作业

2023年8月8日 9:55

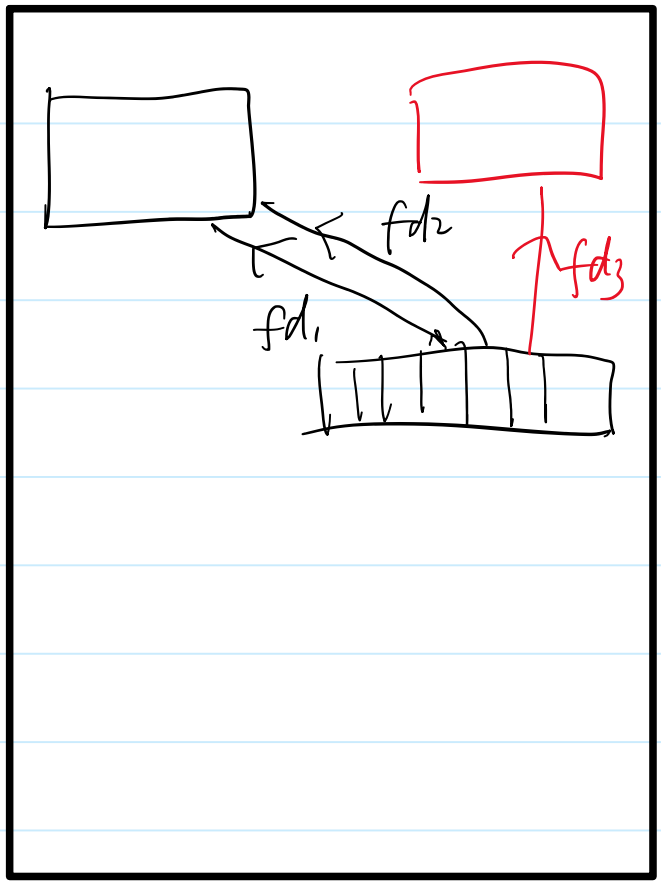
如果fd1和fd2共享文件对象共享偏移量.

fd1和fd3打开的同一个磁盘文件.

```
fd1 = open(path, oflags);
fd2 = dup(fd1);
fd3 = open(path, oflags);
```

write(fd1, "hello", 5)

write(fd2, "world", 5)



2023年8月8日 10:06

```c
int main(int argc, char *argv[])
{
    // ./0_homework file1
    ARGS_CHECK(argc,2);
    int fd1 = open(argv[1],O_RDWR);
    int fd2 = dup(fd1);
    int fd3 = open(argv[1],O_RDWR);
    printf("fd1 = %d, fd2 = %d, fd3 = %d\n", fd1,fd2,fd3);
    write(fd1,"hello",5);
    write(fd2,"world",5);
    write(fd3,"howareyou",9);
    close(fd2);
    close(fd1);
    close(fd3);
    return 0;
}
```

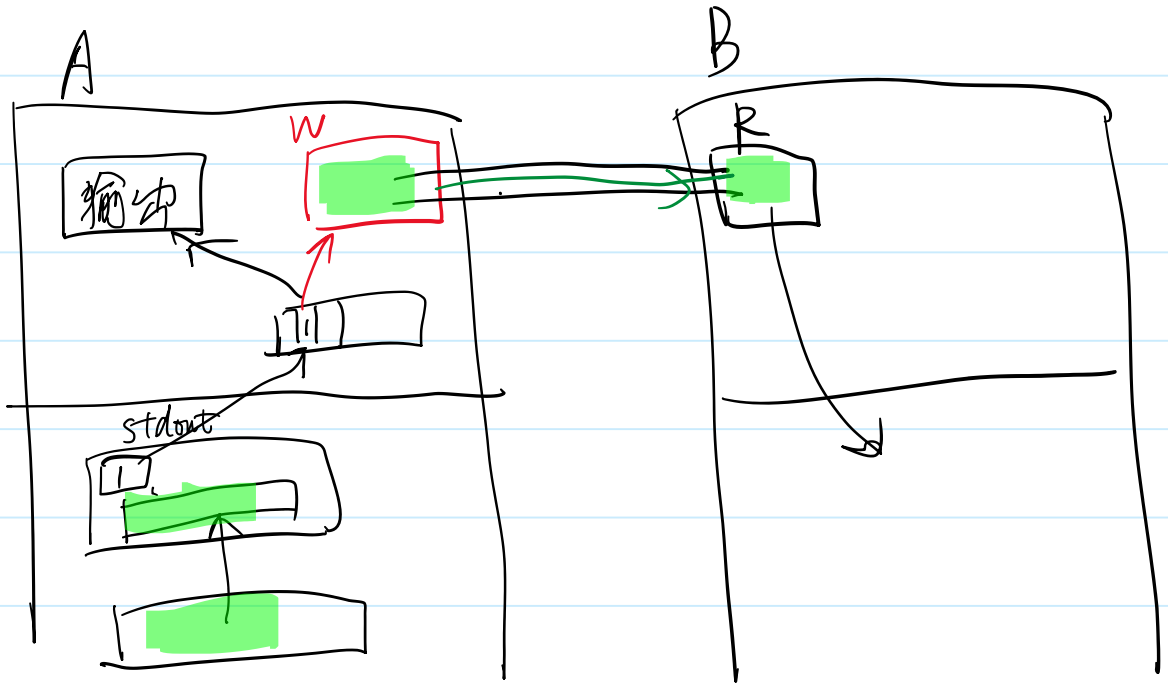
2023年8月8日 10:12

```c
int main(int argc, char *argv[])
{
    // ./1_redirect file1 file2
    ARGS_CHECK(argc,3);
    int fd1 = open(argv[1],O_WRONLY);
    int fd2 = open(argv[2],O_WRONLY);
    printf("\n");
    dup2(fd1,STDOUT_FILENO);
    dup2(fd2,STDERR_FILENO);
    printf("Hello\n");
    fprintf(stderr,"World\n");
    return 0;
}
```

避免OS的BUG.

如果close(stdout)之前，没有打印换行可能会失效.

2023年8月8日 10:14



2023年8月8日 10:24

3_A.c buffers

```c
#include <52func.h>
int main(int argc, char *argv[])
{
    // ./3_A 1.pipe
    ARGS_CHECK(argc,2);
    int fdw = open(argv[1],O_WRONLY);

    printf("1 Helloworld!\n");

    int backup_fd = 10;
    dup2(STDOUT_FILENO,backup_fd);//备份stdout设备
    dup2(fdw,STDOUT_FILENO); //让1引用管道的写端
    printf("2 Helloworld!\n");
    dup2(backup_fd,STDOUT_FILENO);//让1引用输出设备
    printf("3 Helloworld!\n");
    return 0;
}
```

3_B.c

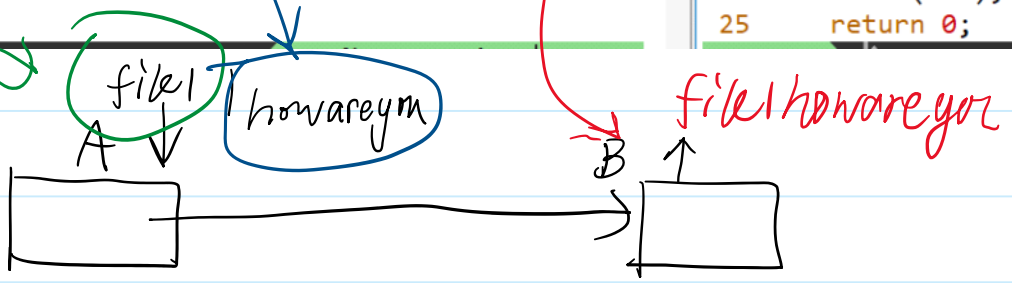
```c
#include <52func.h>
int main(int argc, char *argv[])
{
    // ./3_B 1.pipe
    ARGS_CHECK(argc,2);
    int fdr = open(argv[1],O_RDONLY);
    char buf[4096] = {0};
    read(fdr,buf,sizeof(buf));
    printf("buf = %s\n", buf);
    return 0;
}
```

管道是流式 消息之间没有边界

2023年8月8日 11:16

```c
int main(int argc, char *argv[])
{
    // ./5_A 1.pipe file1
    ARGS_CHECK(argc,3);
    // A 发送 写端
    int fdw = open(argv[1],O_WRONLY);
    int fd = open(argv[2],O_RDONLY);//fd是磁盘文>
    // 发送文件名
    write(fdw,argv[2],strlen(argv[2]));
    char buf[4096];
    while(1){
        memset(buf,0,sizeof(buf));
        ssize_t sret = read(fd,buf,sizeof(buf));
        if(sret == 0){
            break;
        }
        write(fdw,buf,sret);
    }
    close(fd);
    close(fdw);
    return 0;
}
```

```c
int main(int argc, char *argv[])
{
    // ./5_B 1.pipe
    ARGS_CHECK(argc,2);
    int fdr = open(argv[1],O_RDONLY);
    char filename[4096] = {0};
    read(fdr,filename,sizeof(filename));
    mkdir("dir1",0777);
    char filepath[8192] = {0};
    sprintf(filepath,"%s/%s","dir1",filename);
    int fd = open(filepath,O_WRONLY|O_TRUNC|O_CREAT,0666);
    char buf[4096];
    while(1){
        memset(buf,0,sizeof(buf));
        ssize_t sret = read(fdr,buf,sizeof(buf));
        if(sret == 0){
            // 写端已经断开了
            break;
        }
        write(fd,buf,sret);
    }
    close(fd);
    close(fdr);
    return 0;
}
```



除了要让A和B的数据一致
A和B对数据操作的结果
还成功了

协议

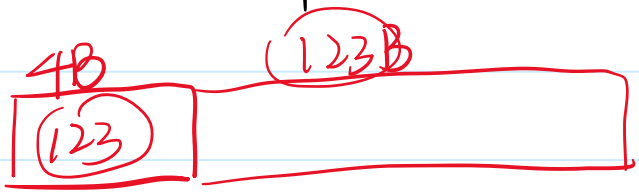
2023年8月8日 11:21

通信双方对于通信内容的约定.

发送方 按约定发
接收方 按约定解析 } → 约束

① 约定的结束符. C风格字符串.

② 固定长度的字段



"小火车协议"

redis/nginx

5_A.c

```c
#include <52func.h>
typedef struct train_s {
    int length;//4B 火车头
    char data[1000];// 1000只是上限 真正的长度由length决定
} train_t;
int main(int argc, char *argv[])
{
    // ./5_A 1.pipe file1
    ARGS_CHECK(argc,3);
    // A 发送 写端
    int fdw = open(argv[1],O_WRONLY);
    int fd = open(argv[2],O_RDONLY);//fd是磁盘文件
    train_t train;
    // 发送文件名
    train.length = strlen(argv[2]);
    memcpy(train.data,argv[2],train.length);
    write(fdw,&train.length,sizeof(int));
    write(fdw,train.data,train.length);

    while(1){
        memset(train.data,0,sizeof(train.data));
        ssize_t sret = read(fd,train.data,sizeof(train.data));
        train.length = sret;
        if(sret == 0){
            break;
        }
        write(fdw,&train.length,sizeof(int));
        write(fdw,train.data,train.length);
    }
    train.length = 0;
    write(fdw,&train.length,sizeof(int));
    sleep(300);
    close(fd);
    close(fdw);
    return 0;
}
```

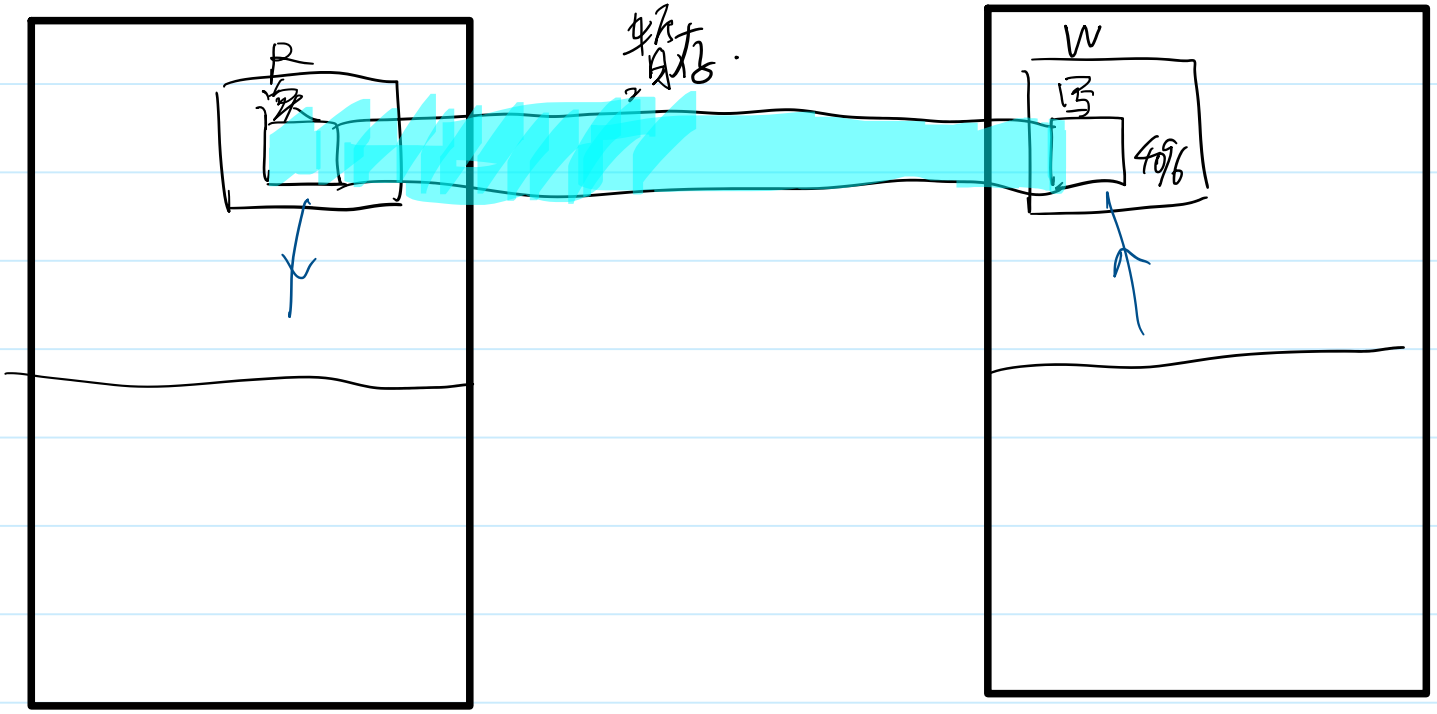
5_B.c

```c
#include <52func.h>
typedef struct train_s {
    int length;
    char data[1000];
} train_t;
int main(int argc, char *argv[])
{
    // ./5_B 1.pipe
    ARGS_CHECK(argc,2);
    int fdr = open(argv[1],O_RDONLY);
    train_t train;
    char filename[4096] = {0};
    read(fdr,&train.length,sizeof(int));
    read(fdr,train.data,train.length);
    memcpy(filename,train.data,train.length);

    mkdir("dir1",0777);
    char filepath[8192] = {0};
    sprintf(filepath,"%s/%s","dir1",filename);
    int fd = open(filepath,O_WRONLY|O_TRUNC|O_CREAT,0666);
    while(1){
        memset(train.data,0,sizeof(train.data));
        read(fdr,&train.length,sizeof(int));
        read(fdr,train.data,train.length);
        if(train.length == 0){
            // 写端已经发完了
            break;
        }
        write(fd,train.data,train.length);
    }
    sleep(300);
    close(fd);
    close(fdr);
    return 0;
}
```

写阻塞

2023年8月8日 14:36



读事件：有数据就绪
无数据阻塞.

写事件：写缓冲区有多少一个字节，阻塞.
为空 就绪.

open 管道 O_RDWR

2023年8月8日 14:42

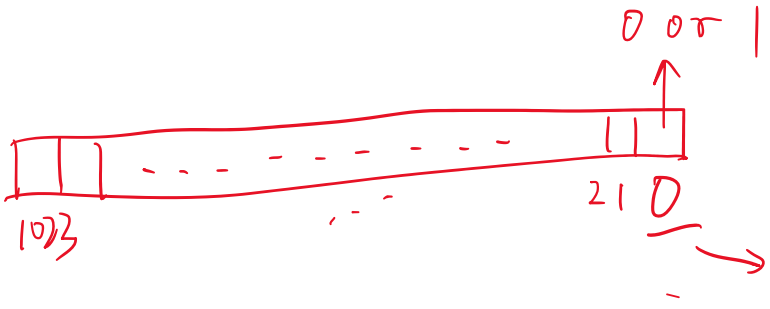
以非阻塞的方式 打开一端 (不常用)

open 两次，可以打开管道的两端.

```c
char buf[4096] = {0};
// 单次写入4096 单次读取2048
fd_set rdset;
fd_set wrset;
int cnt = 0;
while(1){
    FD_ZERO(&rdset);
    FD_SET(fd1,&rdset);
    FD_ZERO(&wrset);
    FD_SET(fd2,&wrset);
    int ret = select(fd2+1,&rdset,&wrset,NULL,NULL);
    printf("ret = %d\n", ret);
    if(FD_ISSET(fd1,&rdset)){
        printf("read ready,cnt = %d\n", cnt++);
        read(fd1,buf,2048);
    }
    if(FD_ISSET(fd2,&wrset)){
        printf("write ready,cnt = %d\n", cnt++);
        write(fd2,buf,4096);
    }
    //sleep(1);
}
return 0;
```

select的底层原理

2023年8月8日 15:05




```
typedef struct
  {
    __fd_mask __fds_bits[1024 / (8 * (int) sizeof (__fd_mask))];
  } fd_set;
```

位图

$$sizeof(\_\_fd\_mask) \times \frac{1024}{8 * sizeof(\_\_fd\_mask)}$$

$\frac{1024}{8}$ B → 1024bit.

open files	(-n) 1024

select运行流程

2023年8月8日 15:12

① 将监听集合拷贝到内核态.

② 内核轮询(polling) 0~nfds-1

③ 轮询直到某个fd主就绪. 得到就绪集合

④ 将就绪集合拷贝回用户态.

用户还要做.
检查每个fd是否在就绪集合内.

select的缺陷

2023年8月8日 15:19

① 上限1024个 不好修改.

② 存在大量内核态和用户态之间的拷贝.

③ 监听和就绪集合 耦合.

④ 就绪检查不合理

epoll.

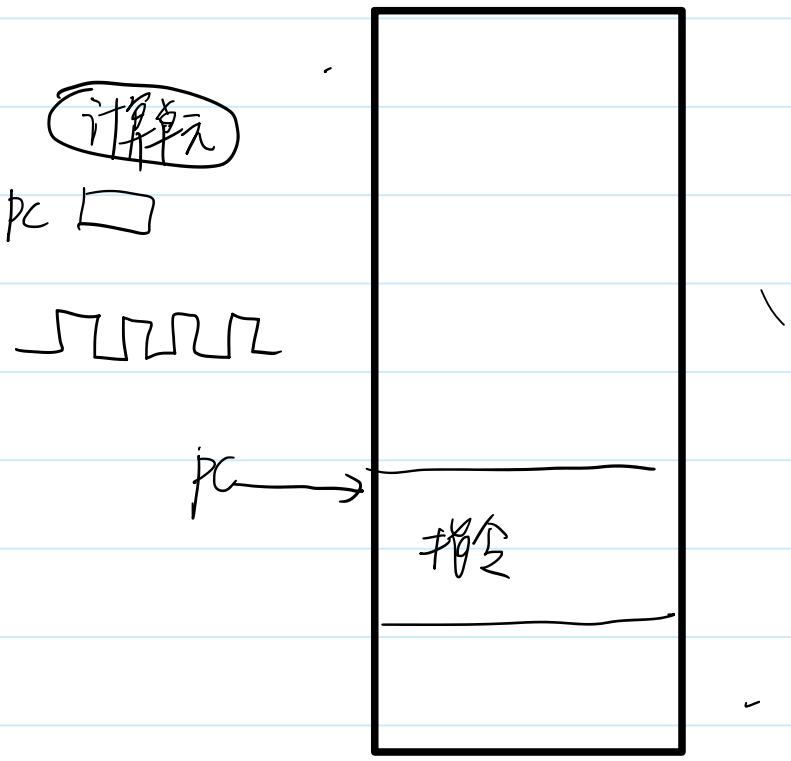
进程

2023年8月8日 15:29

作业.

.bat

正在运作的程序（不含闲）



批处理系统（batch）

↓

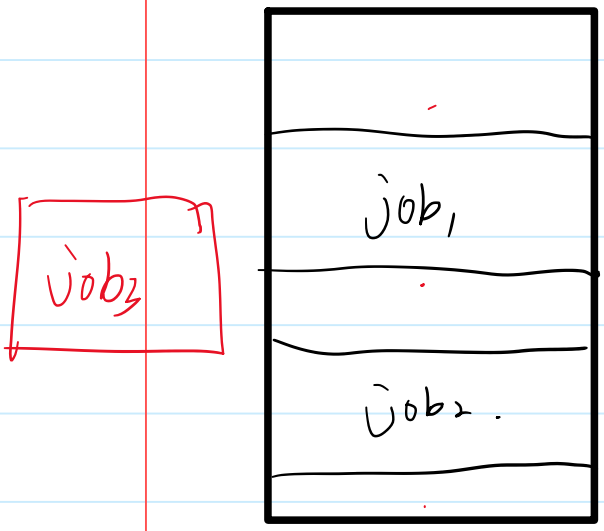
多道程序设计.

↓

分时系统 time-sharing.

多道程序设计的问题

2023年8月8日 16:00



① 地址无隔离

② 相对地址 → 效率差

③ 利用率低

在计算机世界中，所有问题都可以加一个中间层来解决

分层

用户代码 ⟷ 虚拟内存 ⟷ 物理内存

Virtual - 几乎是真实的.

虚拟内存

2023年8月8日 16:12

每个进程以为自己有独立的内存.

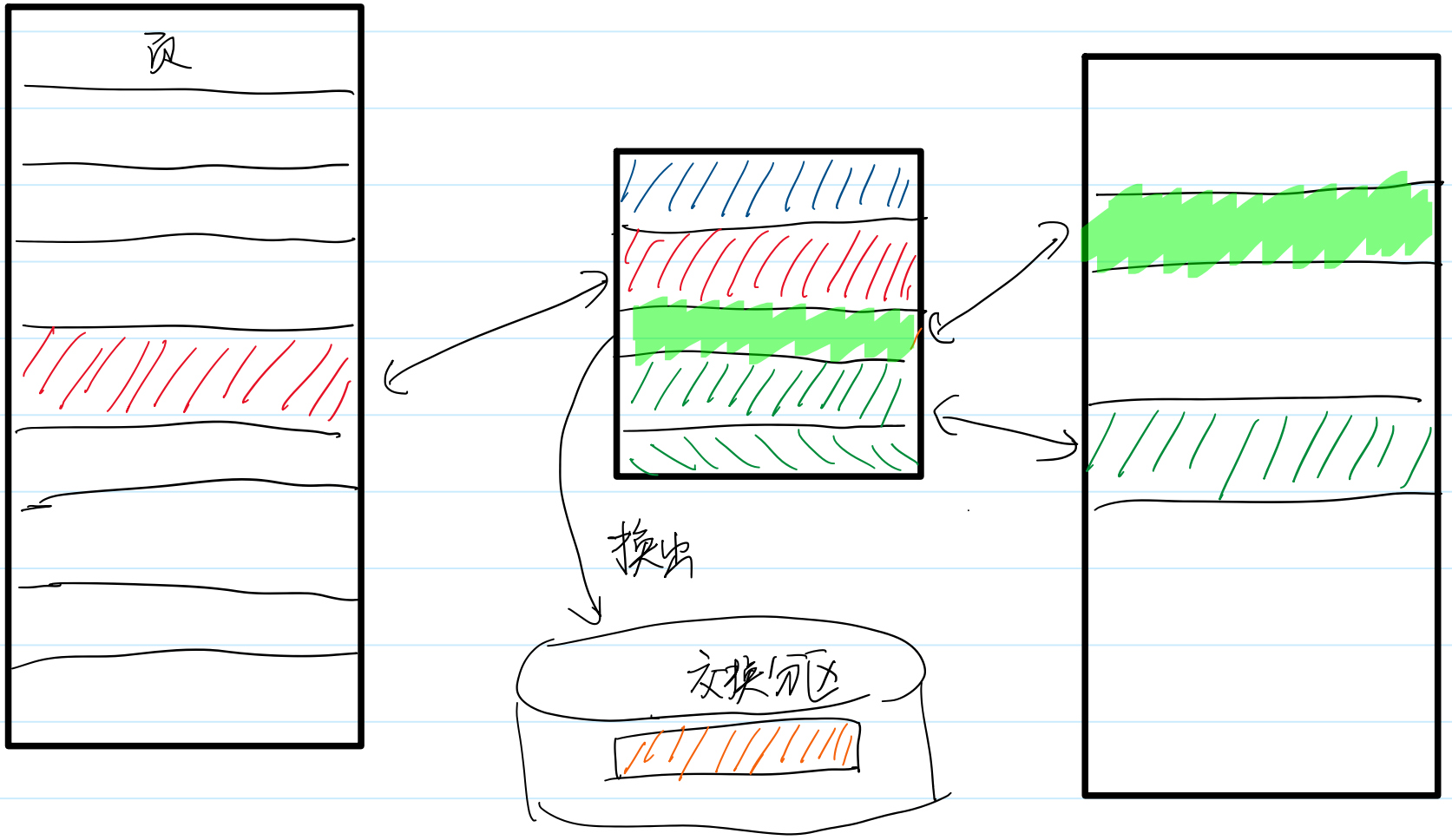
所有字节,每一个地址都是可用的.

→ 隔离性

物理内存支持虚拟内存

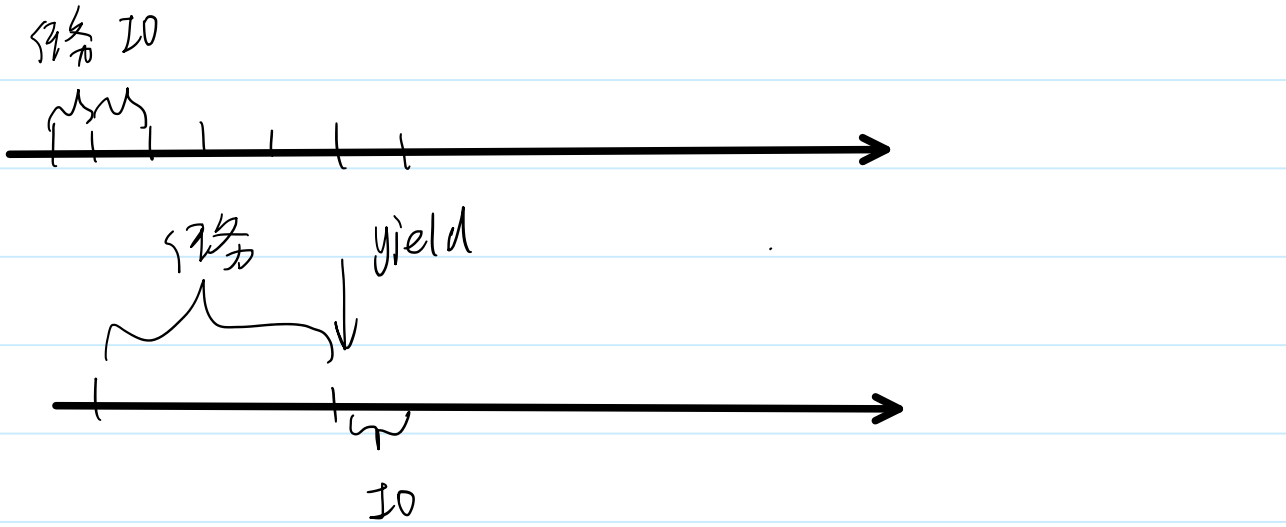
2023年8月8日 16:18

局部性原理 分页机制 内存分成固定大小



一种分时方案

2023年8月8日 16:33



用户代码 → 虚拟CPU → CPU

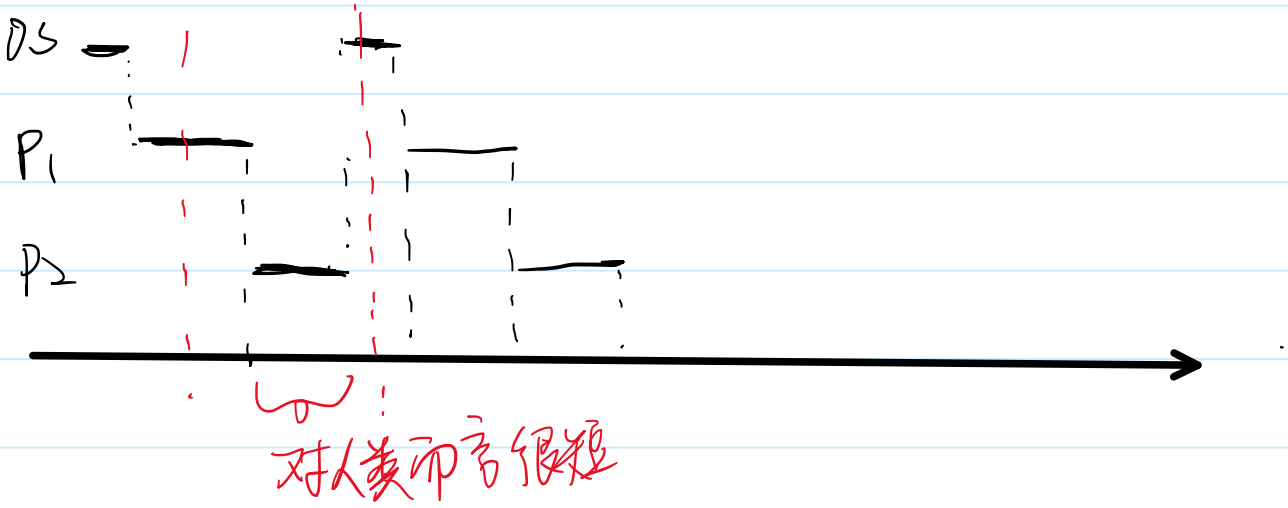
虚拟CPU

2023年8月8日 16:38

让每个进程以为独占CPU.

并发 在一段时间，多个执行流同时运行 单核可以并发，不能并行

并行 在某个时刻、 - - - - - -

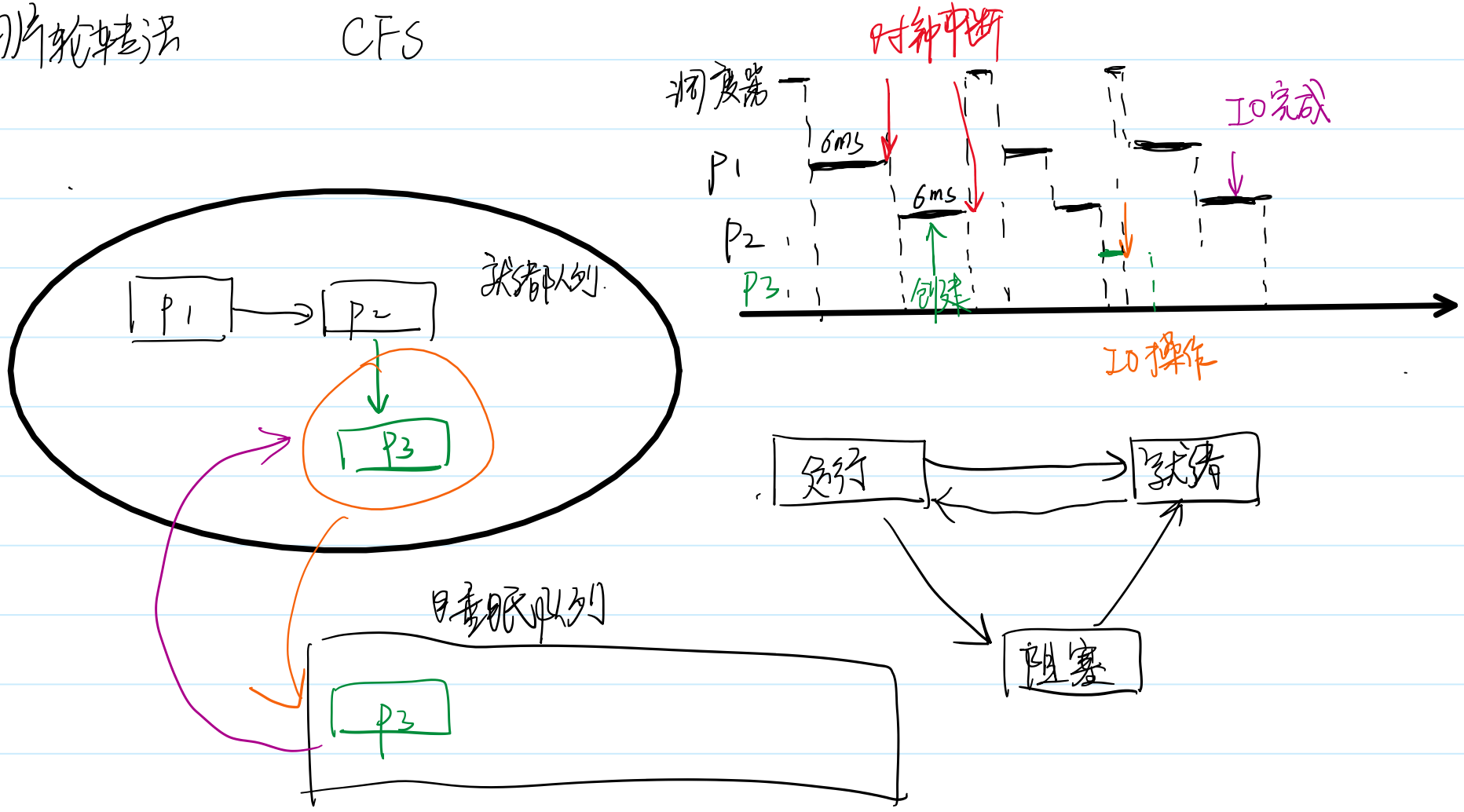


进程调度

时间片　　调度周期 12ms

2023年8月8日　17:05

时间片轮转法　　CFS



背景知识

2023年8月8日 17:20

进程切换：保存寄存器数据
↳上下文

时间片长度：太长"机械"
太短 性能

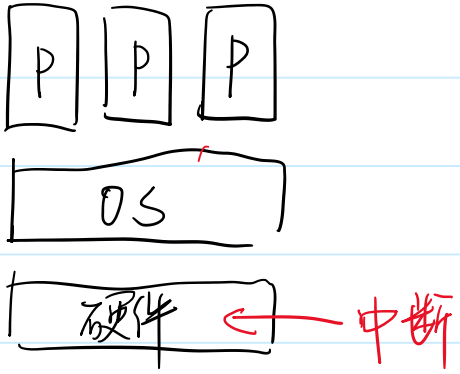
用户态 内核态：① CPU硬件状态

② 地址空间的分区。

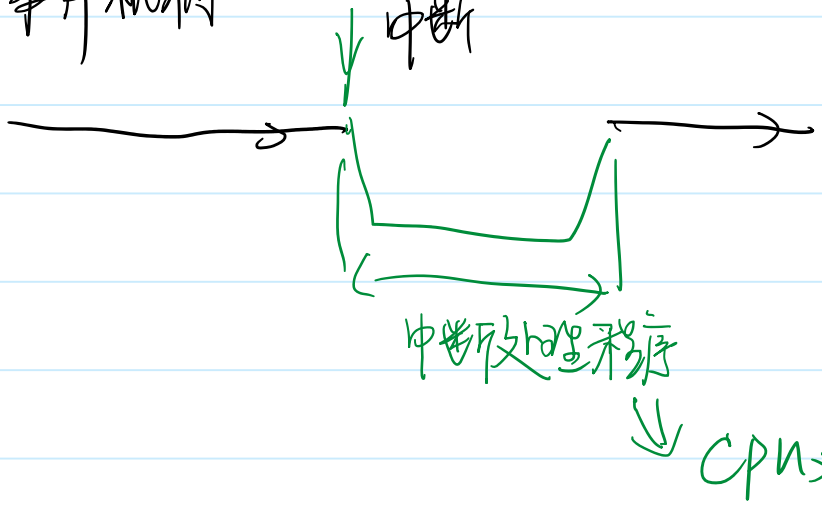
内核态 执行所有指令（内存管理，外部硬件，访问内核态地址空间）

用户态：部分

中断：



事件机制



系统调用 主动向内核发中断。

进程控制块 PCB

2023年8月8日 17:35

OS管理进程的信息的数据结构.

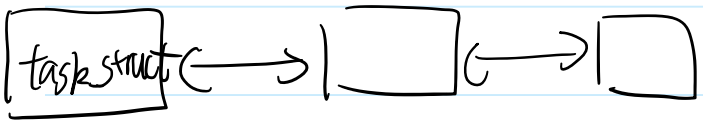
进程控制块 task_struct

Linux.

- 进程状态. CPU..
- 内存.
- 打开文件

```
637  struct task_struct {
638  #ifdef CONFIG_THREAD_INFO_IN_TASK
639          /*
640           * For reasons of header soup (see current_thread_info()), this
641           * must be the first element of task_struct.
642           */
643          struct thread_info              thread_info;
644  #endif
645          /* -1 unrunnable, 0 runnable, >0 stopped: */
646          volatile long                   state;
647
648          /*
649           * This begins the randomizable portion of task_struct. Only
650           * scheduling-critical items should be added above here.
651           */
652          randomized_struct_fields_start
653
654          void                            *stack;
655          refcount_t                      usage;
656          /* Per task flags (PF_*), defined further below: */
657          unsigned int                    flags;
658          unsigned int                    ptrace;
```

任务队列



pid 进程的标识

2023年8月8日 17:49

bash是shell的一个版本 zsh

↓

命令解释器

一个整数，给我们使用的，用来区分不同的进程。

```
liao:LinuxDay11$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   19098   19097  0  80   0 -  5167 do_wai pts/2    00:00:00 bash
0 R  1000   28518   19098  0  80   0 -  5012 -      pts/2    00:00:00 ps
```