Creating Process with C

- The parent of all processes, init, is started when the operating system boots. It is process ID number 1
- As other programs are started, each is assigned a unique process identifier, known as a PID. process IDs are assigned in a sequential order. As processes stop, the previously unavailable PIDs can be used again. Usually, PIDs are in the 1 to 32768 range
- Starting a process is as simple as typing a command at the Unix shell prompt or starting a program from a menu.

- A fork library call or an execve system call is used to start the new program/process
- In fork current running program is copied to make a child.
- The Unix shell includes a built-in command called exec that replaces the running shell with a new
- Program. For example, typing exec date will run the date program, and the original shell will be closed.

- #! (often called a sh-bang) tell the kernel to run the program listed after the #!

//////////////////////////CREATING PROCESS WITH
1.FORK
2.EXEC

///////////////////Fork System Call /////////////////////////

Code1

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

   fork();

  //Any Statements after fork will be copied to child process

   printf("Hello world!\n");

   return 0;
}
```

Output:

```
Hello World!
Hello World!
```

Code2:

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

int main(){

pid_t pid;
pid=fork();
if(pid<0){
printf("Failed to Create Process\n");
return 1;
}
else{
printf("Child Process Created\n");
printf("Child pid=%d Parent pid=%d\n",getpid(),getppid());
}
return 0;
}
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# gcc fork2.c -o fork2
root@ubuntu:~/Desktop/shell_scripts# ./fork2
Child pid=17287 Parent pid=17071
```

Code3:

```
#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>


int main(){


pid_t pid;

pid=fork();

if(pid<0){

printf("Failed to Create Process\n");

return 1;

}

else{

if(pid==0){

printf("Child Process\n");

printf(" pid=%d Child pid=%d\n",pid,getpid()); //Here pid=0 Child pid=17514

}

else{

printf("pid=%d Parent pid=%d\n",pid,getppid()); //Here pid is child process id 17514 Parent pid=17071

}

}

return 0;

}
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# ./fork3

pid=17514 Parent pid=17071

root@ubuntu:~/Desktop/shell_scripts# Child Process

 pid=0 Child pid=17514
```

/////////////////////////// Exec //////////////////////

exec includes a group of system calls in C. Here the copy of the parent process is not created rather parent process dies and a new image is created for the child

```
#include<stdio.h>

#include<unistd.h>  //exec here

#include<sys/types.h>



/* Family or group of functions in exec  l, lp, v, vp, ve

int execl(const char *path, const char *arg, …, NULL);

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

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

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

int execle(const char *path, const char *arg, …, NULL, char * const envp[] );

int execve(const char *file, char *const argv[], char *const envp[]);

*/


//1. execl() system function runs the command and prints the output. If any error occurs, then
execl() returns -1.


int main(void) {

  char *path_command = "/bin/ls";

  char *arg1 = "-l";

  char *arg2 = "/root/Desktop/shell_scripts";

 execl(path_command, path_command, arg1, arg2, NULL); //Two times we have to
mention the path

 return 0;

}
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# gcc exec1.c -o exec1
```

```
root@ubuntu:~/Desktop/shell_scripts# ./exec1
total 176
-rwxr-xr-x 1 root root   25 Aug 14 21:18 1.script
-rwxr-xr-x 1 root root  197 Aug 15 09:46 2.sh
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap1
```

Code-2

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



/* Family or group of functions in exec  l, lp, v, vp, ve
int execl(const char *path, const char *arg, …, NULL);
int execlp(const char *file, const char *arg, …, NULL );
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execle(const char *path, const char *arg, …, NULL, char * const envp[] );
int execve(const char *file, char *const argv[], char *const envp[]);
*/
//2. execlp() System will find the path
int main(void) {
  char *just_progname = "ls"; //Instead of Entire path where the prog is present, we give command name only
  char *arg11 = "-l";
  char *arg12 = "/root/Desktop/shell_scripts";
  execlp(just_progname, just_progname, arg11, arg12, NULL); //System will find the path where the command is located
  return 0;
}
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# gcc exec2.c -o exec2
root@ubuntu:~/Desktop/shell_scripts# ./exec2
total 192
-rwxr-xr-x 1 root root   25 Aug 14 21:18 1.script
-rwxr-xr-x 1 root root  197 Aug 15 09:46 2.sh
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap1
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap2
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap3
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap4
```

Code3:

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

 /* Family or group of functions in exec  l, lp, v, vp, ve
int execl(const char *path, const char *arg, …, NULL);
int execlp(const char *file, const char *arg, …, NULL );
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execle(const char *path, const char *arg, …, NULL, char * const envp[] );
int execve(const char *file, char *const argv[], char *const envp[]);
*/
//Here the Arguments are passed in an array of string pointers
int main(void) {
  char *prog_path = "/bin/ls";
  char *args[] = {prog_path, "-l", "/root/Desktop/shell_scripts", NULL};
  execv(prog_path, args); //Takes arguments in Array of String pointers
  return 0;
}
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# gcc exec3.c -o exec3
root@ubuntu:~/Desktop/shell_scripts# ./exec3
total 208
-rwxr-xr-x 1 root root   25 Aug 14 21:18 1.script
-rwxr-xr-x 1 root root  197 Aug 15 09:46 2.sh
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap1
```

Code 4

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

 /* Family or group of functions in exec  l, lp, v, vp, ve
int execl(const char *path, const char *arg, …, NULL);
int execlp(const char *file, const char *arg, …, NULL );
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execle(const char *path, const char *arg, …, NULL, char * const envp[] );
int execve(const char *file, char *const argv[], char *const envp[]);
*/

//Here the Arguments are passed in an array of string pointers
int main(void) {
  char *prog_path = "ls";
  char *args[] = {prog_path, "-l", "/root/Desktop/shell_scripts", NULL};
   execvp(prog_path, args); //Takes arguments in Array of String pointers
   return 0;
}
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# gcc exec4.c -o exec4
root@ubuntu:~/Desktop/shell_scripts# ./exec4
total 224
-rwxr-xr-x 1 root root   25 Aug 14 21:18 1.script
-rwxr-xr-x 1 root root  197 Aug 15 09:46 2.sh
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap1
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap2
```

Code 5:

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

 /* Family or group of functions in exec  l, lp, v, vp, ve
int execl(const char *path, const char *arg, …, NULL);
int execlp(const char *file, const char *arg, …, NULL );
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execle(const char *path, const char *arg, …, NULL, char * const envp[] );
int execve(const char *file, char *const argv[], char *const envp[]);
*/

//Here This takes Path to your command or program additionally
int main(void) {
  char *prog_path = "/bin/ls";
  char *arg1 = "-l";
  char *arg2= "/root/Desktop/shell_scripts";
  char *env[] = {"PATH=/bin/ls", NULL};
 // execle(prog_path, prog_path, arg1,arg2,NULL,env);
  execle(prog_path, "ls", arg1,arg2,NULL,env);

  return 0;
}
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# gcc exec5.c -o exec5
root@ubuntu:~/Desktop/shell_scripts# ./exec5
total 240
-rwxr-xr-x 1 root root   25 Aug 14 21:18 1.script
-rwxr-xr-x 1 root root  197 Aug 15 09:46 2.sh
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap1
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap2
```

Code 6

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

 /* Family or group of functions in exec  l, lp, v, vp, ve
int execl(const char *path, const char *arg, ..., NULL);
int execlp(const char *file, const char *arg, ..., NULL );
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execle(const char *path, const char *arg, ..., NULL, char * const envp[] );
int execve(const char *file, char *const argv[], char *const envp[]);
*/
```

```
//Here This takes Path to your command or program additionally in Array
int main(void) {
  char *prog_path = "/bin/ls";
  char *arr[]={"ls","-l","/root/Desktop/shell_scripts",NULL};
  char *env[] = {"PATH=/bin/ls", NULL};
  execve(prog_path, arr,env);

  return 0;
}
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# gcc exec6.c -o exec6
root@ubuntu:~/Desktop/shell_scripts# ./exec6
total 256
-rwxr-xr-x 1 root root   25 Aug 14 21:18 1.script
-rwxr-xr-x 1 root root  197 Aug 15 09:46 2.sh
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap1
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap2
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap3
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap4
-rw-r--r-- 1 root root    8 Aug 15 17:58 Chap5
-rwxr-xr-x 1 root root 8656 Aug 15 20:55 exec1
```

/////////////////////Wait System Call in C///////////////

Code 1

```
// Parent Waits until Child is terminated
#include<stdio.h>
#include<stdlib.h>  //exit defined
#include<sys/wait.h> //fork defined
#include<unistd.h>
```

```
 int main()
{
   pid_t pid;
   pid=fork();
  if(pid<0)
     {
           printf("Child Process cannot be created\n");
           return 1;
           }
  else if(pid== 0)
     {
     printf("Child Process =%d\n",getpid());
     exit(0);          /* terminate child */
     //Dont have to press ctrl+c
     }
  else
           {
   pid = wait(NULL); /* reaping parent */
   printf("Parent pid = %d\n", getppid());
   printf("Child pid = %d\n",pid );
 }
   return 0;
}
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# gcc wait_exit1.c -o wait_exit1
root@ubuntu:~/Desktop/shell_scripts# ./wait_exit1
Child Process =18673
Parent pid = 17071
Child pid = 18673
```