Department of Computer Science and Engineering
Institute of Technical Education & Research, SOA, Deemed to be University

MINOR ASSIGNMENT-06

UNIX Network Programming (CSE 4042)

Publish Date: 02-07-2022 **Submission Date:** 11-07-2022

Working with IO multiplexing techniques: `select` and `poll`

This assignment is designed to practice with file descriptors, file pointers and to monitor multiple file descriptors.

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.

```c
#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;
}
```

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

```c
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;
}
```

Hint: To get around the list of descriptors opened for the process, visit the directory **`ls /proc/PID/fd`** in a new terminal without terminating the program.

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;
}
```

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;
}
```

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;
}
```

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;
}
```

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;
}
```

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;
}
```

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;
}
```

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**.

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***.

12. Develop a program using **select()** function call to monitor the standard input file descriptor (i.e *stdin* as file pointer or *0/STDIN_FILENO* as file descriptor) for 10 seconds and 20 microseconds. If the data is ready in the standard file descriptor, it will display the data onto the monitor or time out:data not ready will be displayed.

13. Consider the below given code segment. Compile the code after adding required headers and write the output.

```
int main(void)
{
   fd_set rfds;
   struct timeval tv;
   int retval;
   FD_ZERO(&rfds);
   FD_SET(0, &rfds);
   tv.tv_sec = 5;
   tv.tv_usec = 0;
   retval = select(1, &rfds, NULL, NULL, &tv);
   if (retval == -1)
      perror("select error");
   else if (retval)
      printf("Data is available now.\n");
   else
     printf("No data within five seconds.\n");
   return 0;
}
```

Now, modify the above code as per the following requirements and check the output.

(a) Create a file **file1.txt** in the same directory with content "*select() function is interesting to implement I/O multiplexing in UNIX OS*" and save it.

(b) Open the file **file1.txt** in read mode and returned descriptor will be **fd1**. Set the descriptor **fd1** using **FD_SET** macro. Change the select() system call of the above program to monitor of two fds namely **STDIN** and **fd1**.

(c) Check, which one of the descriptors is ready using **FD_ISSET** macro.

(d) Add one more select() system call to the program to monitor **STDOUT** and one more file where you want to write something from another file.

(e) Can you explain why **fd1** is always providing *TRUE* results and **STDIN** always *FALSE* in case of question (c)?

14. Implement the TCP client using select to monitor whether the socket is readable or **stdin** is readble in TCP concurrent client-server application for echo server that performs the following steps;

i) The client reads a line of text from its standard input and write the line to the server.

ii) The server reads the line from its network input and echoes the line back to the client.

iii) The client reads the echoed line and prints its on its standard output.

iv) The server will keep a track of the client number that is connected to the server and also the child server process ID.

15. Modify the client side code in question no.-14 to handle the EOF correctly with **shutdown** function (*Reference:* **Figure 6.13** of the Text Book).

16. Implement Question number 12 using **poll** method to monitor multiple file descriptor.

17. Write the following code for testing **poll()** system call.

```
int main()
{
   int fd,ret,pollret,timeout;
   char buf[20];
   struct pollfd fds[1]; /*predefined structure */
   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
      {
         memset((void*)buf,0,11);
         ret=read(fd,(void*)buf,10);
         printf("ret=%d\n",ret);
         if(ret!=1)
            printf("buf=%s",buf);
       }
   }
   return 0;
}
```

The above code contains **memset()** function. The function prototype and uses of the function is given as;

```
memset - fill memory with a constant byte

The memset() function fills the first n bytes of the
   memory area pointed to by s with the constant byte c.
```

```
#include <string.h>
void *memset(void *s, int c, size_t n);

RETURN VALUE: The memset() function returns a pointer to
   the memory area s.
```

For more information on **memset**, refer to **man** page. Another function **void bzero(void *s, size_t n);** sets the first **n** bytes of the area starting at **s** to zero (bytes containing '\0').

Incorporate the following changes in the program to get more exposure to **poll** event flags.

(a) Add one more **fd** in **pollfd** structure and set them accordingly. Design your **poll()** system call.

(b) Use `POLLOUT` for checking standard output polling.

(c) Use `POLLRDNORM`, and `POLLWRNORM` for checking the same.

(d) Use `POLLERR` (i.e you can write test as `fds[i].revents==POLLERR` for *error occured on the descriptor*) in your code.

(e) Check the output without using memset().

(f) Check the error in **poll()** system call using invalid **fds** using `POLLNVAL` (i.e you can write test as `fds[i].revents==POLLNVAL` for *file descriptor invalid*). In the same fashion `fds[i].revents==POLLHUP` can be stated for *device has been disconnected*

(g) Check **fd[0].revents** value in the given program. Do the same after the modification of (a), (b) and (c) respectively.

NOTE: `POLLERR`, `POLLNVAL`, and `POLLHUP` are specifically used along with **revents** only.

18. [**Optional**] You can also monitor multiple file descriptors using another techinque called **pselect()** - *allow a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become "ready" for some class of I/O operation (e.g., input possible). A file descriptor is considered ready if it is possible to perform a corresponding I/O operation.* Modify the program in question number-14 to use pselect.

You are required to show the sample input and output for various part of the assignment.