

Week 10: C programs based on APIs.

1. Write a C/C++ program which demonstrates interprocess communication between a reader process and a writer process. Use `mkfifo`, `open`, `read`, `write` and `close` APIs in your program.

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

```
int main (int argc, char * argv[])
```

```
{
    char buf [100];
```

```
    int fd, n;
```

```
    mkfifo (argv [1], S_IFIFO | 0777);
```

```
    if (argc == 3)
```

```
    {
        fd = open (argv [1], O_WRONLY);
```

```
        write (fd, argv [2], strlen (argv [2]));
```

```
        close (fd);
```

```
    }
```

```
    if (argc == 2)
```

```
    {
        fd = open (argv [1], O_RDONLY);
```

```
        n = read (fd, buf, sizeof (buf));
```

```
        buf [n] = '\0';
```

```
        printf ("msg: ", buf);
```

```
        close (fd);
```

```
    }
```

OUTPUT:

\$ cc fifo.c

\$ ./a.out fifo "5b linux lab prog" &  
12168\$ ./a.out fifo  
[1]+ Done

./a.out fifo "5b linux lab prog".

2. Write a C/C++ program to emulate the unix ls command.

⇒ #include &lt;unistd.h&gt;

#include &lt;stdio.h&gt;

#include &lt;string.h&gt;

int main (int argc, char \* argv[])

{

if (argc &lt; 3 || argc &gt; 4)

{  
printf ("Error in usage \n");  
return -1;

}

if (argc == 4 &amp;&amp; strcmp (argv[1], "-s") != 0)

{  
printf ("for symbolic link use -s option");  
return -1;

}

if (argc == 4 &amp;&amp; access (argv[2], F\_OK) == -1)

{  
printf ("source file does not exist");  
return -1;

}

if (argc == 3 &amp;&amp; access (argv[1], F\_OK) == -1)

{  
printf ("source file does not exist");  
return -1;

{



if (argc == 4)

```
symlink (argv[2], argv[3]);
printf ("Symbolic link is created");
return 0;
```

if (argc == 3)

```
link (argv[1], argv[2]);
printf ("Hardlink is created");
return 0;
```

OUTPUT:

\$ nano ms.c

\$ gcc ms.c

\$ ./a.out -s pven.sh symlink

\$ ./a.out vevet.sh hardlink

3. Write a C/C++ POSIX compliant program that prints the POSIX defined configuration options supported on any given system using feature test macros.

2) #define \_POSIX\_SOURCE

#define \_POSIX\_C\_SOURCE 199309L

#include <iostream>

#include <unistd.h>

int main ()

{

using namespace std;

#ifdef \_POSIX\_JOB\_CONTROL

cout << "System supports Job Control feature" << endl;

#else

cout << "System does not support job control fn";

```

#end if
# if def _POSIX_SAVED_IDS
cout << "System supports saved set-UID and saved set-GID"
<<endl;
# else
cout << "System doesn't support saved set-UID\n";
#end if
# if def _POSIX_CHOWN_RESTRICTED
cout << "System supports change ownership feature:" <<endl;
# else
cout << "System does not support change ownership feature\n";
#end if
# if def _POSIX_NO_TRUNC
cout << "System supports path truncation option:" <<endl;
# else
cout << "System does not support path truncation\n";
#end if
# if def _POSIX_VDISABLE
cout << "System supports disable character for files:" <<endl;
# else
cout << "System does not support disable character\n";
#end if
return 0;
}

```

OUTPUT:

```

cc source.c
./a.out

```

System supports lab control features.

System supports change ownership features.

System supports path truncation option.

System supports disable character for files.



Write a C/C++ program to find output the contents of its environment list.

```

int main (int argc, char *argv[], char *envp[])
{
    int i;
    for (i=0; envp[i] != NULL; i++)
        printf ("%s", envp[i]);
    getchar ();
    return 0;
}

```