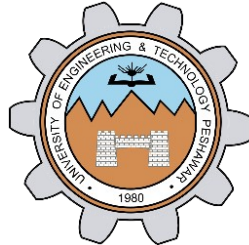


**MULTIPLE FILE COPYING**

**LAB # 07**



**Fall 2020**

**CSE302L System Programming Lab**

Submitted by: **Shah Raza**

Registration No. : **18PWCSE1658**

Class Section: **B**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_

Submitted to:

**Engr. Madiha Sher**

Friday, January 22<sup>nd</sup>, 2021

**Department of Computer Systems Engineering**  
**University of Engineering and Technology, Peshawar**

## Lab Objective(s):

- Understand and implement the select function.

## Task # 01:

**Write a program that copies two files sequentially in a single process.**

## Code:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <errno.h>
#include <string.h>

int main(int argc, char* argv[])
{
    if (argc != 5)
    {
        printf("Invalid number of arguments\n");
        return -1;
    }
    int fd[4];
    for (int i = 0; i < argc-1; i+=2)
    {
        fd[i] = open(argv[i+1], O_RDONLY);
        if (fd[i] == -1)
        {
            printf("Failed to open %s: %s\n", argv[i+1], strerror(errno));
            return -1;
        }
        fd[i+1] = open(argv[i + 2], O_WRONLY | O_CREAT , S_IRWXU);
        if (fd[i+1] == -1)
        {
            printf("Failed to open %s: %s\n", argv[i+2], strerror(errno));
            return -1;
        }
    }
    char buffer[1024];
    int bytesRead;
    int bytesWritten;
    for (int i = 0; i < argc-1; i += 2)
    {
        do
        {
            bytesRead = read(fd[i],buffer,sizeof(buffer));
            if (bytesRead == -1)
            {
                printf("Failed to read from %s: %s\n",argv[i+1],strerror(errno));
                return -1;
            }
        }
    }
```

```

    }
    bytesWritten = write(fd[i+1],buffer,bytesRead);
    if (bytesWritten == -1)
    {
        printf("Failed to Write to %s: %s\n", argv[i + 2], strerror(errno));
        return -1;
    }
} while (bytesRead!=0);
}
for (int i = 0; i < argc-1; i++)
{
    int ret = close(fd[i]);
    if (ret == -1)
        printf("Failed to close %s: %s\n",argv[i+1],strerror(errno));
}
return 0;
}

```

### Output/Results:

```

shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 1$ ls
f1.txt f2.txt task1 task1.c
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 1$ cat f1.txt
The woods are lovely, dark and deep
But I have promises to keep
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 1$ cat f2.txt
And miles to go before I sleep
And miles to go before I sleep
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 1$ ./task1 f1.txt c1.txt f2.txt c2.txt
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 1$ ls
c1.txt c2.txt f1.txt f2.txt task1 task1.c
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 1$ cat c1.txt
The woods are lovely, dark and deep
But I have promises to keep
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 1$ cat c2.txt
And miles to go before I sleep
And miles to go before I sleep

```

### Discussion:

While sequentially copying files in a single process works fine in our case, it may lead to a problem where the process gets blocked even though one of the two files is ready for copying.

## Task # 02:

Write a program that monitors two files by forking a child.

### Code:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <errno.h>
#include <string.h>
#include <sys/wait.h>

int main(int argc, char* argv[])
{
    if (argc != 5)
    {
        printf("Invalid number of arguments\n");
        return -1;
    }
    int x;
    for(int i=1;i<argc;i+=2)
    {
        x= fork();
        if(x==0)
        {
            int fd1 = open(argv[i],O_RDONLY);
            if(fd1==-1)
            {
                printf("Failed to open %s: %s\n", argv[i], strerror(errno));
                return -1;
            }

            int fd2 = open(argv[i+1],O_WRONLY | O_CREAT, S_IRWXU |
S_IRWXG | S_IRWXO);
            if(fd2==-1)
            {
                printf("Failed to open %s: %s\n", argv[i+1], strerror(errno));
                return -1;
            }
            int bytesread;
            char buffer[1024];
            do
            {
                bytesread = read(fd1,buffer,sizeof(buffer));
```

```

        if(bytesread==-1)
        {
            printf("Failed to read from %s: %s\n",argv[i],strerror(errno));
            return -1;
        }
        int byteswritten = write(fd2,buffer,bytesread);
        if(byteswritten==-1)
        {
            printf("Failed to Write to %s: %s\n", argv[i + 1],
strerror(errno));
            return -1;
        }
    }while(bytesread!=0);

    int cfd1 = close(fd1);
    if(cfd1==-1)
    {
        printf("Failed to close %s: %s\n",argv[i],strerror(errno));
        return -1;
    }
    int cfd2 = close(fd2);
    if(cfd2==-1)
    {
        printf("Failed to close %s: %s\n",argv[i+1],strerror(errno));
        return -1;
    }
    break;
}
}
if(x>0)
{
    for(int i=1;i<argc/2;i++)
        wait(NULL);
}
return 0;
}

```

## Output:

```
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 2$ ls
f1.txt f2.txt task2 task2.c
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 2$ cat f1.txt
After ages of despair
There's a smile, be it brief
Are you truly happy
Or distracted from your grief?
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 2$ cat f2.txt
I could have persisted
My will could have been stronger
If I knew it was the last time
I would've stayed longer
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 2$ ./task2 f1.txt c1.txt f2.txt c2.txt
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 2$ ls
c1.txt c2.txt f1.txt f2.txt task2 task2.c
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 2$ cat c1.txt
After ages of despair
There's a smile, be it brief
Are you truly happy
Or distracted from your grief?
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 2$ cat c2.txt
I could have persisted
My will could have been stronger
If I knew it was the last time
I would've stayed longer
```

## Discussion:

Copying files in parallel using multiple child processes is a good solution to our problem but it also has its limitations. The parent and child processes have separate address spaces and it is difficult for them to communicate.

### **Task # 03:**

**Write a program that monitors two files for reading using 'select'.**

#### **Code:**

```
#include <stdio.h>
#include <sys/select.h>
#include <time.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <errno.h>
#include <string.h>

int main(int argc, char *argv[])
{
    if(argc!=3)
    {
        printf("Invalid Number of Arguments\n");
        return -1;
    }
    int fd1 = open(argv[1],O_RDONLY);
    if(fd1==-1)
    {
        printf("Failed to open %s: %s\n",argv[1],strerror(errno));
        return -1;
    }

    int fd2 = open(argv[2],O_RDONLY);
    if(fd2==-1)
    {
```

```

        printf("Failed to open %s: %s\n",argv[2],strerror(errno));
        return -1;
    }
    fd_set readSet;
    FD_ZERO(&readSet);
    FD_SET(fd1,&readSet);
    FD_SET(fd2,&readSet);
    int maxFD = (fd1>fd2)?fd1:fd2;
    struct timeval timeout;
    timeout.tv_sec = 5;
    timeout.tv_usec = 73;
    int nfds = select(maxFD+1,&readSet,NULL,NULL,&timeout);
    if(nfds==-1)
    {
        perror("An error Occured with select function\n");
        return -1;
    }
    if(nfds==0)
    {
        printf("Timeout\n");
        return -1;
    }
    printf("Number of files ready: %d\n",nfds);
    if(FD_ISSET(fd1,&readSet))
        printf("%s is ready\n",argv[1]);
    else
        printf("%s is not ready\n",argv[1]);
    if(FD_ISSET(fd2,&readSet))
        printf("%s is ready\n",argv[2]);
    else

```



```
        printf("%s is not ready\n",argv[2]);  
    return 0;  
}
```

### Output:

```
shahsomething@ubuntu:~/System Programming/labs/Lab 7/Task 3$ ./task3 file1.txt file2.txt  
Number of files ready: 2  
file1.txt is ready  
file2.txt is ready
```

### Discussion:

The select function is a complete solution to our problem. It prevents the process from being blocked if a file is not ready and it can also communicate easily with the calling process using its return values.

### Task # 04:

**Write a program that monitors N files for reading using 'select'.**

#### Code:

```
#include <stdio.h>  
#include <sys/select.h>  
#include <time.h>  
#include <unistd.h>  
#include <fcntl.h>  
#include <sys/stat.h>  
#include <errno.h>  
#include <string.h>  
  
int main(int argc, char *argv[])  
{  
    if(argc<2)  
    {  
        printf("Invalid Number of Arguments\n");  
    }  
}
```

```
        return -1;
    }
    int fd[argc-1];
    for(int i=0;i<argc-1;i++)
    {
        fd[i] = open(argv[i+1],O_RDONLY);
        if(fd[i]==-1)
        {
            printf("Failed to open %s: %s",argv[i+1],strerror(errno));
            return -1;
        }
    }
    fd_set readSet;
    FD_ZERO(&readSet);
    for(int i=0;i<argc-1;i++)
    {
        FD_SET(fd[i],&readSet);
    }
    int maxFD = fd[0];
    for(int i=0;i<argc-1;i++)
    {
        if(fd[i]>maxFD)
            maxFD = fd[i];
    }
    struct timeval timeout;
    timeout.tv_sec = 5;
    timeout.tv_usec = 42;
    int nfd = select(maxFD+1,&readSet,NULL,NULL,&timeout);
    if(nfd==-1)
    {
```

```

        perror("An error Occured with select function\n");
        return -1;
    }
    if(nfds==0)
    {
        printf("Timeout\n");
        return -1;
    }
    printf("Number of files ready: %d\n",nfds);
    for(int i=0;i<argc-1;i++)
    {
        if(FD_ISSET(fd[i],&readSet))
            printf("%s is ready\n",argv[i+1]);
        else
            printf("%s is not ready\n",argv[i+1]);
    }
    return 0;
}

```

### Output:

```

shahsomething@ubuntu:~/System Programming/Labs/Lab 7/Task 4$ ./task4 f1.txt f2.txt f3.txt f4.txt
Number of files ready: 4
f1.txt is ready
f2.txt is ready
f3.txt is ready
f4.txt is ready

```

### Discussion:

One of the main advantages of select function is that it is not limited to monitoring just two files, it can be used to monitor as many files as we want.

### Task # 05:

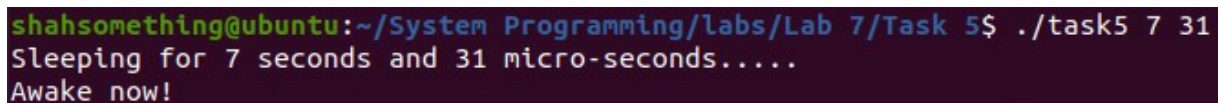
**Write a function that creates a delay of N seconds using select function. Pass N as an argument to the function.**

#### Code:

```
#include <stdio.h>
#include <sys/select.h>
#include <time.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    printf("Sleeping for %s seconds and %s micro-seconds.....\n",argv[1],argv[2]);
    struct timeval timeout;
    timeout.tv_sec = atoi(argv[1]);
    timeout.tv_usec = atoi(argv[2]);
    int ret = select(1,NULL,NULL,NULL,&timeout);
    printf("Awake now!\n");
    return 0;
}
```

#### Output:



```
shahsomething@ubuntu:~/System Programming/Labs/Lab 7/Task 5$ ./task5 7 31
Sleeping for 7 seconds and 31 micro-seconds.....
Awake now!
```

#### Discussion:

While there are dedicated functions such as sleep(), that are used for delaying purpose, it is also note-worthy that the select function can also be used for creating delays.