return immediately after checking the descriptors. To specify this, the **timeout** argument must point to a **timeval** structure and the timer value (*the number of seconds and microseconds specified by the structure*) must be 0.

```
#include<stdio.h>
#include<sys/select.h>
#include<sys/time.h>
int main()
{
  fd_set e;
  struct timeval timeout;
  timeout.tv_sec = ____;
  timeout.tv_usec = ____;
  FD_ZERO(_______);
  FD_SET(____, ____);
  FD_SET(____, ____);
  select(____, _____, _____, _____, ____);
  return 0;
}
```

PROGRAM:-

```
#include<stdio.h>
#include <sys/select.h>
#include<sys/time.h>
int main(){
fd_set e;
struct timeval timeout;
timeout.tv_sec = 5;
```

```
void timeout.tv_usec = 0;
FD_ZERO(fd_set *fdset);
FD_ISSET(int fd, fd_set *fdset);
FD_CLR(int fd, fd_set *fdset);
return 0;
}
```

10. Write the required statements using the macro **FD_CLR(___, ___)** for the code snippet given in question no.-8 to turn off the bit for fds 4 and 5 in the **fdset**.

PROGRAM:-

```
#include<sys/select.h>
#include<sys/time.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main(void){
int retval;
fd_set rfds;
struct timeval tv;
FD_ZERO(&rfds);
FD_SET(0, &rfds);
tv.tv_sec = 10;
tv.tv_usec =20;
retval = select(1, &rfds, NULL, NULL, &tv);
if (retval == -1)
 {
        perror("select()");
        }
    else if (retval){
        printf("Data onto the monitor.\n");
         if(FD_ISSET(0, &rfds))
         printf("FD_0 is set");
        }
    else
    {
        printf("Time Out: Data Not Ready.\n");
        }
    return 0;
}
```

11. Write the required statements using the macro **FD_ISSET(___, ___)** after select call for the code snippet given in question no.-9 to test, *is the bit for fds 2 and 7* ***on*** *(i.e. set or not) in the* **fdset**.

PROGRAM:-

```
#include<sys/select.h>
#include<sys/time.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#inlude<string.h>
int main(){
int fd, ret, pollret, timeout;
char buf[20];
struct pollfd fds[1];
while(1){
fds[0].fd=0;
fds[0].events=0;
```

```
        fds[0].events|=POLLIN;
        timeout=6000;
        pollret =poll (fds, 1, timeout);
        if(pollret=-0){
        printf("timeout :No fd ready\n");
        }
        else{
        menset ((void*)buf,0,11) ;
        ret=read(fd, (void*)buf, 10) ;
        printf("ret-%d\n,ret);
        if(ret!=1)
        printf ("buf=%s" , buf);
        }
        }
      return 0;
  }
```

OUTPUT:-

```
coldwater@sushovan:~/Desktop/UNP/1941012580/Assignment6$ gcc A6Q11.c
coldwater@sushovan:~/Desktop/UNP/1941012580/Assignment6$ ./a.out
PID=6021
```