NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

PROGRAM: SOFTWARE ENGINEERING

OPERATING SYSTEMS LAB

Lab_Task 08

SUBMITTED BY:
Name: Ahmed Ali
Roll No: 22P-9318
Section: BS(SE)-5B

INSTRUCTOR NAME: Sir Saad
A DEPARTMENT OF COMPUTER SCIENCE

Terminal 1

```
fast@HALAB-11:~$ ps
    PID TTY          TIME CMD
   4559 pts/1    00:00:00 bash
   4592 pts/1    00:00:00 ps
fast@HALAB-11:~$ hello
Ahmed_22p-9319_BSE-5B
```

Terminal 2

```
fast@HALAB-11:~$ echo hello
hello
fast@HALAB-11:~$ echo hello>hello.txt
fast@HALAB-11:~$ cat hello.txt
hello
fast@HALAB-11:~$ echo hello > /proc/4559/fd/1
fast@HALAB-11:~$ echo Ahmed_22p-9319_BSE-5B > /proc/4559/fd/1
fast@HALAB-11:~$
```

```
fast@HALAB-11:~$ cd /proc
fast@HALAB-11:/proc$ ls
1     1451  151   1624  1728  1775  185   2022  2407  2982  34    40    4452  4725  53   658  7     73
1021  1457  152   1633  1738  1779  187   2026  2543  3     345   4043  4479  4726  57   66   70    74
1034  1458  1521  1634  1745  1780  1892  2042  26    30    35    4084  4516  4749  58   67   700   75
1036  1459  1525  1685  1748  1783  19    2055  260   310   356   41    4534  48    59   68   701   757
1038  146   1528  1692  1750  1790  1901  21    27    3109  358   4113  4551  49    599  685  702   814
118   1470  153   1693  1761  1791  1902  2188  28    3126  36    4212  4554  5     6    686  703   83
1197  1480  1531  1694  1762  1795  1914  22    280   3128  3681  423   4559  50    60   689  707   844
12    1485  154   17    1766  1797  1923  220   2825  3142  38    4292  4591  501   61   69   708   847
1204  1494  155   1702  1768  18    1939  221   2827  3181  3873  43    46    505   62   690  709   871
13    1497  1556  1708  1769  1824  1993  23    29    32    39    4319  4622  506   63   691  710   910
14    15    1588  1713  1771  183   1996  2352  2900  3275  3923  4326  47    507   64   692  711   98
1413  150   16    1717  1772  1833  2     2371  2942  33    393   44    4706  51    65   693  712   acpi
1450  1503  1604  1721  1773  184   20    24    2960  3359  4     4443  4718  52    657  699  72    asound
```

4.1 Open a File

```c
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
char *path = argv[1];
int fd = open(path, O_WRONLY | O_EXCL | O_CREAT);

if (fd == -1)
{
printf("Error: File not Created\n");
return 1;
}
return 0;
}
~
```

```
fast@HALAB-11:/proc/4559/fd$ cd
fast@HALAB-11:~$ vim demo.c
fast@HALAB-11:~$ gcc demo.c -o demo
fast@HALAB-11:~$ ./ demo createThisFile
bash: ./: Is a directory
fast@HALAB-11:~$ ./ demo createThisFile
bash: ./: Is a directory
fast@HALAB-11:~$ ./demo createThisFile
fast@HALAB-11:~$ ls
 1      a.out           DANYAL_LAB6.zip    demo.c                          Documents
 abc    combine.txt     DANYAL_LAB6..zip  'described in this manual page are'   Downloads
 abc1   createThisFile  demo               Desktop                         files.txt
fast@HALAB-11:~$ ls
 1      a.out           DANYAL_LAB6.zip    demo.c                          Documents
 abc    combine.txt     DANYAL_LAB6..zip  'described in this manual page are'   Downloads
 abc1   createThisFile  demo               Desktop                         files.txt
fast@HALAB-11:~$ ./demo createThisFile
Error: File not Created
fast@HALAB-11:~$ ls
 1      a.out           DANYAL_LAB6.zip    demo.c                          Documents
 abc    combine.txt     DANYAL_LAB6..zip  'described in this manual page are'   Downloads
 abc1   createThisFile  demo               Desktop                         files.txt
fast@HALAB-11:~$
```

Question: What is the size of the file? Why is it this size?
Answer: When I run the program, createThisFile is created but it is empty. Therefore, the size of the file is **0 bytes** because the file is created without any content being written to it
it just exists as a file on the filesystem without any data

The size of createThisFile is 0 bytes because it was created but not written to.

what are argc and argv in c and How it works:
The first parameter, argc (argument count) is an integer that indicates how many arguments were entered on the command line when the program was started.
The second parameter, argv (argument vector), is an array of pointers to arrays of character objects.

argv and argc are how command line arguments are passed to main() in C and C++.

argc will be the number of strings pointed to by argv. This will (in practice) be 1 plus the number of arguments, as virtually all implementations will prepend the name of the program to the array.

The variables are named argc (*argument count*) and argv (*argument vector*) by convention, but they can be given any valid identifier: int main(int num_args, char** arg_strings) is equally valid.

They can also be omitted entirely, yielding int main(), if you do not intend to process command line arguments.

Printing value of fd:

```
fast@HALAB-11:~$ vim demo1.c
fast@HALAB-11:~$ gcc demo1.c -o demo1
fast@HALAB-11:~$ ./demo1
val of fd: -1 Error: File not Created
fast@HALAB-11:~$
```

```
fast@HALAB-11:~$ ./demo1 Ahme
val of fd: 3 fast@HALAB-11:~$
```

WITHOUT COMMENTING close(fd); **Output**

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
if (argc != 2)
{
printf("Error: Run like this: ");
printf("%6s name-of-new-file\n", argv[0]);
return 1;
}
char *path = argv[1];
int i = 0;
while(i<2)
{
int fd = open(path, O_WRONLY | O_CREAT);
printf("Created! Descriptor is %d\n", fd);
close(fd);
i++;
}
return 0;
}
~
```

```
fast@HALAB-11:~$ vim Ahmed.c
fast@HALAB-11:~$ gcc Ahmed.c -o Ahmed.out
fast@HALAB-11:~$ ./
abc/            a.out           DANYAL_LAB6.zip/ Desktop/
abc1/           .cache/         demo             Documents/
Ahmed.out       .config/        demo1            Downloads/
fast@HALAB-11:~$ ./Ahmed.out Ahmed
Created! Descriptor is 3
Created! Descriptor is -1
fast@HALAB-11:~$
```

Now COMMENTING close(fd);

```c
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
if (argc != 2)
{
printf("Error: Run like this: ");
printf("%6s name-of-new-file\n", argv[0]);
return 1;
}
char *path = argv[1];
int i = 0;
while(i<2)
{
int fd = open(path, O_WRONLY | O_CREAT);
printf("Created! Descriptor is %d\n", fd);
//close(fd);
i++;
}
return 0;
}
```

Output:

```
fast@HALAB-11:~$ ./a.out Ahmed
Created! Descriptor is 3
Created! Descriptor is 4
fast@HALAB-11:~$
```

I am getting different values for the file descriptors because I have not closed the first one. This keeps it in use, so the second open call receives the next available higher value instead of reusing the first descriptor

Q1 What is 0666 that is specied in the open() call? What does it mean?
Answer: 0666 sets the file permissions for a newly created file, allowing read and write access for the owner, group, and others.

```c
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
char* get_timeStamp()
{
time_t now = time(NULL);
return asctime(localtime(&now));
}
int main(int argc, char* argv[])
{
char *filename = argv[1];
char *timeStamp = get_timeStamp();
int fd = open(filename, O_WRONLY | O_APPEND | O_CREAT, 0666);
size_t length = strlen(timeStamp);
write(fd, timeStamp, length);
close(fd);
return 0;
}
```

OUTPUT:

```
fast@HALAB-11:~/Desktop/LAB TASK 8$ vim write.c
fast@HALAB-11:~/Desktop/LAB TASK 8$ gcc write.c
fast@HALAB-11:~/Desktop/LAB TASK 8$ ./a.out
fast@HALAB-11:~/Desktop/LAB TASK 8$ ./a.out Ok
fast@HALAB-11:~/Desktop/LAB TASK 8$ cat Ok
Mon Oct 21 13:07:53 2024
fast@HALAB-11:~/Desktop/LAB TASK 8$
```

The first digit (0) indicates that it is an octal number
The next three digits (666) specify the permissions:
6 (read and write) for the owner
6 (read and write) for the group
6 (read and write) for others

Q2 What is O_APPEND doing in the same call? Run the program again and check it's output.
Answer: O_APPEND ensures that any new data is added to the end of the file. When you run the program multiple times, each timestamp will be appended instead of overwriting previous ones.

```c
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>

char* get_timeStamp()
{
    time_t now = time(NULL);
    return asctime(localtime(&now));
}
int main(int argc, char* argv[])
{
    char *filename = argv[1];
    char *timeStamp = get_timeStamp();
    //O_APPEND appends data to the end of the file
    int fd = open(filename, O_WRONLY | O_APPEND | O_CREAT, 0666);
    size_t length = strlen(timeStamp);
    write(fd, timeStamp, length);
    close(fd);
    return 0;
}
```

```
fast@HALAB-11:~/Desktop/LAB TASK 8$ gcc write1.c
fast@HALAB-11:~/Desktop/LAB TASK 8$ ./a.out Ok
fast@HALAB-11:~/Desktop/LAB TASK 8$ cat Ok
Mon Oct 21 13:07:53 2024
Mon Oct 21 13:38:11 2024
```

Q3 Modify the following line in the code and then compile and run the program and check it's output.
From:
size_t length = strlen(timeStamp);
To:
size_t length = strlen(timeStamp)-5;

```c
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
char* get_timeStamp()
{
    time_t now = time(NULL);
    return asctime(localtime(&now));
}
int main(int argc, char* argv[])
{
    char *filename = argv[1];
    char *timeStamp = get_timeStamp();
    int fd = open(filename, O_WRONLY | O_APPEND | O_CREAT, 0666);
    size_t length = strlen(timeStamp) - 5;
    write(fd, timeStamp, length);
    close(fd);
    return 0;
}
```

OUTPUT:

```
fast@HALAB-11:~/Desktop/LAB TASK 8$ vim write2.c
fast@HALAB-11:~/Desktop/LAB TASK 8$ ./a.out Ok
fast@HALAB-11:~/Desktop/LAB TASK 8$ cat Ok
Mon Oct 21 13:07:53 2024
Mon Oct 21 13:38:11 2024
Mon Oct 21 13:43:42 2024
fast@HALAB-11:~/Desktop/LAB TASK 8$
```

READ FROM FILE:

```c
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
if (argc != 2)
{
printf("Error: Run like this: ");
printf("%6s name-of-existing-file\n",
argv[0]);

return 1;
}
char *path = argv[1];
int fd = open(path, O_RDONLY);
if (fd == -1)
{
printf("File does not exist\n");
return 1;
}
char buffer[200];
read(fd, buffer, sizeof(buffer)-1);
printf("Contents of File are:\n");
printf("%s\n", buffer);
close(fd);
return 0;
}
```

Output:

```
fast@HALAB-11:~/Desktop/LAB TASK 8$ ./read.out write.c
Contents of File are:
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
char* get_timeStamp()
{
time_t now = time(NULL);
return as
fast@HALAB-11:~/Desktop/LAB TASK 8$
```