前台和后台

2023年4月25日 9:25

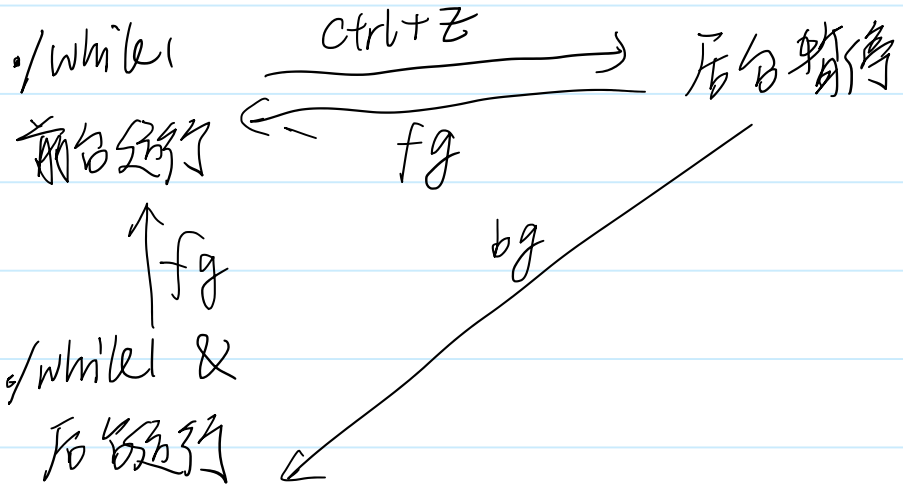
```
$ sudo apt install manpages-posix-dev
```

bash

前台：可以响应键盘中断的进程（ctrl+C）

后台：不可以响应……

jobs 列出所有进程



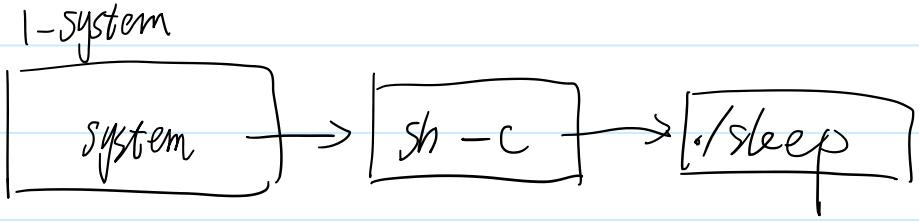
执行一次可执行程序，创建多个进程

2023年4月25日 10:12

```
NAME
       system - execute a shell command

SYNOPSIS
       #include <stdlib.h>

       int system(const char *command);
```

创建一个子进程，执行 command 命令.

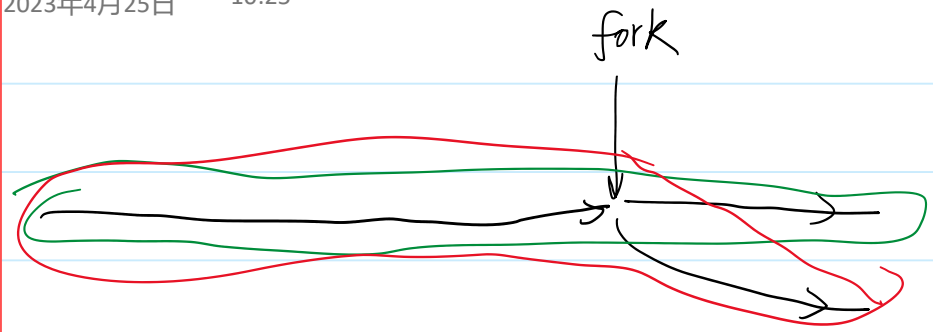
```
0 S liao      28447   25732  0  80   0 -   591 do_wai 10:17 pts/4    00:00:00 ./1_system
0 S liao      28448   28447  0  80   0 -   654 do_wai 10:17 pts/4    00:00:00 sh -c ./sleep
0 S liao      28449   28448  0  80   0 -   624 hrtime 10:17 pts/4    00:00:00 ./sleep
```




```c
#include <49func.h>
int main()
{
    //system("ls");
    //system("./sleep");
    system("python3 hello.py");
    return 0;
}
```

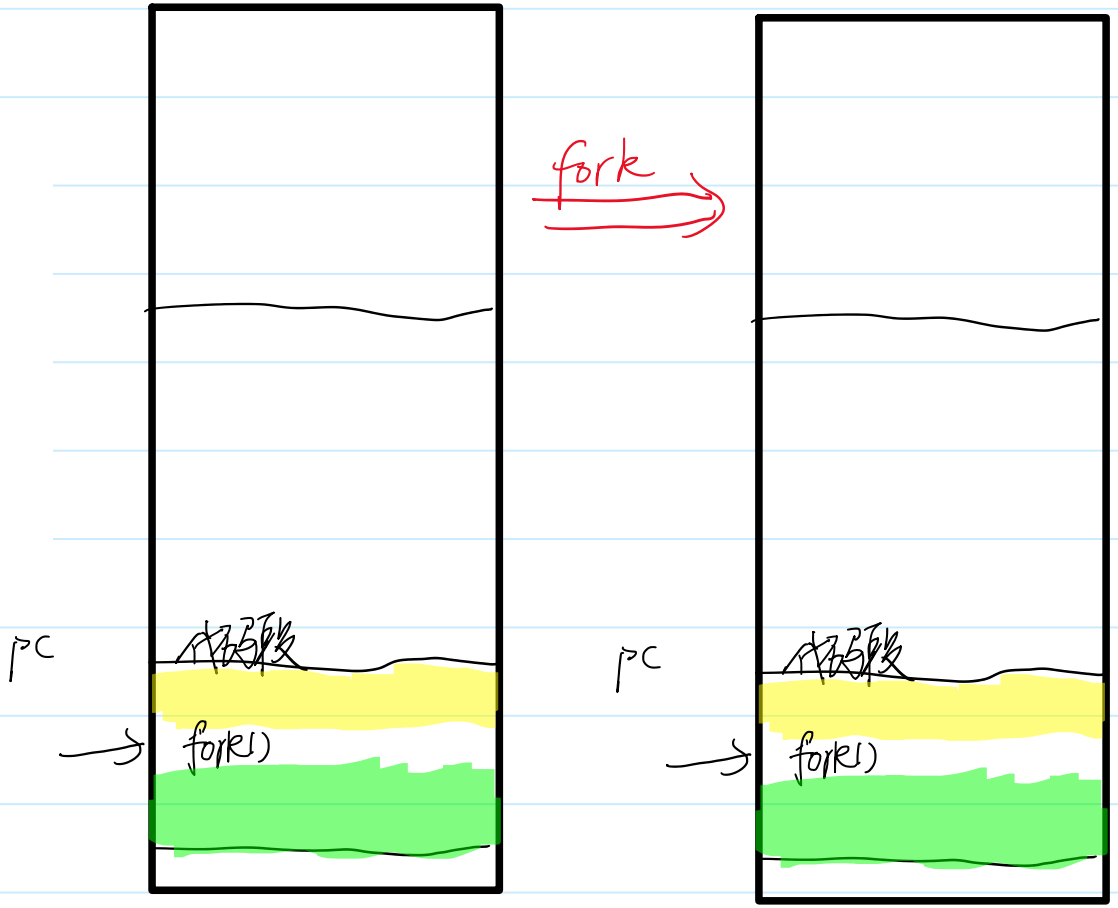
fork

2023年4月25日 10:25



复制父进程 创建子进程

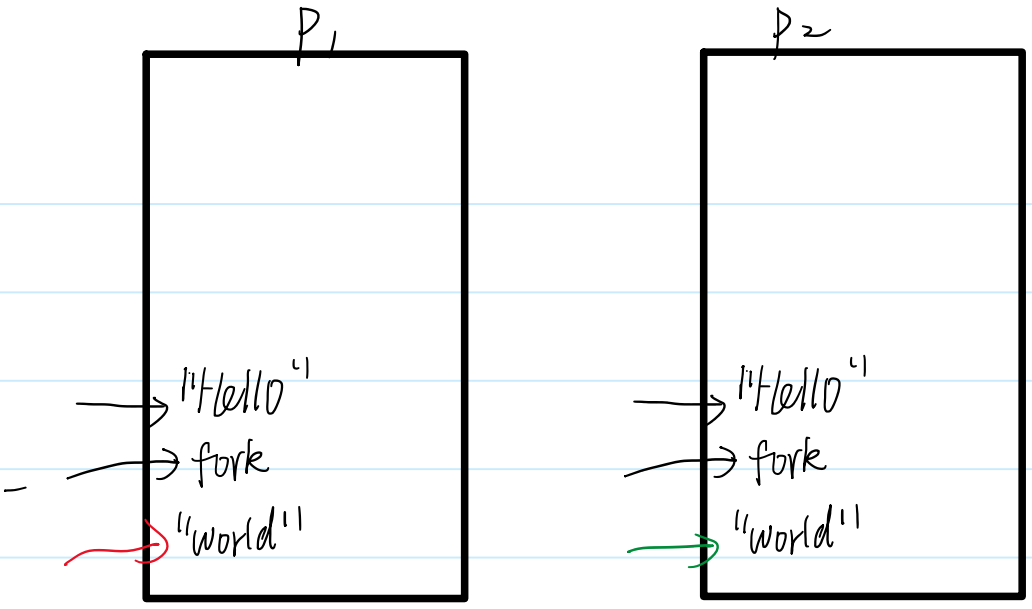
```
NAME
       fork - create a child process

SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

       pid_t fork(void);
```



fork的执行效果

2023年4月25日 11:10

```c
#include <49func.h>
int main()
{
    printf("Hello\n");
    fork();
    printf("World\n");
    sleep(20);
    return 0;
}
```



父子进程执行不同的代码

2023年4月25日 11:13

On success, the PID of the child process is returned in the parent, and 0 is returned in the child.

fork的返回值+if

父进程：子进程的PID，非0

子进程：返回0.

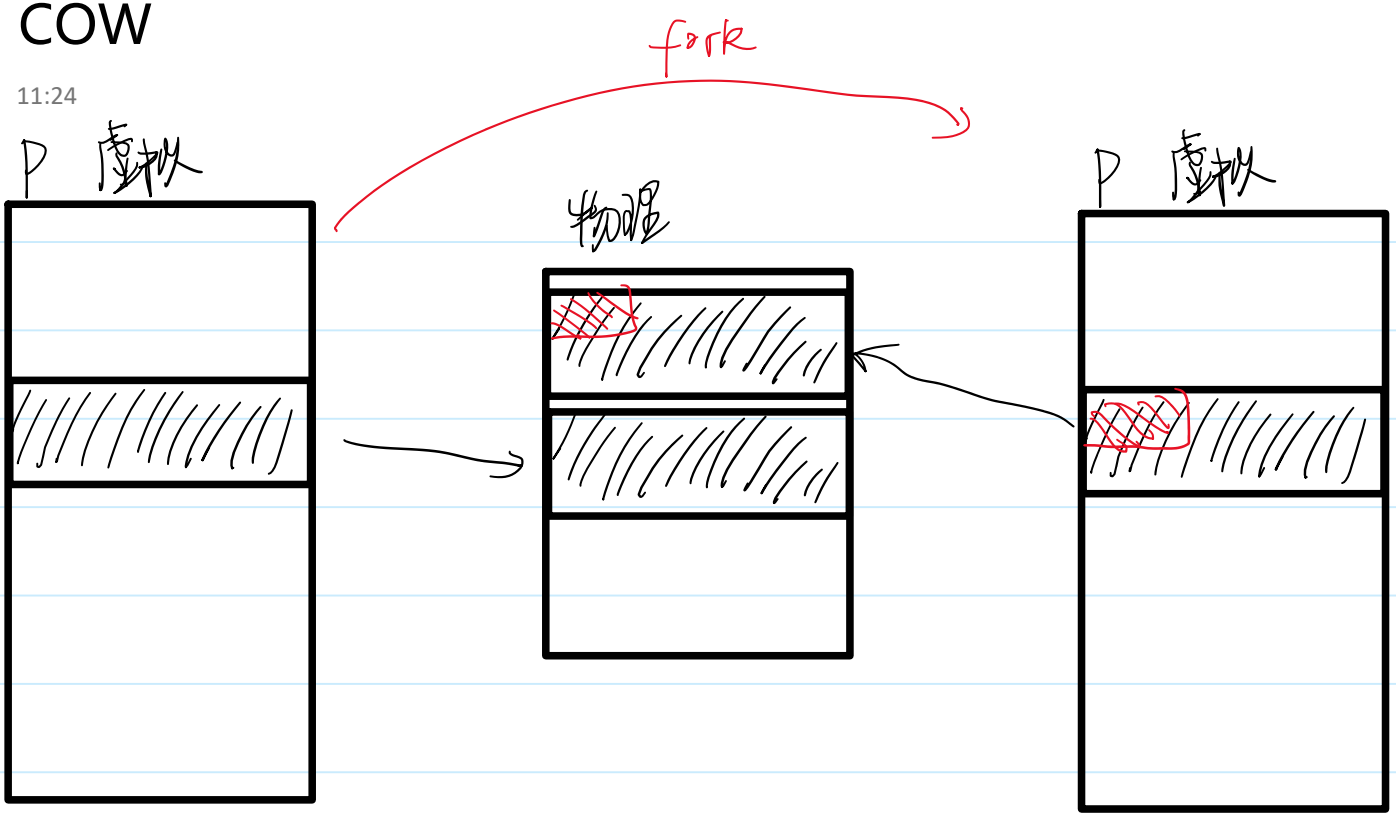
```c
#include <49func.h>
int main()
{
    printf("Hello\n");
    pid_t pid;
    pid = fork();
    if(pid != 0){
        //父进程
        printf("I am parent, pid = %d, ppid = %d\n",getpid(),getppid());
        sleep(1);
    }
    else{
        //子进程
        printf("I am child, pid = %d,ppid = %d\n", getpid(),getppid());
    }
    return 0;
}
```

```
[liao@ubuntu Linuxday_11]$ ./3_fork
Hello
I am parent, pid = 29073, ppid = 25630
I am child, pid = 29074,ppid = 29073
```

子并发.

写时复制 COW

2023年4月25日 11:24



开始，父 fork 子，父和子虚拟页映射到同一个物理页

假如一直做读操作：映射关系不变

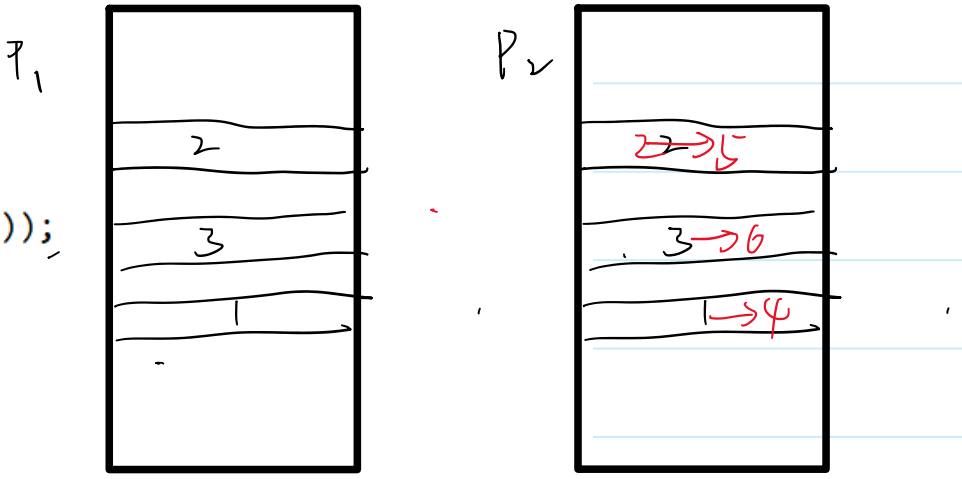
直到写操作，分配新的物理页，子进程映射新的物理页

fork的数据拷贝

2023年4月25日 11:39

4_fork_copy.c buffers

```c
#include <49func.h>
int global = 1;
int main()
{
    int stack = 2;
    int *pHeap = (int *)malloc(sizeof(int));
    *pHeap = 3;
    if(fork() == 0){
        //子进程
        global += 3;
        stack += 3;
        *pHeap += 3;
        printf("I am child, global = %d, stack = %d, heap = %d\n",
               global,stack,*pHeap);
    }
    else{
        //父进程
        sleep(2);
        printf("I am parent, global = %d, stack = %d, heap = %d\n",
               global,stack,*pHeap);
    }

    return 0;
}
```



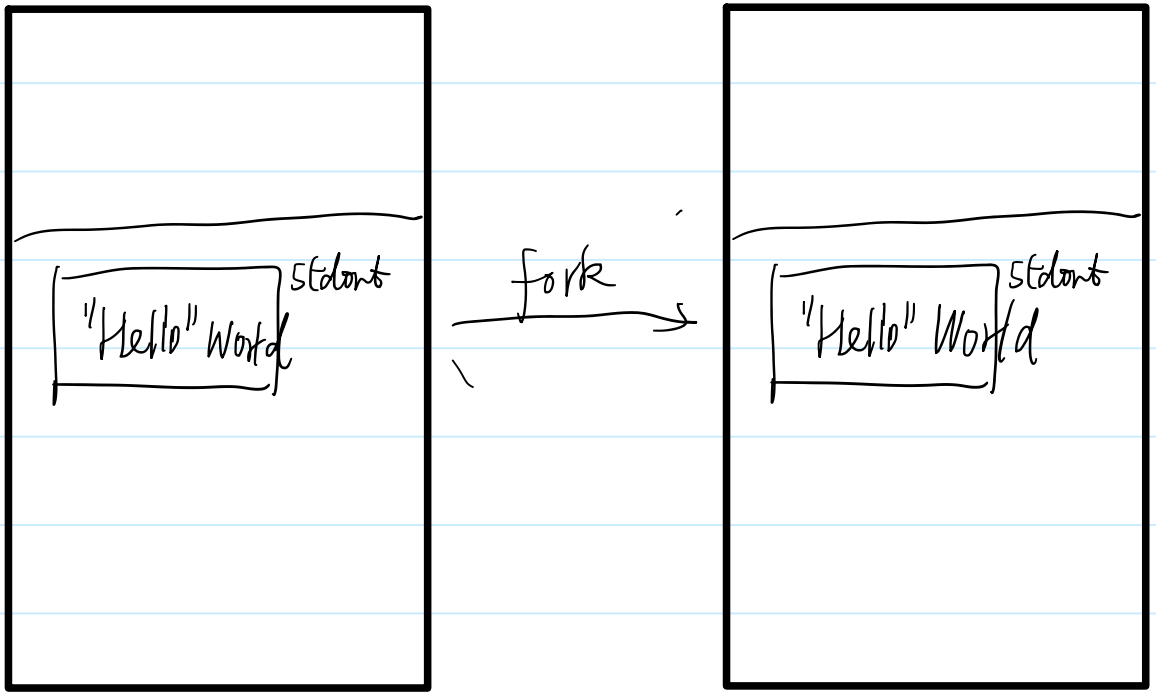
FILE在fork之后会拷贝一份

2023年4月25日 11:47

```c
int main()
{
    printf("Hello");
    fork();
    printf("World!\n");
    return 0;
}
```

printf ①拷贝到stdout

②stdout拷贝到文件缓冲区(屏幕)



一份代码

2023年4月25日 14:47

```
int main(){
   int *p = malloc(4);
   fork();
   free(p);
}
```

作业题

2023年4月25日 14:54

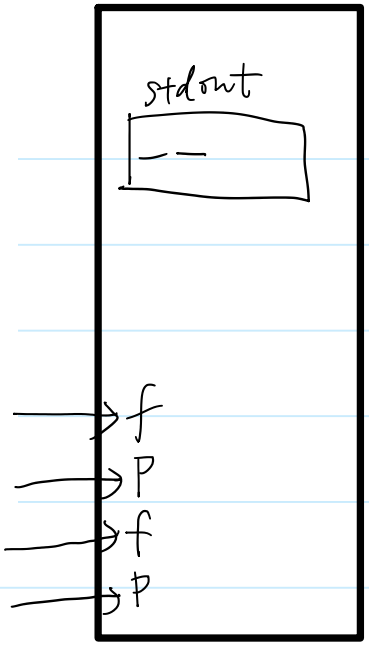
(1). 请问下面的程序一共输出多少个"-"

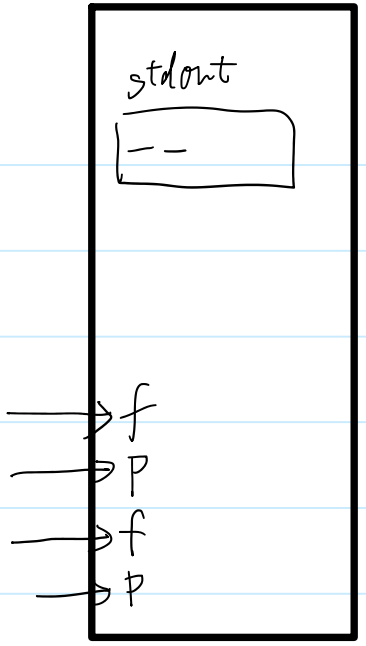
```
int main()
{
fork();
printf("-");

fork();
printf("-");
}
return 0;
}
```

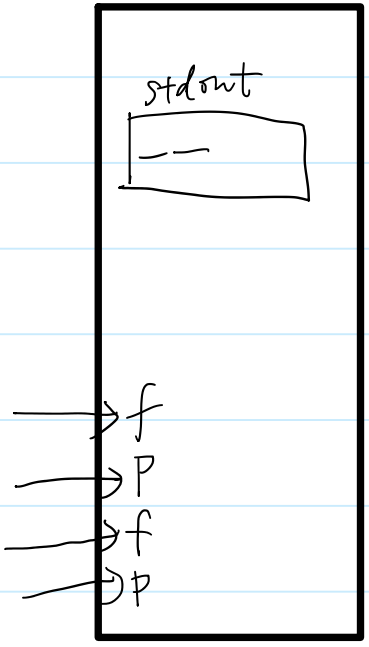
fork.习题

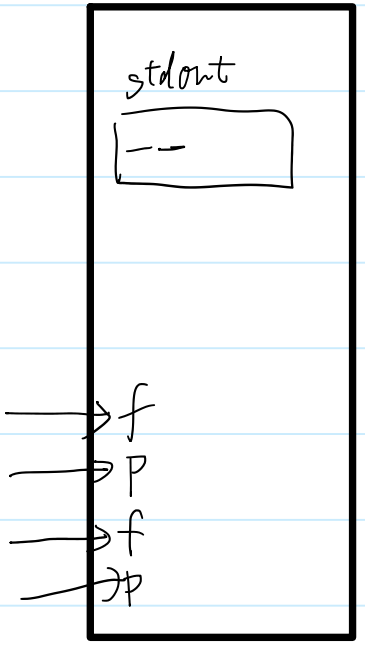
画出地址空间

清楚代码段和PC指针









fork

2023年4月25日 15:04

复制父进程 → 创建子进程

子进程地址空间内的内容是复制父进程

↓

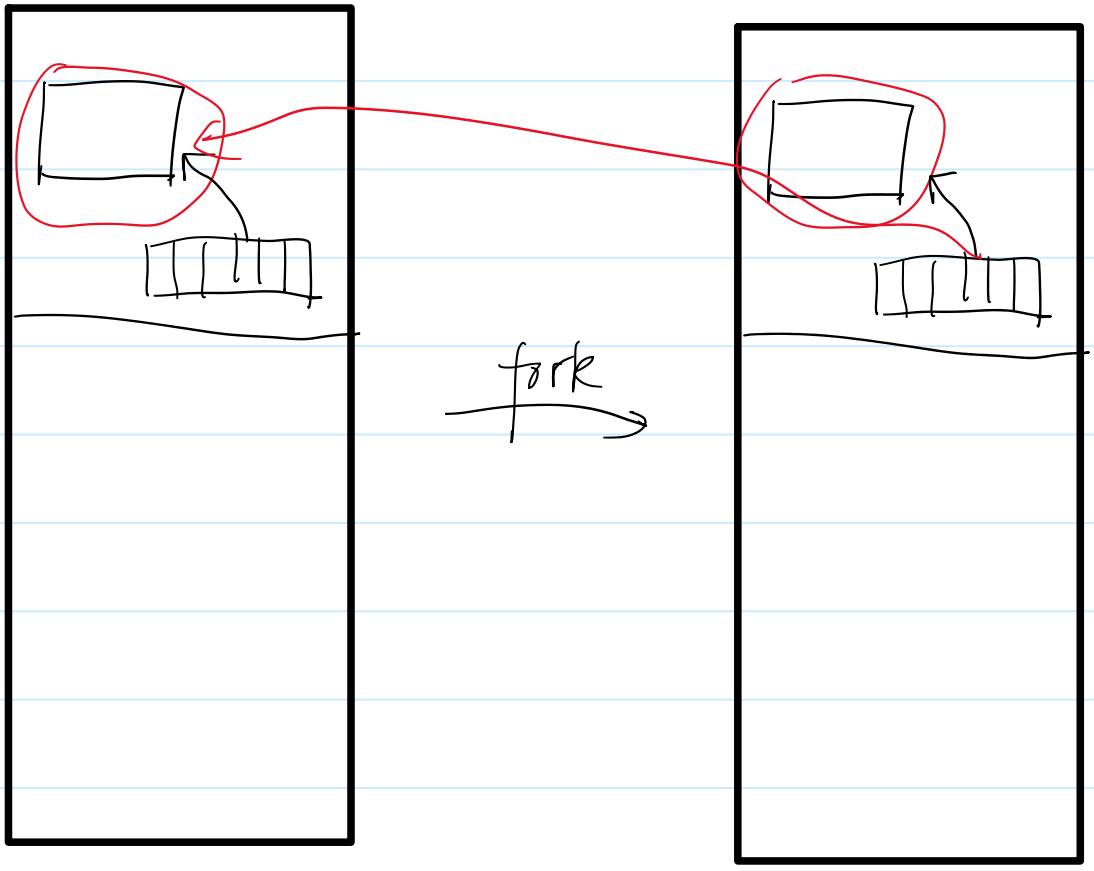
父子进程地址空间的内容各自独立的.

栈、堆、数据段、文件流

先打开文件对象，再fork

2023年4月25日 15:09

先打开文件对象，再fork



文件对象是共享的

一个进程修改，另外进程也知道。

exec函数族

2023年4月25日 15:53

让进程加载一个可执行程序文件.

l → list 参数用可变参数

v → vector 参数用数组.

```
int execl(const char *pathname, const char *arg, ...
                /* (char  *) NULL */);
int execlp(const char *file, const char *arg, ...
                /* (char  *) NULL */);
int execle(const char *pathname, const char *arg, ...
                /*, (char *) NULL, char *const envp[] */);
int execv(const char *pathname, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[],
                char *const envp[]);
```

execl

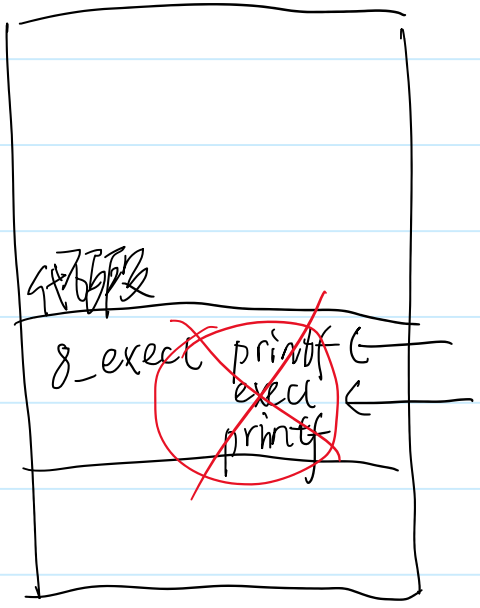
2023年4月25日 16:01

```c
int main(int argc, char *argv[])
{
    // ./add 3 4
    ARGS_CHECK(argc,3);
    printf("lhs + rhs = %d\n",  atoi(argv[1]) + atoi(argv[2]));
    return 0;
}
```

```
[liao@ubuntu Linuxday_11]$ ./add 3 4
lhs + rhs = 7
```

```c
#include <49func.h>
int main()
{
    printf("You can see me!\n");
    execl("add","./add","3","4",NULL);
    // 可执行程序的路径 命令行的各个参数
    printf("You can't see me!\n");
    return 0;
}
```

./8_execl

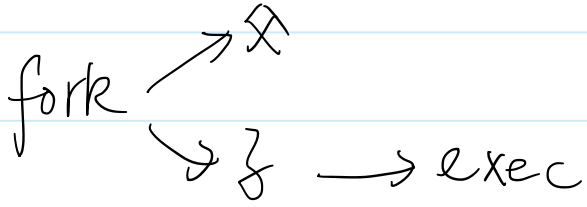


①清空栈
②用可执行程序的代码段
取代旧的代码段
③重置PC指针

CreateProcess

2023年4月25日 16:15

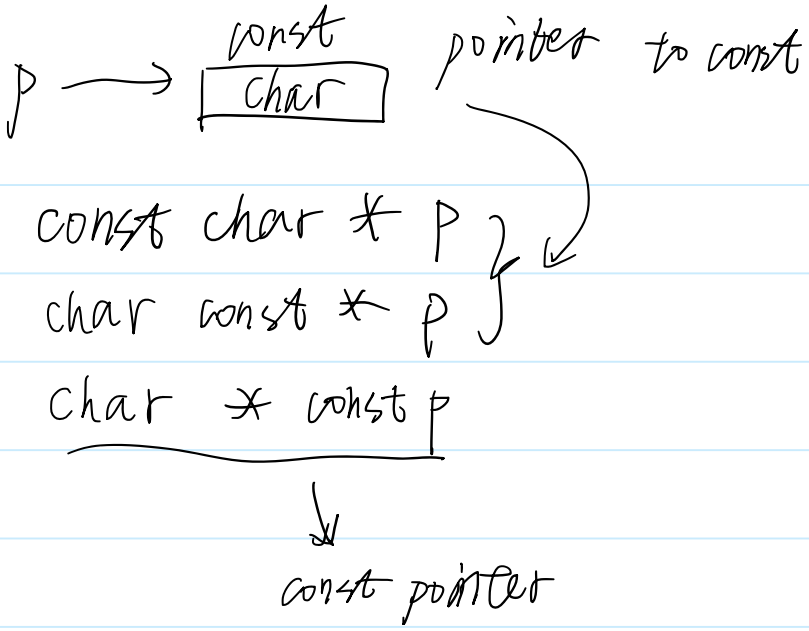
创建一个子进程，运行可执行程序的代码。



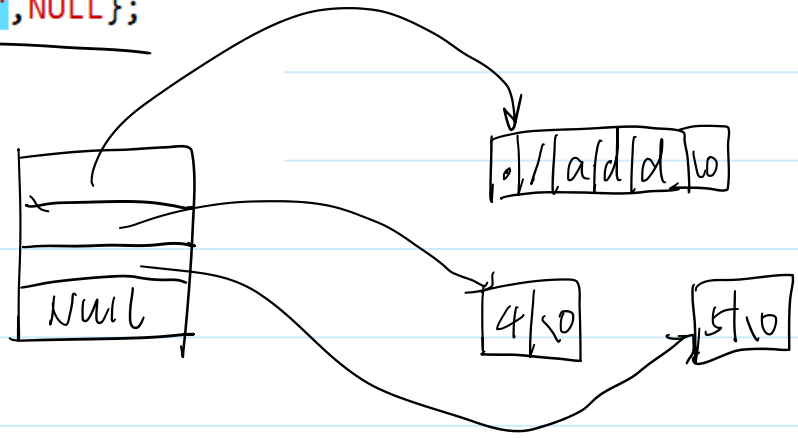
execv

2023年4月25日 16:18

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




```c
#include <49func.h>
int main()
{
    printf("You can see me!\n");
    char * const args [] = {"./add","4","5",NULL};
    execv("add",args);
    printf("You can't see me!\n");
    return 0;
}
```



如果父进程先终止

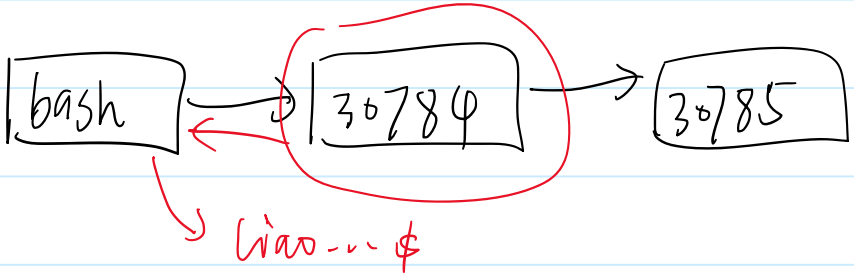
2023年4月25日 16:33

```
if(pid != 0){
    //父进程
    printf("I am parent, pid = %d, ppid = %d\n",getpid(),getppid());
    //sleep(1);
}
else{
    //子进程
    printf("I am child, pid = %d,ppid = %d\n", getpid(),getppid());
}
```

```
[liao@ubuntu Linuxday_11]$ ./10_fork
Hello
I am parent, pid = 30784, ppid = 25630
[liao@ubuntu Linuxday_11]$ I am child, pid = 30785,ppid = 1
```

① ②

父进程在子进程之前终止
子进程 ppid改为1.

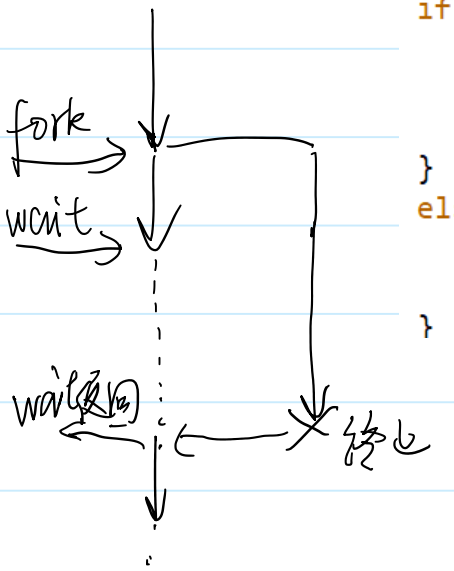


子进程终止了，资源由父进程回收

2023年4月25日 16:38

```
pid_t wait(int *wstatus);
```

等待子进程终止，并回收资源




```c
if(pid != 0){
    //父进程
    printf("I am parent, pid = %d, ppid = %d\n",getpid(),getppid());
    wait(NULL);
}
else{
    //子进程
    printf("I am child, pid = %d,ppid = %d\n", getpid(),getppid());
}
```

```
0 S liao      31222   25630  0  80   0 -   624 hrtime 17:14 pts/2    00:00:00 ./10_fork
1 Z liao      31223   31222  0  80   0 -     0 -      17:14 pts/2    00:00:00 [10_fork] <defunct>
```

wait获取进程的退出状态

2023年4月25日 17:18

WIFEXITED(wstatus)
returns true if

WEXITSTATUS(wstatus)
returns the e
call to **exit**(3)
true.

WIFSIGNALED(wstatus)
returns true if

WTERMSIG(wstatus)

```c
int main(int argc, char *argv[])
{
    if(fork() == 0){
        //return -1;
        while(1);
    }
    else{
        int wstatus;
        wait(&wstatus);
        if(WIFEXITED(wstatus)){
            printf("exit normally! exit code = %d\n", WEXITSTATUS(wstatus));
        }
        else if(WIFSIGNALED(wstatus)){
            printf("exit abnormally! terminal signal = %d\n", WTERMSIG(wstatus));
        }
    }
    return 0;
}
```

waitpid

2023年4月25日 17:31

WNOHANG return immediately if no child has exited.

非阻塞.

```
pid_t waitpid(pid_t pid, int *wstatus, int options);
```

一般配合循环使用.

-1 meaning wait for any child process.

```c
if(fork() == 0){
    //return -1;
    while(1);
}
else{
    int wstatus;
    //wait(&wstatus);
    while(1){
        int ret = waitpid(-1,&wstatus,WNOHANG);
        if(ret != 0){
            if(WIFEXITED(wstatus)){
                printf("exit normally! exit code = %d\n", WEXITSTATUS(wstatus));
            }
            else if(WIFSIGNALED(wstatus)){
                printf("exit abnormally! terminal signal = %d\n", WTERMSIG(wstatus));
            }
            break;
        }
        else{
            printf("No child process has dead yet!\n");
            sleep(3);
        }
    }
}
return 0;
```