1st myexecuter

`man 2 execve`

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char ** argv)
{
    if (argc < 2)
    {
        printf("Usage: %s path-to-elf\n", argv[0]);
        exit(-1);
    }

    getchar();  // used only to debug by ps in another terminal

    char* newargv[] = {argv[1], NULL};
    char* newenvp[] = {NULL};

    execve(argv[1], newargv, newenvp);
    printf("Exec failed, kernel is not the mode of executing programs\n");
    return -2;
}
ubun
```

```
./myexecuter /usr/bin/ls

executer.c   fork_demo2.c  myFemtoShell.c  print_args_env.c   simple_shell.c
fork_demo.c  fork_demo3.c  myexecuter      redirect_stderr.c  zombie_demo.c
```

Creating New Process Fork

What Pids

```
#include <stdio.h>
#include <unistd.h>
```

```
int main()
{
    getchar(); // only for debuging

    pid_t pid = fork();
    if (pid > 0)
    {
        printf("PARENT: my pid = %d, my child pid = %d\n", getpid(), pid);
    }
    else if (pid == 0)
    {
        printf("CHILD: my pid = %d, my parent pid = %d\n", getpid(),
getppid());
    }
    else
    {
        printf("PARENT: failed to fork\n");
    }

    getchar();
    return 0;
}
```

Notice the Numbers of pids

```
./fdemo1

PARENT: my pid = 3673, my child pid = 3675
CHILD: my pid = 3675, my parent pid = 3673
```

```
ps -lt pts/0
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    3360    3353  0  80   0 -  2812 do_wai pts/0    00:00:00 bash
0 S  1000    3733    3360  0  80   0 -   624 wait_w pts/0    00:00:00 fdemo1
1 S  1000    3736    3733  0  80   0 -   624 n_tty_ pts/0    00:00:00 fdemo1
```

Notice that u will need 2 chars to close parent and child processes

Continue Debugging

by ps -lt pts/0

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    getchar();

    pid_t pid = fork();
    if (pid > 0)
    {
        while(1) {
                printf("PARENT: my pid = %d, my child pid = %d\n", getpid(),
pid);
            usleep(500000);
        }
    }
    else if (pid == 0)
    {
        while(1) {
            printf("CHILD: my pid = %d, my parent pid = %d\n", getpid(),
getppid());
            usleep(500000);
        }
    }
    else
    {
        printf("PARENT: failed to fork\n");
    }

    getchar();
    return 0;
}
```

Notice that both are independent process so my be scheduled by any way

```
CHILD: my pid = 3760, my parent pid = 3759
PARENT: my pid = 3759, my child pid = 3760
CHILD: my pid = 3760, my parent pid = 3759
PARENT: my pid = 3759, my child pid = 3760
PARENT: my pid = 3759, my child pid = 3760
CHILD: my pid = 3760, my parent pid = 3759
CHILD: my pid = 3760, my parent pid = 3759
```

```
PARENT: my pid = 3759, my child pid = 3760
```

Memory Sharing Areas

Notice That memory is similar before fork then new stack heap and bss per each process is created after fork

```
#include <stdio.h>
#include <unistd.h>

int x = 5;
int y;

int main()
{
    int z = 10;
    getchar();

    x++; y++; z++;
    printf("PARENT: before fork x= %d, y=%d, z=%d\n", x, y, z);
    pid_t pid = fork();
    if (pid > 0)
    {
        while(1) {
                printf("PARENT: my pid = %d, my child pid = %d\n", getpid(),
pid);
            printf("PARENT: after fork x= %d, y=%d, z=%d\n", x, y, z);
            x++; y++; z++;
            usleep(500000);
        }
    }
    else if (pid == 0)
    {
        while(1) {
            printf("CHILD: my pid = %d, my parent pid = %d\n", getpid(),
getppid());
            printf("CHILD: after fork x= %d, y=%d, z=%d\n", x, y, z);
            x++; y++; z++;
            x++; y++; z++;
            x++; y++; z++;
            usleep(500000);
        }
    }
    else
    {
        printf("PARENT: failed to fork\n");
    }

    getchar();
    return 0;
}
```

Notice Numbers increasing

```
PARENT: before fork x= 6, y=1, z=11
CHILD: my pid = 3808, my parent pid = 3807
CHILD: after fork x= 6, y=1, z=11
PARENT: my pid = 3807, my child pid = 3808
PARENT: after fork x= 7, y=2, z=12
CHILD: my pid = 3808, my parent pid = 3807
CHILD: after fork x= 9, y=4, z=14
PARENT: my pid = 3807, my child pid = 3808
PARENT: after fork x= 8, y=3, z=13
CHILD: my pid = 3808, my parent pid = 3807
...
CHILD: my pid = 3808, my parent pid = 3807
CHILD: after fork x= 483, y=478, z=488
PARENT: my pid = 3807, my child pid = 3808
PARENT: after fork x= 165, y=160, z=170
```