LABORATORY ASSIGNMENTS-05

UNIX Systems Programming (CSE 3041)

Working with UNIX I/O: read, write, open, close, select, poll, and dup2 system calls.

Practice Assignment/ Minor Assignment::

1. Consider the below given code segment;

```
#define BLKSIZE 8
int main(void)
{
   char buf[BLKSIZE];
   read(STDIN_FILENO, buf, BLKSIZE);
   write(STDOUT_FILENO, buf, BLKSIZE);
   return 0;
}
```

   Run the above code for the different inputs given below, and observe the output. Find a conclusion for each output that the system generate.

   (a) STUDENT↵ [Exactly 8 bytes including enter]
   (b) STUDENTS↵ [Exactly 9 bytes including enter]
   (c) STUDENTSpwd↵ [More than 8 byte]
   (d) STUDENTSpwd;who↵ [More than 8 byte]
   (e) STUDENTSecho $$↵ [More than 8 byte]
   (f) STUDENTSBETCH↵ [More than 8 byte]
   (g) studentsecho ”USP IS TO PRACTICE”↵ [More than 8 byte]
   (h) STUD↵ [Less than 8 byte]

2. Run the given sample code to print the file descriptor numbers assigned on `open()` system call.

```
int main()
{
  int fd,i;
  for(i=0;i<10;i++){
     fd=open("read.c",O_RDONLY);
     if(fd==-1){
       perror("Open error");
       return 1;
     }
    sleep(2);
    printf("File descriptor Number=%d\n",fd);
  }
   return 0;
}
```

   Observe that file descriptor numbers are assigned sequentially depending on first unused slot.

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

```
FILE *fp;
fp=fopen("/home/student/Test.dat","w");
if(fp==NULL){
   return 1;
}
fprintd(fp,"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 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 vale */

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

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

4. Guess the output of the below code snippet.

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/stat.h>
int main()
{
 FILE *myfp,*fp;
 int fd;
 fd=open("read.c",O_WRONLY);
 printf("File descriptor  number=%d\n",fd);
 fd=open("read.c",O_WRONLY);
 printf("File descriptor  number=%d\n",fd);
 fd=open("read.c",O_WRONLY);
 printf("File descriptor  number=%d\n",fd);
 myfp=fopen("swap.c","w");
 fp=fopen("swap.c","w");
 fd=open("read.c",O_WRONLY);
 printf("File descriptor  number=%d\n",fd);
 fd=open("read.c",O_WRONLY);
 printf("File descriptor  number=%d\n",fd);
 fd=open("read.c",O_WRONLY);
 printf("File descriptor  number=%d\n",fd);
 return 0;
}
```

Are you getting any deviation in printing the file descriptor sequence numbers? Why it is happening? State your answer.

5. Write a program to open an existing file and do the followings;
   (a) Count the number of characters in the file.
   (b) Count the number of words in the file.
   (c) Count the number of line in the file.
   (d) Copy first three lines of a file to another.

6. Consider the given code snippet to generate few fd values. If the fds are printed only odd values, then state the answers for even fds and draw the schematic diagram of the process file table descriptor.

```
#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++){
      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;
}
```

7. Consider the given code snippet to generate few fd values. If the fds 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;
}
```

8. Let us consider a text file **input.txt** given below;

```
10 12
21 14
23 56
54 67
67 87
```

   Write a program to read the file and process the inputed file to swap the columns as shown below. Write swapped value into the file **output.txt**.

```
12 10
14 21
56 23
67 54
87 67
```

9. Write a program to open a new file in write mode and do the followings;
   (a) Write the some text in the file and save it.
   (b) Copy the content in another file.
   (c) Print the file descriptor of both the files and give the justification of such fds.

10. How does the output appear when the following program executes?

```
#include <stdio.h>
int main(void)
{
     fprintf(stdout,"a");
     fprintf(stderr, "a has been written\n");
     fprintf(stdout,"b");
     fprintf(stderr,"b has been written\n");
     fprintf(stdout,"\n");
     return 0;
}
```

11. How does the output appear when the following program executes?

```
#include <stdio.h>
int main(void)
{
   int i;
   fprintf(stdout, "a");
   scanf("%d", &i);
   fprintf(stderr, "a has been written\n");
   fprintf(stdout, "b");
   fprintf(stderr, "b has been written\n");
   fprintf(stdout, "\n");
   return 0;
}
```

12. Find the output of the following code snippet.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<wait.h>
#include<sys/stat.h>
int main()
{
   pid_t pid;
   int fd,i,status;
   pid=fork();
   if(pid==0){
      for(i=0;i<5;i++){
         fd=open("select.c",O_RDONLY);
         printf("FD in child=%d\n",fd);
      }
   }
   if(pid==wait(&status)){
      for(i=0;i<5;i++){
         fd=open("select.c",O_RDONLY);
         printf("FD in parent=%d\n",fd);
```

```
        }
    }
    return 0;
}
```

13. Predict the the values of fd will be printed. Which process(parent/child) is printing the fd values?

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<wait.h>
#include<sys/stat.h>
int main()
{
    pid_t pid;
    int fd,i,status;
    pid=fork();
    if(pid==0){
        for(i=0;i<5;i++){
            fd=open("select.c",O_RDONLY);
            printf("FD in child=%d\n",fd);
        }
    }
    if(pid!=waitpid(-1,&status,0)){
        for(i=0;i<5;i++){
            fd=open("select.c",O_RDONLY);
            if(pid==0)
                printf("FD in child=%d\n",fd);
            else
                printf("FD in parent=%d\n",fd);
        }
    }
    return 0;
}
```

14. Predict the output of the following code. Determine which process is printing the fd values.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<wait.h>
#include<sys/stat.h>
int main()
{
    pid_t pid;
    int fd,i,status;
    pid=fork();
    if(pid==0){
       for(i=0;i<5;i++){
          fd=open("select.c",O_RDONLY);
          printf("FD in child=%d\n",fd);
       }
    }
    if(pid!=waitpid(-1,&status,0)){
       for(i=0;i<5;i++){
          fd=open("select.c",O_RDONLY);
             printf("FD in parent=%d\n",fd);
       }
    }
    return 0;
 }
```

15. Create a C code to print the content and total number of bytes written to the standard output device file from two different FDs used by the parent and child processes. Develop the code as per the given procedure;

    (a) Create two different text file named **`ParentData.txt`**, and **`ChildData.txt`** in your current directory. Add character data into **`ParentData.txt`** file and numeric data into **`ChildData.txt`** file. Save your file(s).

    (b) Your main program will take the paremeters **`argc`** and **`argv`**.

    (c) The two files will be given as input to your code from command line. So, total number of arguments will be three. If number of argument will be less or more than three, your program must generate a usage message and return from the program.

    (d) Open two different files in read mode in your program. Provide an error handling message, if **`open()`** returns -1.

    (e) After the successful open use **`fork()`** to create a child process.

    (f) After forking, let the parent process will read from the FD1 file and print onto the standard output device. The parent process also display the total number of bytes written on standard output.

    (g) After forking, let the child process will read from the FD2 file and print onto the standard output device. The child process also display the total number of bytes written on standard output.

    (h) To copy from one FD to another FD, you are required to create a user defined function, whose signature is given as;

    ```
                size_t copyfile(int fromfd, int tofd);
    ```

    (i) The function will be called as **`Totalbytes=copyfile(fd,STDOUT_FILENO)`** and display the total bytes.

    (j) You are not allowed to use **`wait`** or **`waitpid`** at this point.

    (k) After compilation, run the code and make an minute observation on the output. Note the reason for getting such output.

    (l) Can you able to modify the above designed code so that it will print the total number of bytes read from the two files?

(m) Now, use **`wait`** or **`waitpid`** in the program, and observe the output. Does it make any difference from the output of bit-k ?

NOTE: The above program gives an idea of monitoring multiple file descriptors. Now, think of another solution to monitor multiple file descriptors.