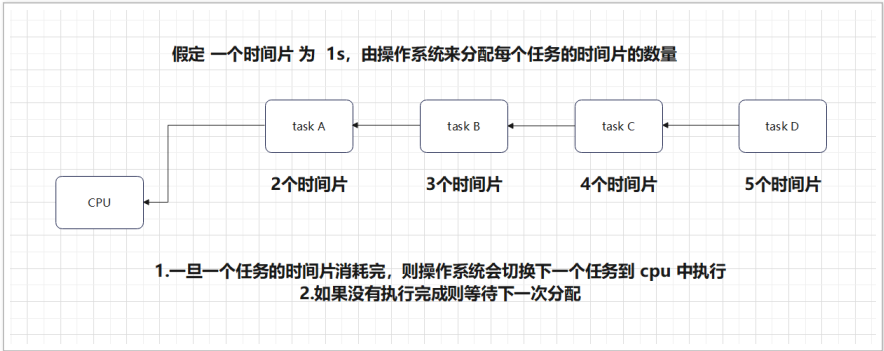


# 3.1 进程的创建\_物联网 / 嵌入式工程师 - 慕课网

“ 慕课网慕课教程 3.1 进程的创建涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

## 1. 进程的创建

- 为了提高计算机执行任务的效率，一般采用的解决方案就是能够让多个任务同时进行，这里可以使用  
并发  
与  
并行 两种方式
  - 并行：在 cpu 多核的支持下，实现物理上的同时执行
  - 并发：在有限的 cpu 核心的情况下（如只有一个 cpu 核心），利用快速交替（时间片轮转）执行来达到宏观上的同时执行



- 并行是基于硬件完成，而并发则可以使用软件算法来完成，在完成任务时，可以创建多个进程并发执行
- 创建进程的函数需要调用 `fork()` 函数，则会产生一个新的进程
- 调用 `fork()` 函数的进程叫做 父进程，产生的新进程则为 子进程
- `fork` 函数详解

## 函数头文件

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

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

## 函数原型

```
pid_t fork(void);
```

## 函数功能

创建一个子进程

## 函数返回值

成功：返回给父进程是子进程的 pid，返回给子进程的是 0

失败：返回 -1，并设置 `errno`

示例：创建一个子进程，并打印 HelloWorld

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

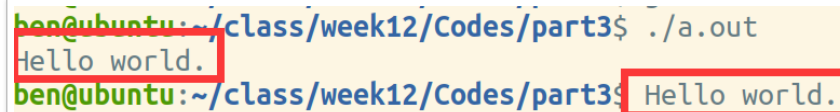
int main(void)
{
    pid_t cpid;

    cpid = fork();

    if (cpid == -1){
        perror("fork(): ");
        return -1;
    }

    printf("Hello world.\n");

    return 0;
}
```



```
ben@ubuntu: ~/class/week12/Codes/part3$ ./a.out
Hello world.
ben@ubuntu: ~/class/week12/Codes/part3$ Hello world.
```

- 之所以 显示两个“helloworld” 是因为打印语句在两个进程中都运行了
- 示例：创建一个子进程，并打印 父进程与子进程的 pid

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

static int idata = 200;

int main(void)
{
    pid_t pid;

    int istack = 300;

    pid = fork();

    if (pid < 0)
```

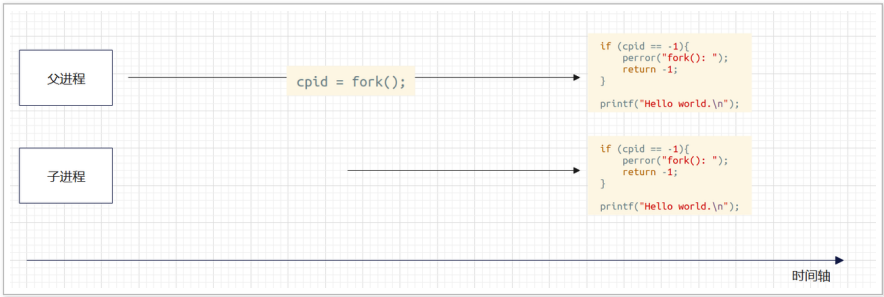
```
{
    perror("fork():");
    exit(-1);
}

else if (pid == 0)
{
    idata *= 2;
    istack *= 3;
}

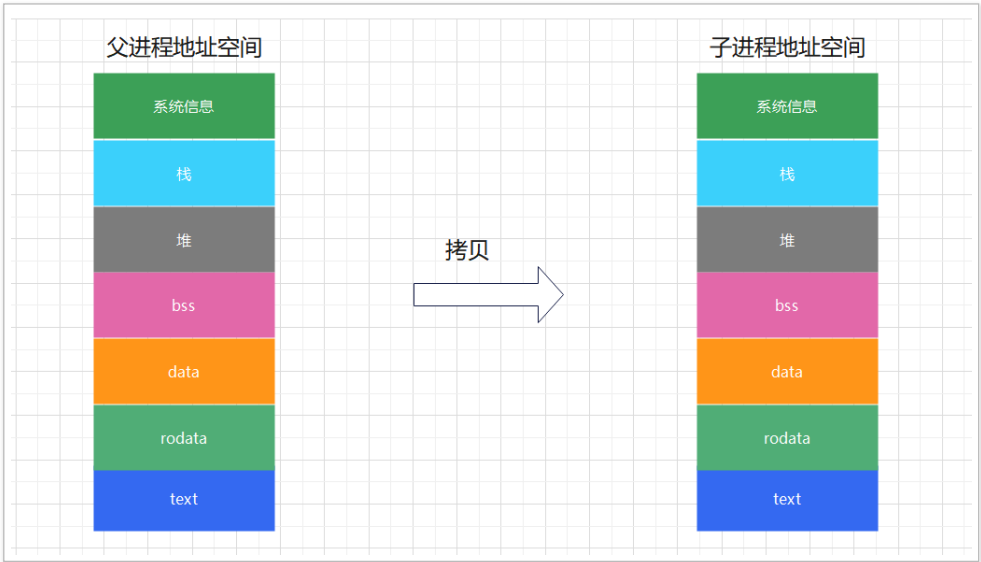
printf(" %s %d %d\n", (pid == 0)?("child"):(("parent")), idata, istack);

return 0;
}
```

- 通过 fork() 函数创建子进程之后，有如下特点:
  - 父子进程并发执行，子进程从 fork() 之后开始执行



- 父子进程的**执行顺序由操作系统算法决定的，不是由程序本身决定**
- 子进程会**拷贝父进程地址空间的内容，包括缓冲区、文件描述符等**



```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>
```

```
int main(void)
{
    pid_t cpid;

    write(STDOUT_FILENO, "Hello", 6);

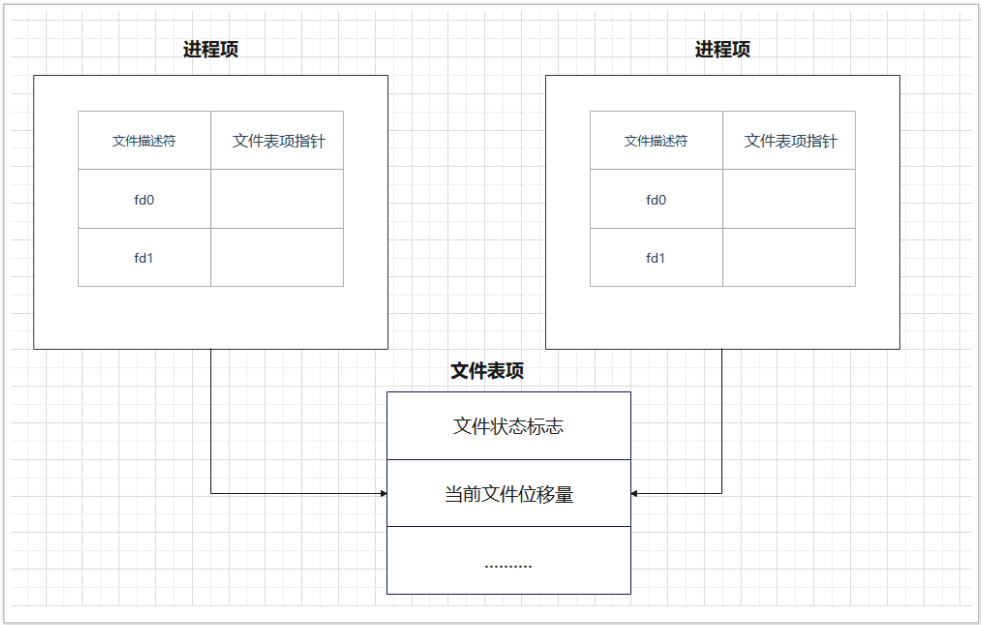
    fputs("Hello", stdout);

    cpid = fork();

    if (cpid == -1){
        perror("fork(): ");
        return -1;
    }

    return 0;
}
```

- 文件描述符拷贝
  - 每个进程都会维护一个文件表项，即文件描述符与文件指针的映射表
  - 在 Linux 内核中有一个 struct file 结构体来管理所有打开的文件
  - 当子进程拷贝了父进程文件描述符后，则会共享文件状态标志与文件偏移量等信息



```
struct file {
    union {
        struct llist_node    fu_llist;
        struct rcu_head      fu_rcuhead;
    } f_u;
    struct path              f_path;
    struct inode              *f_inode;    /* cached value */
    const struct file_operations *f_op;

    /*
     * Protects f_ep_links, f_flags.
     * Must not be taken from IRQ context.
     */
    spinlock_t               f_lock;
    atomic_long_t            f_count;
    unsigned int              f_flags;
    fmode_t                   f_mode;
    struct mutex              f_pos_lock;
    loff_t                    f_pos;
    struct fown_struct        f_owner;
    const struct cred         *f_cred;
};
```



练习：创建一个子进程，并定义一个全局变量 `global = 0`，在子进程中修改值为 100，在父子进程打印 `global` 的值，思考为什么是这样的结果

---

全文完

本文由 简悦 SimpRead 优化，用以提升阅读体验

使用了 全新的简悦词法分析引擎 beta，点击查看详细说明

