1:execl

In execl() system function takes the path of the executable binary file as the first and second argument. Then, the arguments (i.e. -lh, /home) that you want to pass to the executable followed by NULL. Then execl() system function runs the command and prints the output.

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

  main()
  {
     pid_t pid;

     if ((pid = fork()) == -1)
        perror("fork error");
     else if (pid == 0) {
        execl("/u/userid/bin/newShell", "newShell", NULL);
        printf("Return not expected. Must be an execl() error.n");
     }
  }
```

2:execle

NULL-terminated argument list, specify the new process image's environment

```
main() {
pid_t pid;
int e;
char *env[] = { "TERM=xterm", (char *)0 };

        pid=fork();
        if(pid==0){
                execle("/usr/bin/clear","clear", (char *)0,env);
        }
        else{
                wait(&e);
                exit(1);
        }
}
```

3:execlp

NULL-terminated argument list, search for the new process image file in *PATH*

```
#include <process.h>
            #include <stdio.h>
            #include <stdlib.h>
            #include <errno.h>

            char *env[] = {"PATH=C:\\TEST", NULL};

            main()
            {
                execlpe("child.exe", "child", "arg1", "arg2", NULL, env);
                if (errno == ENOENT) {
                    printf("child.exe not found in current directory,\n");
                    printf("  or in any PATH directory\n");
                } else if (errno == ENOMEM)
                    printf("not enough memory to execute child.exe\n");
                else
                    printf("error #%d trying to exec child.exe\n", errno);
            }
```

4:execv

execv(path,argv) **causes the current process to abandon the program that it is running and start running the program in file path**. Parameter argv is the argument vector for the command, with a null pointer at the end. It is an array of strings.

```
int main(void)
{
  int childpid;
  if((childpid = fork()) == -1 )
{
  perror("can't fork");
  exit(1);
}
 else if(childpid == 0)
{
  execl("./testing","","",(char *)0);
  exit(0);
}
else
{
printf("finish");
exit(0);
}
```

```
}
```

5:execve

the execve() system call function is **used to execute a binary executable or a script**. The function returns nothing on success and -1 on error. The first parameter must be the path of a binary executable or a script.

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

  main()
  {
     pid_t pid;
     char *const parmList[] = {"/bin/ls", "-l", "/u/userid/dirname", NULL};
     char *const envParms[2] = {"STEPLIB=SASC.V6.LINKLIB", NULL};

     if ((pid = fork()) ==-1)
        perror("fork error");
     else if (pid == 0) {
        execve("/u/userid/bin/newShell", parmList, envParms);
        printf("Return not expected. Must be an execve error.n");
     }
  }
```

6:execvp

The execvp function is most commonly used to **overlay a process image that has been created by a call to the fork function**. identifies the location of the new process image within the hierarchical file system (HFS).

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>
```

```c
#include <errno.h>


int main(void) {
    const char *args[] = { "vim", "/home/ben/tmp3.txt", NULL };


    errno = 0;
    if (execvp("vim", args) == -1) {
        if (errno == EACCES)
            printf("[ERROR] permission is denied for a file\n");
        else
            perror("execvp");
        exit(EXIT_FAILURE);
    }


    exit(EXIT_SUCCESS);
}
```