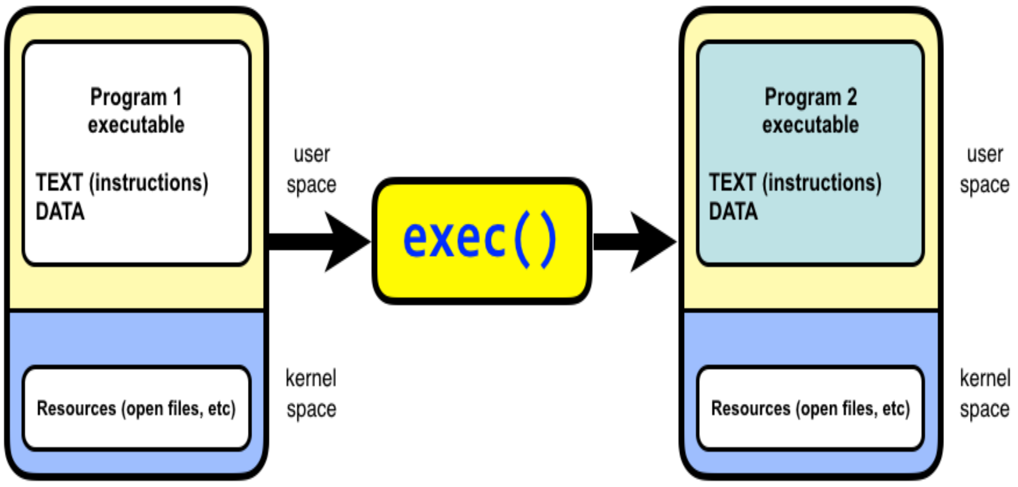
Practical 9- Exec family System calls

- **Exec family functions:**
  - The exec system call replace the program/running process with another program/process.
  - When program calls an exec function, that process immediately stop executing program and begins executing a new program from the beginning, assuming that the exec call does not encounter any error.
  - When a process calls exec, all code (text) and data in the process is lost and replaced with the executable of the new program.
  - all data is replaced,in user space after calling exec.



  - **Exec replaces the calling program with another one, it never returns unless an error occur**
  - The library functions execl, execlp, execle, execv, and execvp are simply convenience functions that allow specifying the arguments in a different way,
  - System call that contain the letter **p** in their names (execvp and execlp) accept a program name and search for a program by that name . Not required for the path of executable file

  - System call that contain the **letter v** in their names (execv, execvp, and execve) accept the argument list for the new program as a NULL terminated array of pointers to strings.
  - System call that contain the letter **l** (execl, execlp, execle) accept the argument list using the C language args mechanism.

- **execl()**

  In execl() system call takes the path of the executable binary file (i.e. /bin/ls) as the first . Second argument. is the list of arguments (i.e. -lh, /home) that passes to the executable followed by NULL.

Syntax :
```
int execl(const char *path, const char *arg, ...  ,  /* (char  *) NULL */);
```

Program –1

```
//This program shows use of execl()
function #include<stdio.h>
int main()
{
            execl("/usr/bin/wc","wc","-l","f1.txt",NULL);
              printf("Done\n");
            exit(0);
}
```

- **execlp()**
  execlp() uses the PATH environment variable. So, if an executable file or command is available in the PATH, then the command or the filename is enough to run it, the full path is not needed.

  Syntax :

  ```
  int execlp(const char *file, const char *arg, …, NULL );
  ```

  Program -2
  ```
  /This program shows use of execlp() function
  #include<stdio.h>
  #include<unistd.h>
  int main()
  {
      execlp("wc","wc" , "-lc", "f1.txt",NULL);
     printf("Done\n");

  }
  ```

- **execv()**
- With execv(), you can pass all the parameters in a NULL terminated array argv. The first element of the array should be the path of the executable file. Otherwise, execv() function works just as execl() function

  ```
  int execv(const char *path, char *const argv[]);
  ```

Program 3

```
#include <unistd.h>

int main(void) {
    char *pathvalue = "/usr/bin/wc";
     char *args[] = {"wc", "-l", "f1.txt", NULL};

  execv(pathvalue, args);

  return 0;
}
```

- **Execvp()**
  Works the same way as execv() system function. But, the PATH environment variable is used. So, the full path of the executable file is not required just as in execlp().

  Syntax
  `int execvp(const char *file, char *const argv[]);`

test.c

```
#include <stdio.h>

void main()
 {
   printf("\nstatement execute by execvp and call test executable file\n");
 }
```

```
$ gcc -o test test.c
```

Program 4

```
#include<stdio.h>
 #include<stdlib.h>
#include<unistd.h>
int main()
{
//A null terminated array of character pointers
char *args[]={"./test",NULL};
execvp(args[0],args);

/*All statements are ignored after execvp() call as this whole process(p4.c) is replaced by
another process (test.c)
*/

printf("Ending      ");
return 0;
}
```

Program 5

```
//Child process takes command and its argument from the user and executes it
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>

int main(int argc, char *argv[])
{
int exitstatus;
switch(fork())
{
  case -1:
    printf("Fork error\n");
    exit(1);
 case 0:
     printf("Child process\n");
     execvp(argv[1],&argv[2]);
    printf("Done\n");

    default:

   wait(&exitstatus);
   printf("proces exit status=%d\n",WEXITSTATUS(exitstatus));
  }
 }
```

Output:

```
$ ./a.out  wc wc -l  f1.txt

child process
4  f1.txt
```

Exercise

1. perform cat f1.txt f2.txt using execl() , execlp() ,execvp() ,execv() [make a seperate program for each exec() call.