

```
pod1-1:~$ cat main1.c
// simple shell example using fork() and exec()

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

int main()
{
    pid_t k;
    int status;

    while(1) {
        printf(stdout, "[%d]$ ", getpid());
        getchar();

        k = fork();
        if (k==0) {
            // child code
            printf(stdout, "I'm the child: %d\n", getpid());
            execl("/bin/ls", "/bin/ls", NULL);
        }
        else {
            // parent code
            printf(stdout, "I'm the parent: %d\n", getpid());
            waitpid(k, &status, 0);
        }
    }
}
```

process ID (process name)

system call

Linux (system OS)

blocking call  
wait for  
1 character

some non-blocking  
call  
Not wait

always check if get

input one  
character 'RET'

Input two  
characters  
'a' 'RET'

server run  
twice!

pid\_t : not a new type

typedef unsigned int pid\_t

↓  
in the header file <unistd.h>

man waitpid()

include <sys/types.h>

<wait.h>

the order is determined by OS.

```
else {  
    // parent code  
    fprintf(stdout, "I'm the parent\n", getpid());  
    waitpid(k, &status, 0);  
}  
}  
}  
"main1.c" 31L, 508C written  
pod1-1 49 % !gc  
gcc main1.c  
pod1-1 50 % ./a.out  
[30049]$  
I'm the parent: 30049  
I'm the child: 30054  
a.out main1.c main.c  
[30049]$
```

parent process run first  
child second  
getchar()  
wait  
光标 blink

```
// simple shell example using fork() and exec()

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

int main()
{
    pid_t k;
    int status;

    while(1) {
        fprintf(stdout, "[%d]$ ", getpid());
        getchar();

        k = fork();
        if (k==0) {
            // child code
            fprintf(stdout, "I'm the child: %d\n", getpid());
            exec("bin/ls", "bin/ls", NULL);
        }
        else {
            // parent code
            fprintf(stdout, "I'm the parent: %d\n", getpid());
            waitpid(k, &status, 0);
        }
    }
}

pod1-1 13 % man getpi
```

exec() → execve() system call  
loader  
not system call  
fprintf(stdout, "fork fail");  
}

daemon

/d i : m a n /

Server runs all the time

canonical mode

type %ls (RET)

3 character

until hit (RET), APP will not see "ls"

raw mode

game keyboard

no buffer

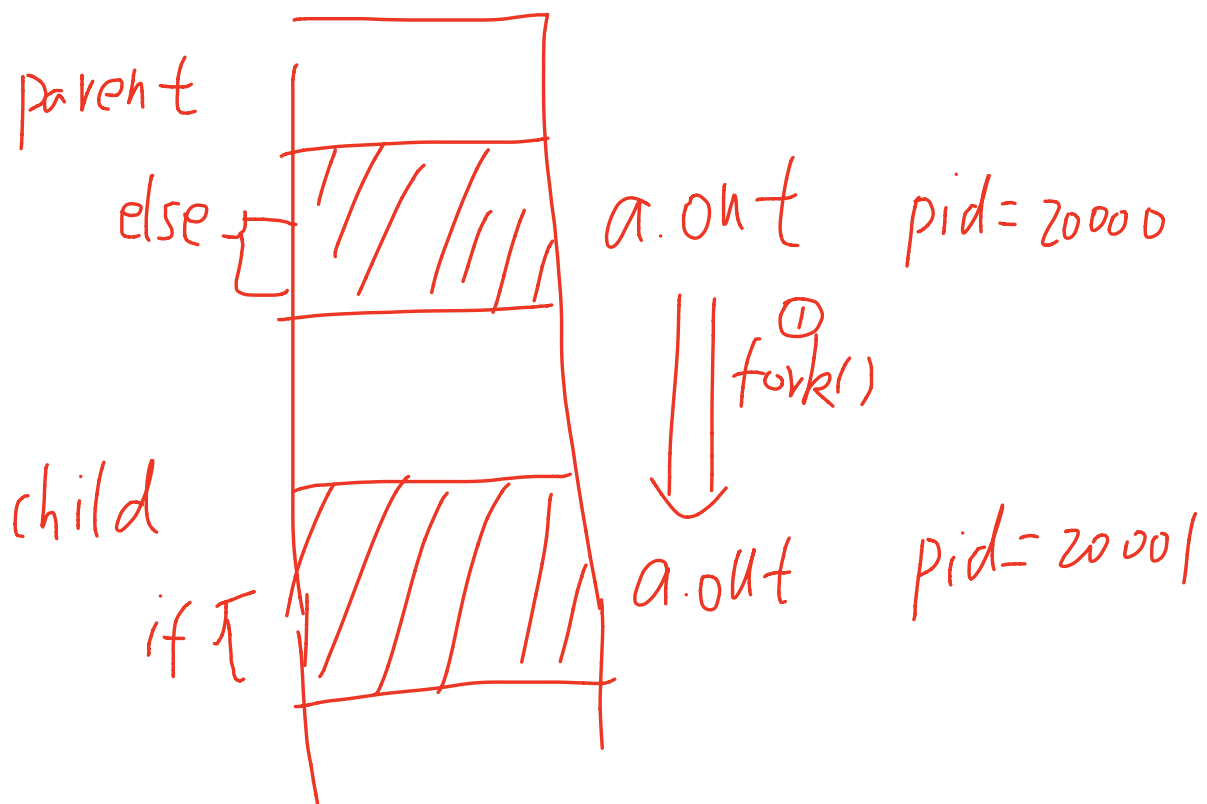
Same

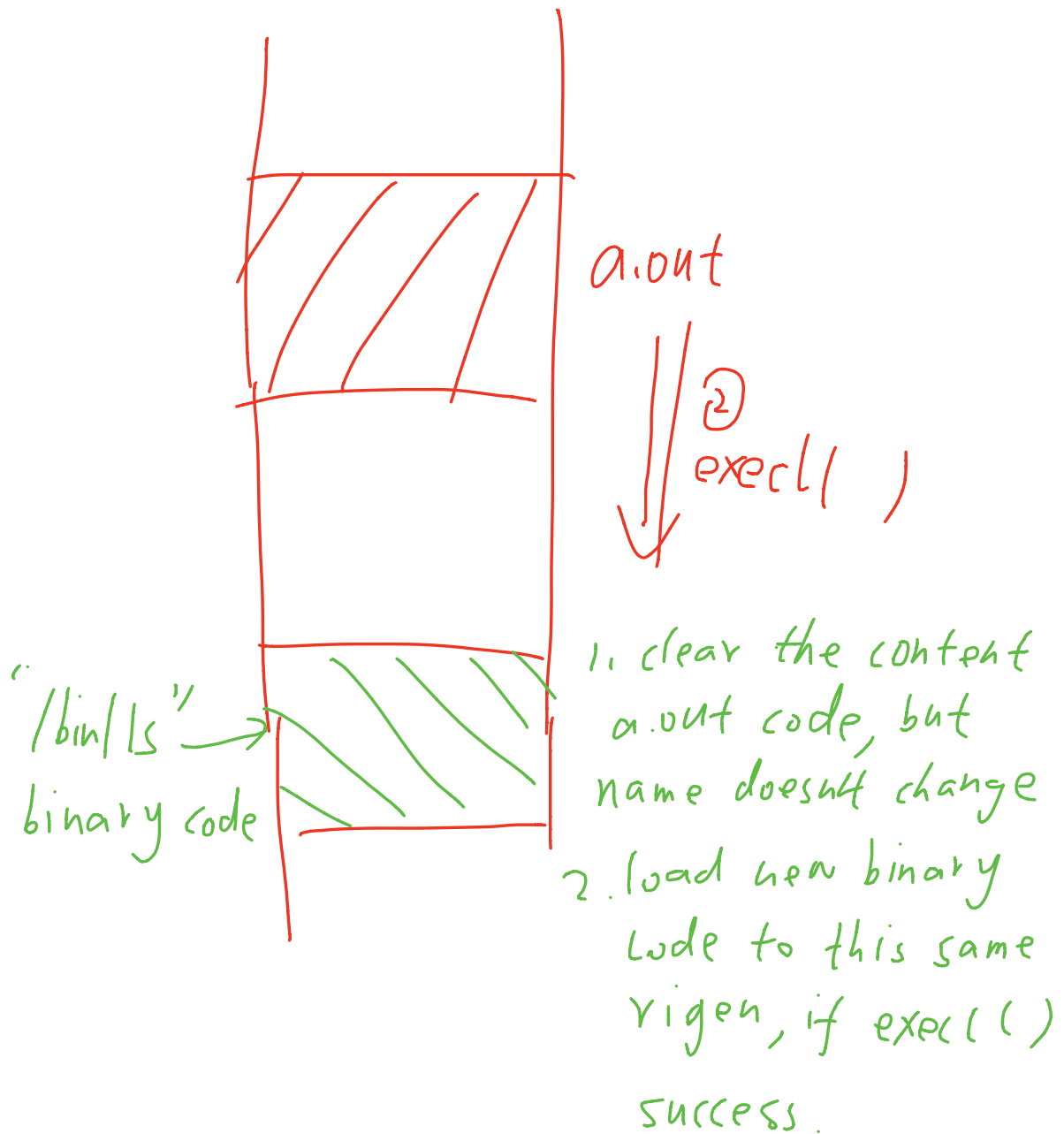
`createProcess()` → windows

`clone()` → Unix

`fork()` → Linux

---





if not exit, child will run the same code as pare

if exec(1) fail, it will return back to parent code  
need -exit(1) the child process