操作系统

Operating system

胡燕

大连理工大学

内容纲要　3.3 进程调度与上下文切换

一、调度基本概念

二、调度与进程队列

三、上下文切换

四、调度算法层次

一、调度基本概念

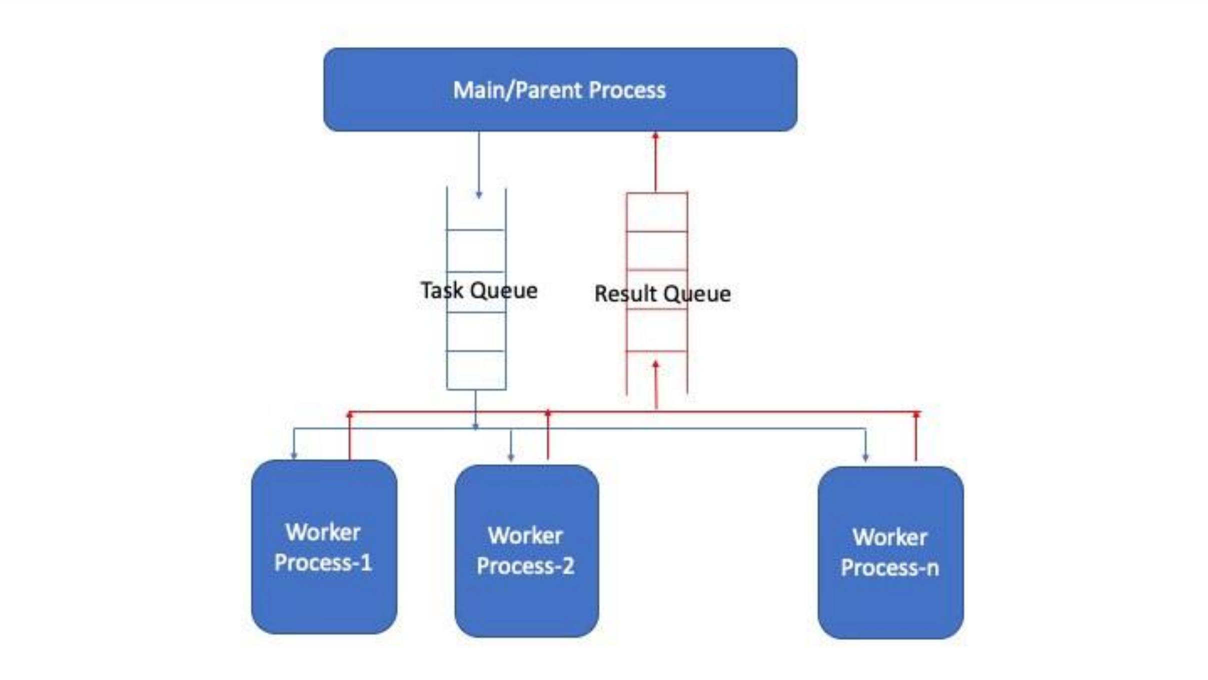
- **为什么要进行调度**
  - 多任务竞争使用CPU资源，需要对它们进行协调
  - 进程不断在变换状态，操作系统需要对不同状态的任务进行管理
- **调度的基本要求**
  - 主要目标：提高CPU利用率 (Maximize CPU Usage)
  - 在不同进程间快速切换，使得多个进程可以分时共享CPU资源
  - 调度器要在合适的时间点，从就绪的进程中选择下一个在CPU上执行的进程

一、调度基本概念

- **为什么要进行调度(必要性探讨)**
  - 如果不进行实施调度：一个进程一旦占用CPU，则一直占用CPU，直到进程终止
  - 有何弊端?

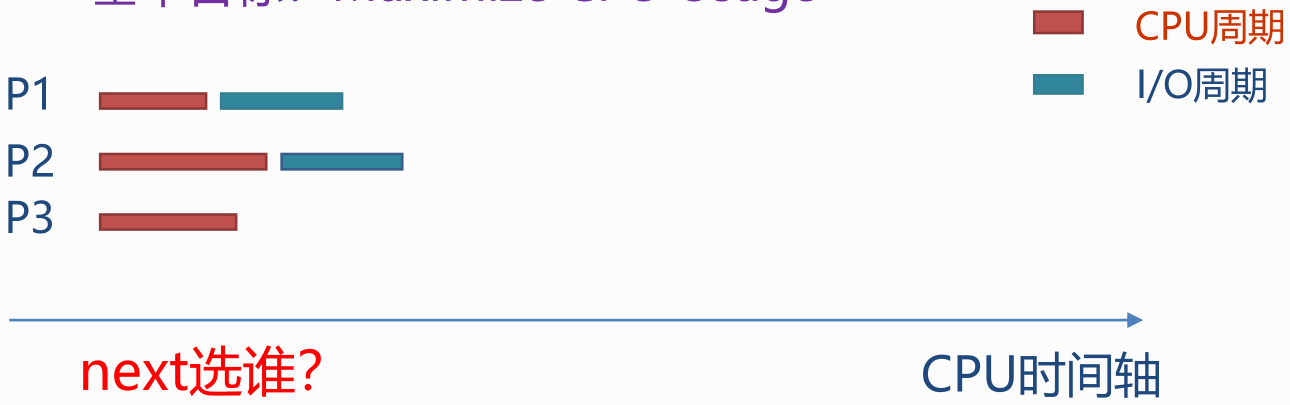
一、调度基本概念

- **为什么要进行调度(必要性探讨)**
  - 如果一个应用从一开始就是以多进程方式设计，则串行根本不能成为一个选项



一、调度基本概念

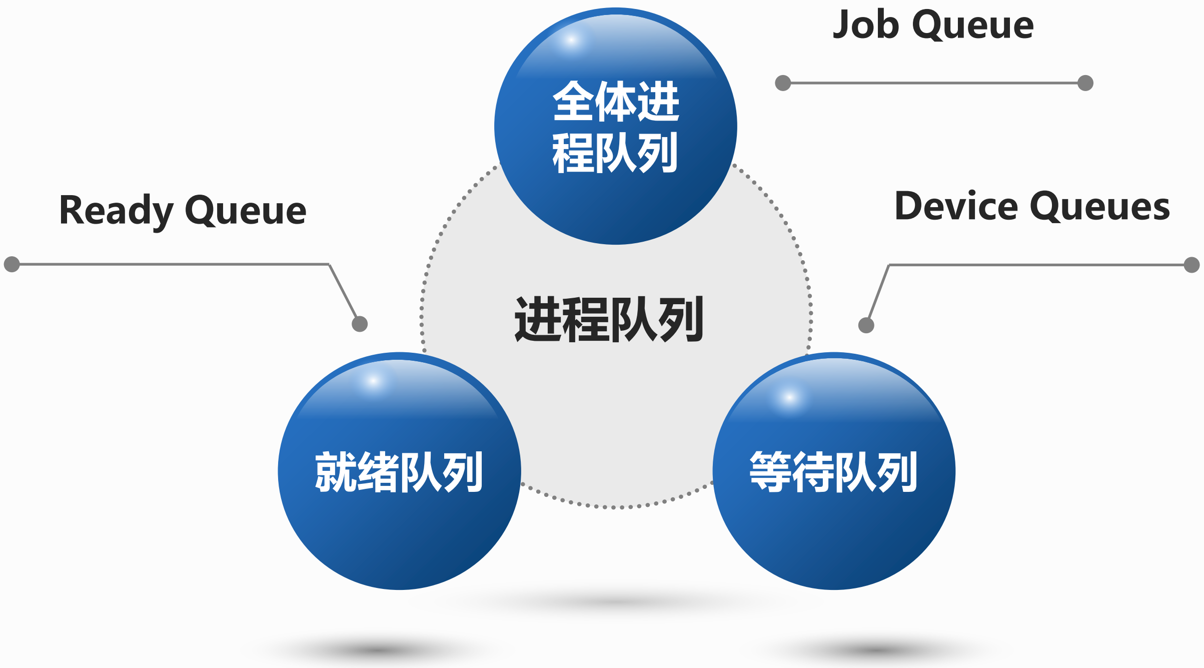
- **调度基本目标与要求(调度是进行高效的并发执行关键)**
  - 基本目标：Maximize CPU Usage



P2,P3排队，选谁看调度标准

二、调度与进程队列

- 调度器需要操作和维护的进程队列包括

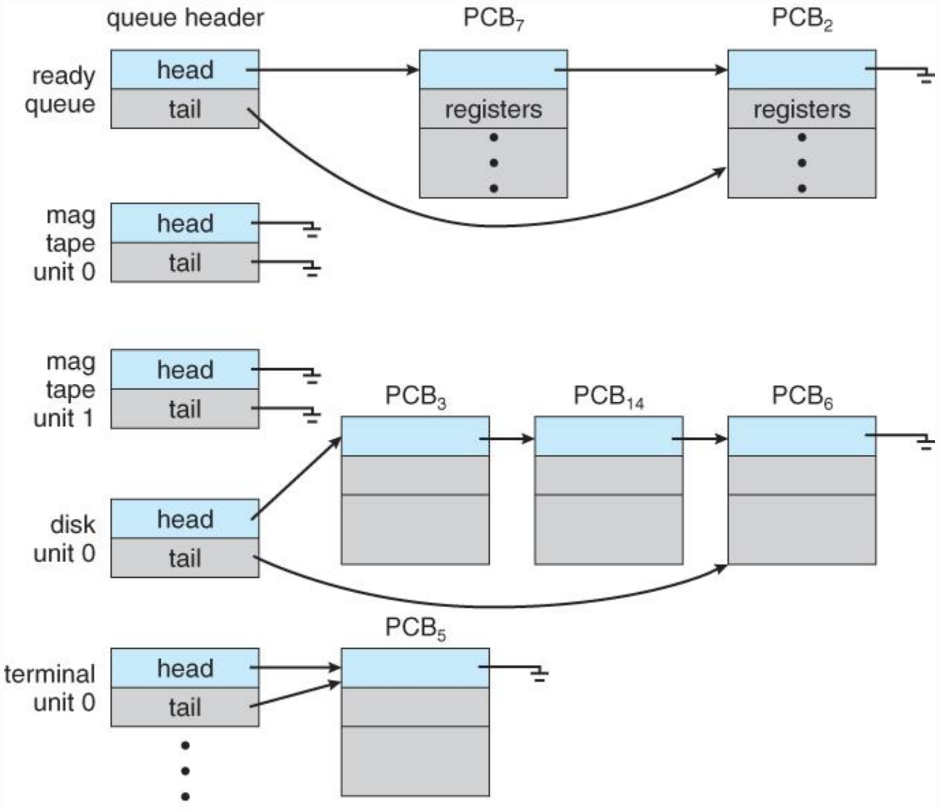


二、调度与进程队列

- **进程队列（思考题）**
  - 全体进程队列，在什么场景下会被用到
  - 就绪队列，应该有几个?
  - 等待队列，应该有几个?

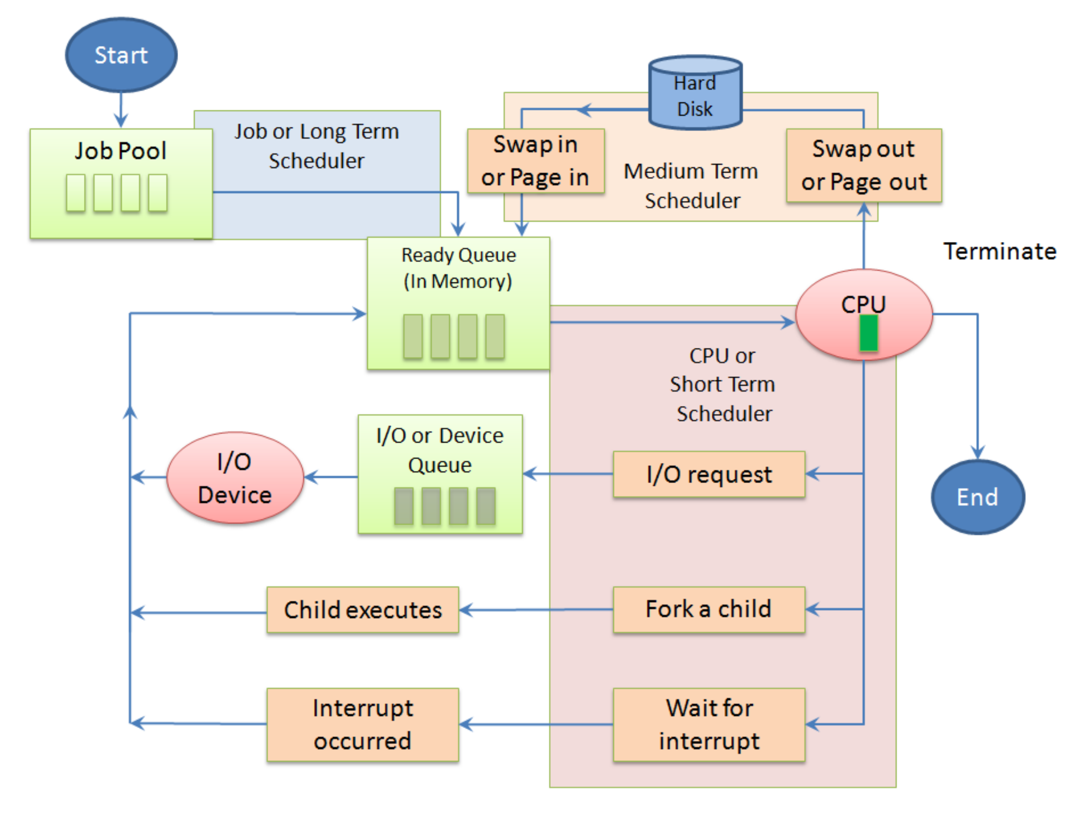
二、调度与进程队列

- **就绪队列与等待队列示意图**



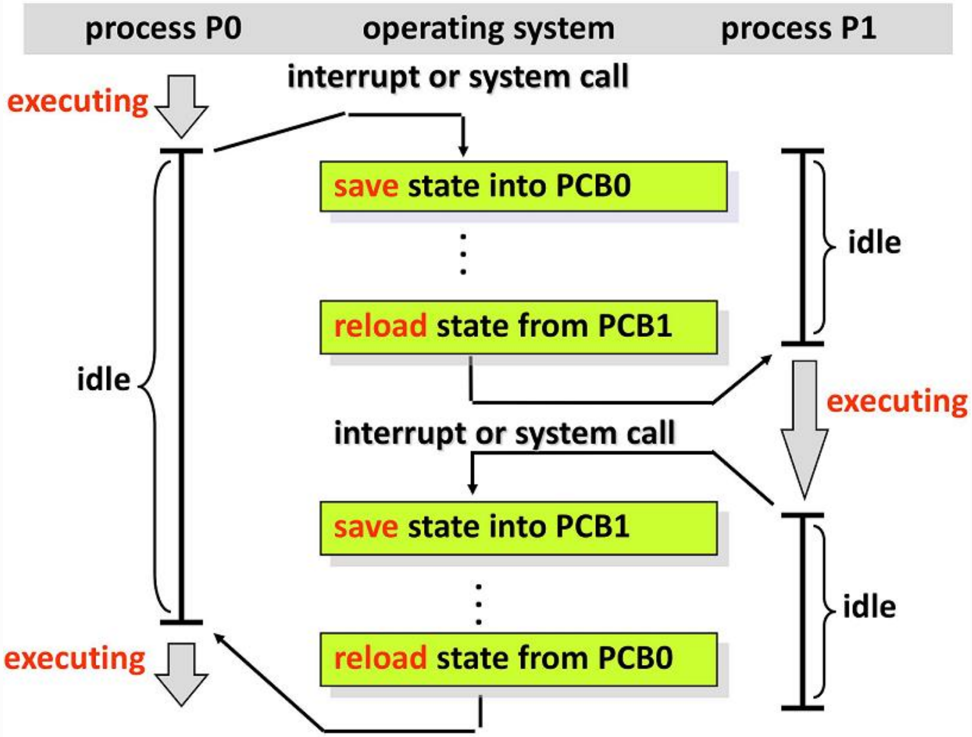
二、调度与进程队列

- 在调度过程中，进程在不同队列之间的迁移图



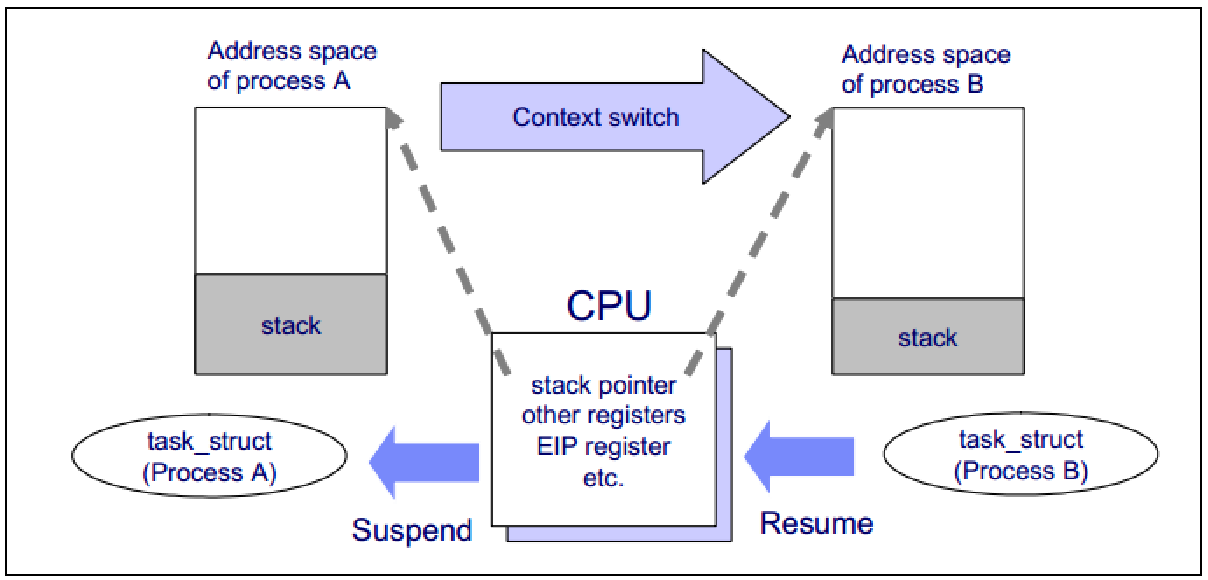
三、上下文切换

- 当进程获取对CPU控制，以及进程退出对CPU控制时，要保存进程执行的现场（又称**上下文**）



三、上下文切换

- **上下文切换细节**



三、上下文切换

- **进程上下文的细分**
  - ① **用户级上下文：Code, Data, User Stack&Heap，共享内存区**
  - ② **寄存器上下文：**通用寄存器、程序寄存器(IP)、处理器状态寄存器(EFLAGS)、栈指针(ESP)；
  - ③ **系统上下文：**进程控制块task_struct、内存管理信息(mm_struct、vm_area_struct、pgd、pte)、内核栈

主观题 8分

设置

请用进程、进程上下文切换的概念描述一下自己在本科学习期间的多任务处理。

正常使用主观题需2.0以上版本雨课堂

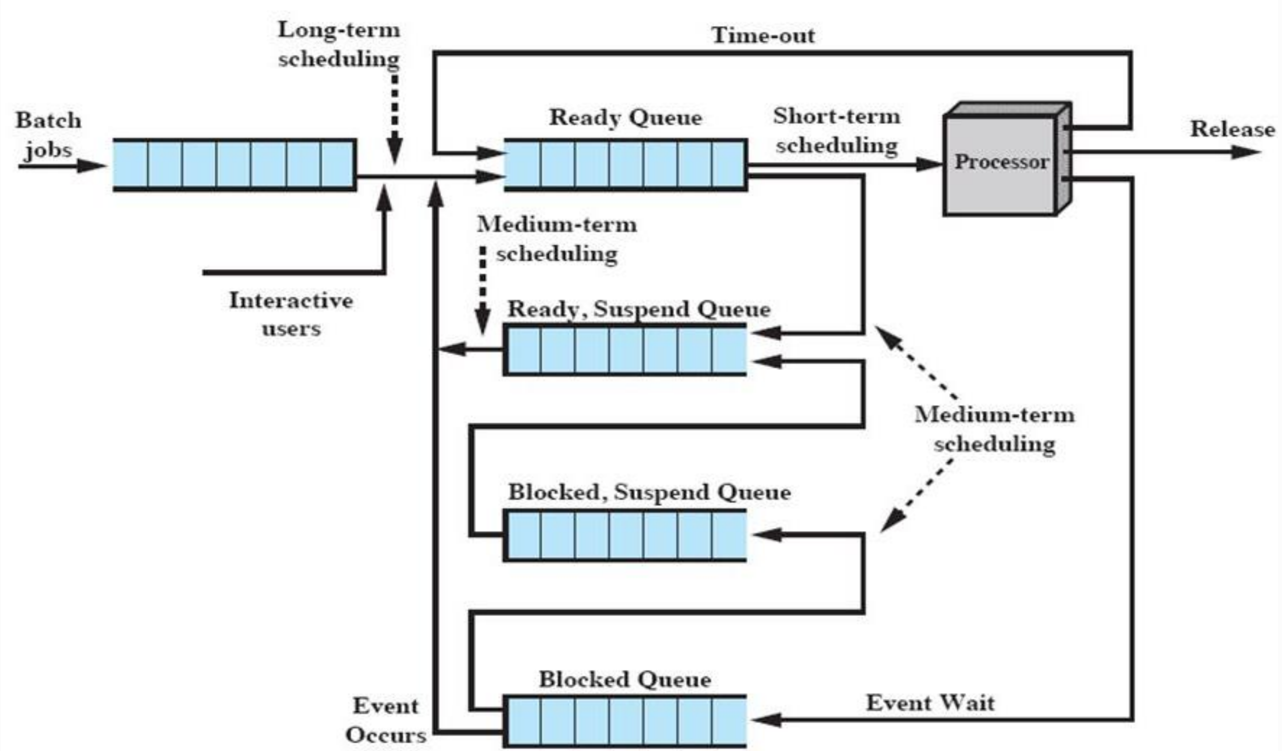
作答

四、调度算法层次

- **调度分为几个层次**
  - 长程调度（Long-Term Scheduling）：控制如何将外存中的作业加载到内存
  - 中程调度（Medium-Term Scheduling）：控制如何对进程实施换入/换出操作，以便控制内存使用率
  - 短程调度（Short-Term Scheduling）：从就绪队列中挑选下个在CPU上执行的进程

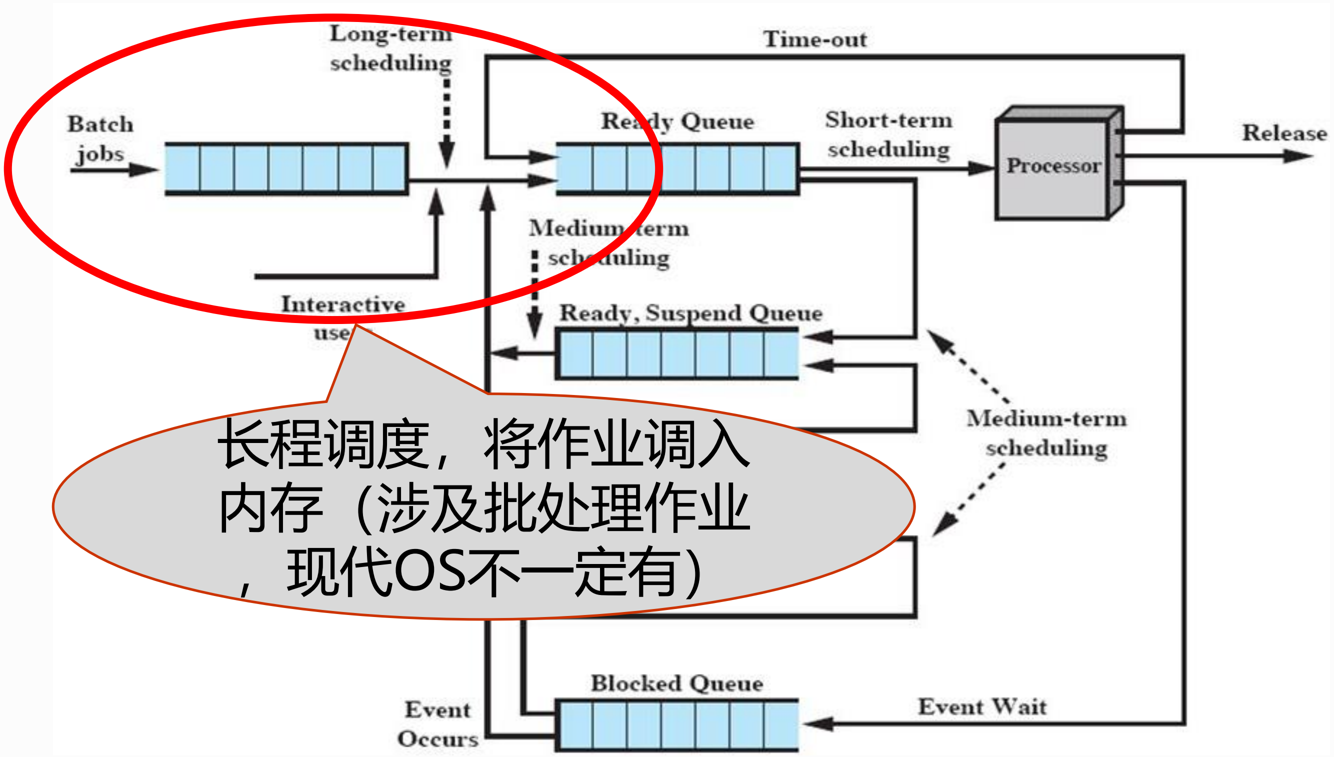
四、调度算法层次

- **调度层次与进程队列关系示意图**



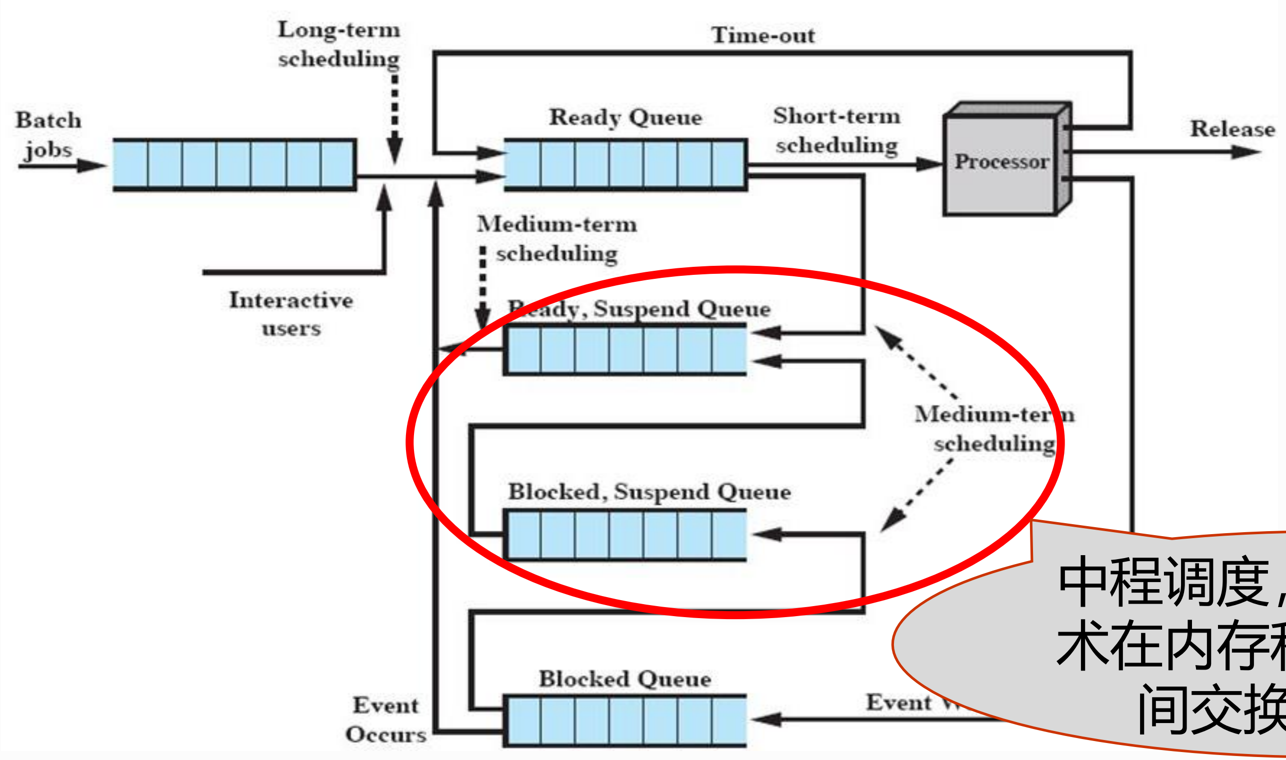
四、调度算法层次

- 调度层次与进程队列关系示意图



四、调度算法层次

- **调度层次与进程队列关系示意图**



中程调度，通过交换技术在内存和交换空间之间交换进程映像

本讲小结

- 调度基本概念
- 调度与进程队列
- 上下文切换
- 调度算法层次

内容纲要　3.4 进程间通信（IPC)

一、IPC基本概念

二、IPC分类

三、共享内存区

四、消息传递

一、IPC基本概念

- **为什么需要进程间通信：**
  - 进程之间的关系可能是独立(independent)，也可能是相互协作（cooperating）。
  - 进程间的协作需要互相传递信息，因此需要专门的通信机制支持

一、IPC基本概念

- **进程间通信的几种目的**

  ① 传送数据

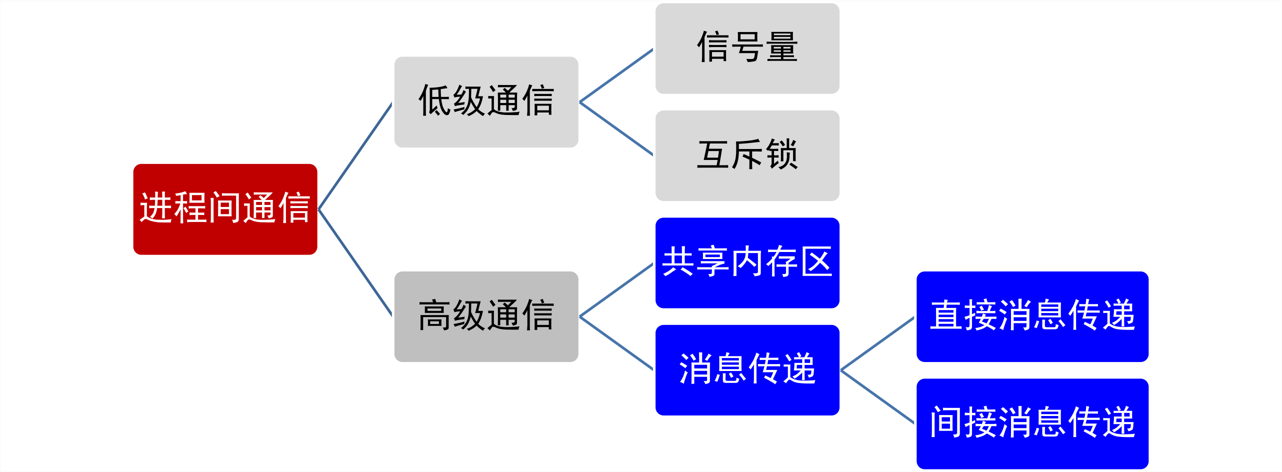
  ② 共享数据

  ③ 通知事件

  ④ 资源共享

  ⑤ 进程控制

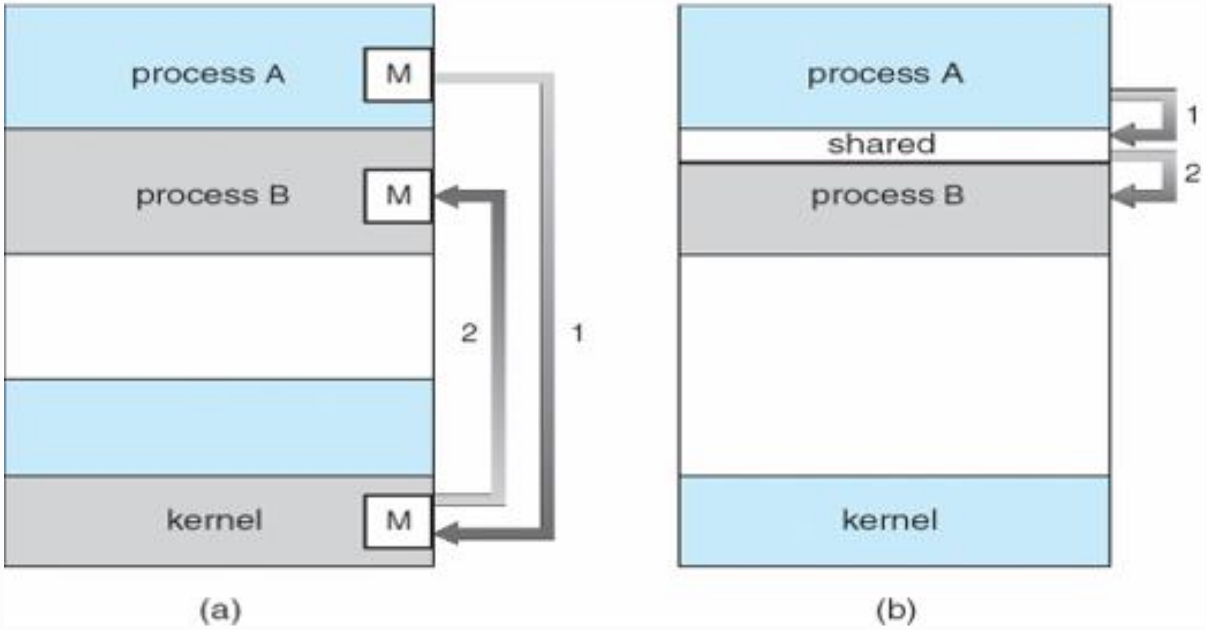
二、IPC分类



- 低级通信（Low-Level IPC）：用于进程控制信息的传递，传输信息量相对较小
- 高级通信：主要用于进程间信息的交换与共享，传输信息量相对较大

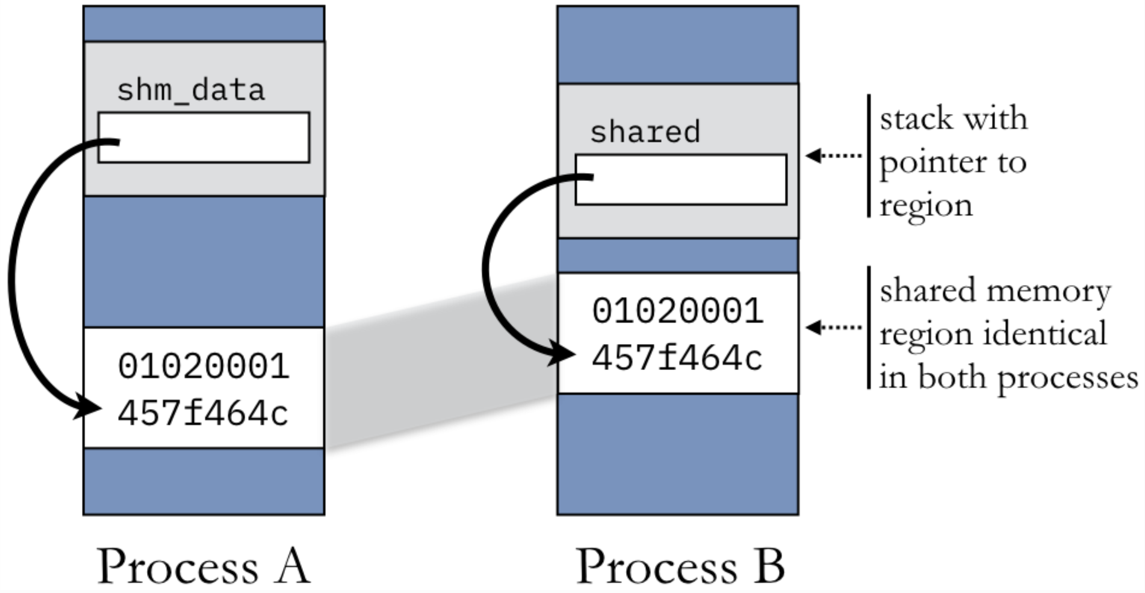
二、IPC分类

- **消息传递与共享内存原理示意**
  - (a) 消息传递　(b) 共享内存



三、共享内存区

Shared Memory



四、消息传递

Message Passing

提供的基本操作：

send(msg), receive(msg)

优势：编程模型简单

需要进行消息传递的通信链路

四、消息传递

- 消息通信又可细分为两类

1 直接消息通信

要求消息发送方和接收方同时在线

2 间接消息通信

(mailbox)

不要求消息发送方和接收方同时在线

本讲小结

- IPC基本概念
- IPC分类
- 共享内存区
- 消息传递

E.1 Linux下编写多任务程序入门

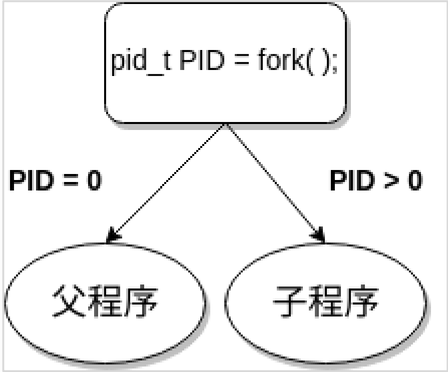
- **Linux下如何编写多任务程序?**
  - 基于fork系统调用

- **fork示例程序1**

#include <unistd.h>

...

pid_t fork(void);




```c
/* example1.c */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(){
    // 從呼叫 fork 開始, 會分成兩支程序多工進行
    pid_t PID = fork();

    switch(PID){
        // PID == -1 代表 fork 出錯
        case -1:
            perror("fork()");
            exit(-1);

        // PID == 0 代表是子程序
        case 0:
            printf("I'm Child process\n");
            printf("Child's PID is %d\n", getpid());
            break;

        // PID > 0 代表是父程序
        default:
            printf("I'm Parent process\n");
            printf("Parent's PID is %d\n", getpid());
    }

    return 0;
}
```

E.1 Linux下编写多任务程序入门

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{

    // make two process which run same
    // program after this instruction
    fork();

    printf("Hello world!\n");
    return 0;
}
```

程序的输出是什么?

主观题 2分

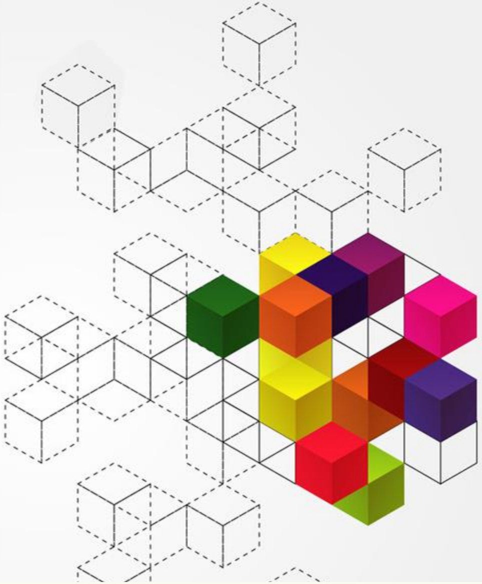
设置

程序的输出是什么?

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

void forkexample()
{
    int x = 1;

    if (fork() == 0)
        printf("Child has x = %d\n", ++x);
    else
        printf("Parent has x = %d\n", --x);
}
int main()
{
    forkexample();
    return 0;
}
```



正常使用主观题需2.0以上版本雨课堂

作答

E.1 Linux下编写多任务程序入门

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
  pid_t pid;
  pid = fork();
  if (pid < 0) {
    perror("fork failed");
    exit(1);
  }
```

E.1 Linux下编写多任务程序入门

```
if (pid == 0) {
    int i;
    for (i = 3; i > 0; i--) {
      printf("This is the child\n");
      sleep(1);
    }
    exit(3);
  } else {
    waitpid(NULL);
    printf("Child has exited. This is Parent Process. \n");
  }
  return 0;
}
```

程序的输出是什么?

E.2 Linux下IPC实践初步

- **Linux下如何进行进程间通信?**
  - 管道: 基于pipe系统调用

E.2 Linux下IPC实践初步

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

main()
{
    int     fd[2];

    pipe(fd);
          .
          .
}
```

E.2 Linux下IPC实践初步

```
pid_t   childpid;

if(childpid == 0)
{
    /* Child process closes up input side of pipe */
    close(fd[0]);
}
else
{
    /* Parent process closes up output side of pipe */
    close(fd[1]);
}
```

E.2 Linux下IPC实践初步

```
if(childpid == 0)
{
    /* Child process closes up input side of pipe */
    close(fd[0]);

    /* Send "string" through the output side of pipe */
    write(fd[1], string, (strlen(string)+1));
    exit(0);
}
```

E.2 Linux下IPC实践初步

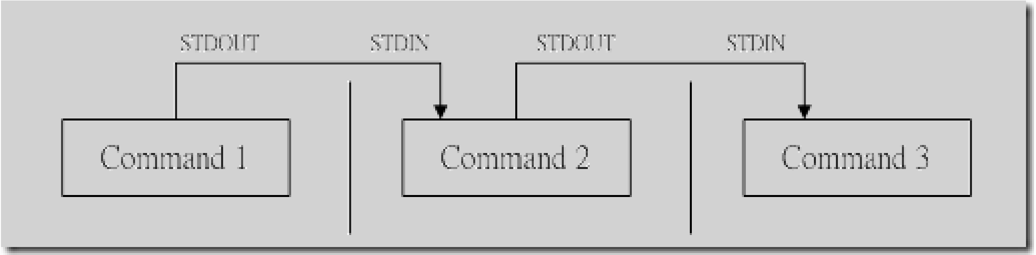
```
int nbytes;
char    readbuffer[80];
```

```
else
{
    /* Parent process closes up output side of pipe */
    close(fd[1]);

    /* Read in a string from the pipe */
    nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
    printf("Received string: %s", readbuffer);
}
```

E.2 Linux下IPC实践初步

- **Shell管道**
  - Linux命令行下的管道应用



E.2 Linux下IPC实践初步

- **Shell管道**
  - 示例：cat命令和head、tail命令配合显示文件特定部分

cat 命令是concatenate 的缩写，常用来显示文件内容，或者将几个文件连接起来显示

```
cat  filename  |head -n 30

cat  filename  |head -n 30 |tail -n  +10
```

E.2 Linux下IPC实践初步

- **Shell管道**
  - 示例：cat命令和head、tail命令配合显示文件特定部分

```
$ cat readme.txt | head –n 2
```

显示文件readme.txt前2行

E.2 Linux下IPC实践初步

"cat" 是Linux中的一个程序

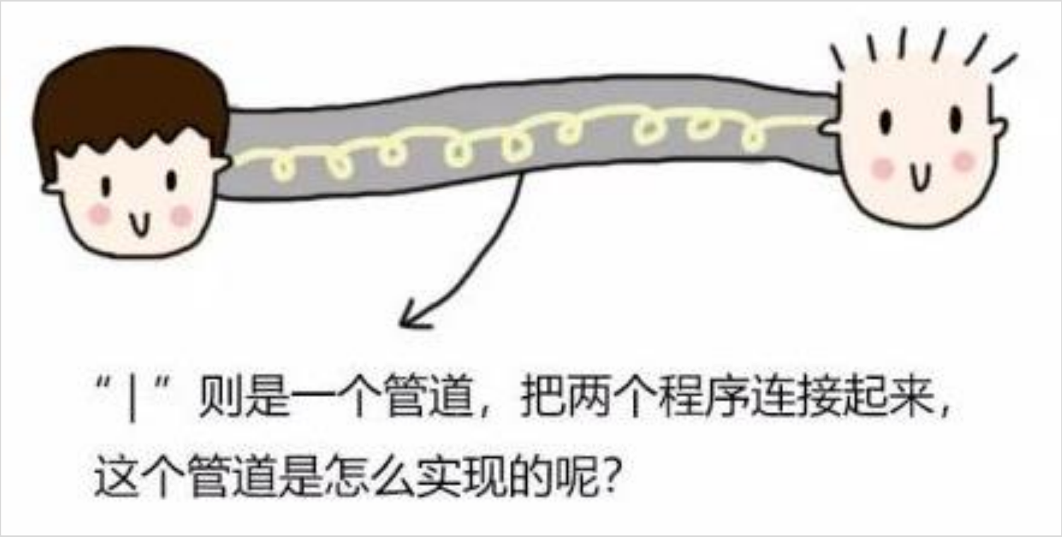
可以显示文件内容

"head" 也是Linux中的一个程序

可以输出一个文件的开头部分

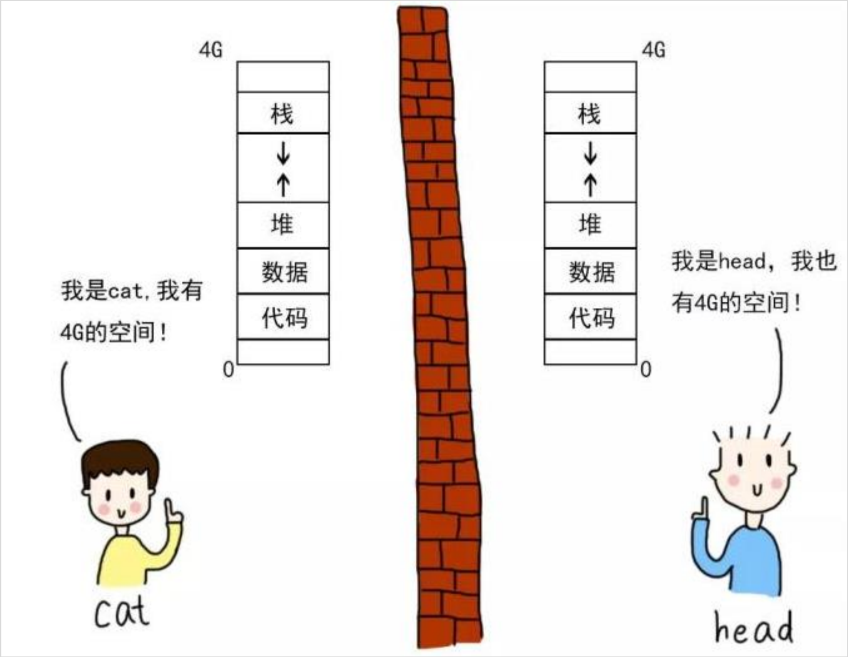
例如输出两行

E.2 Linux下IPC实践初步



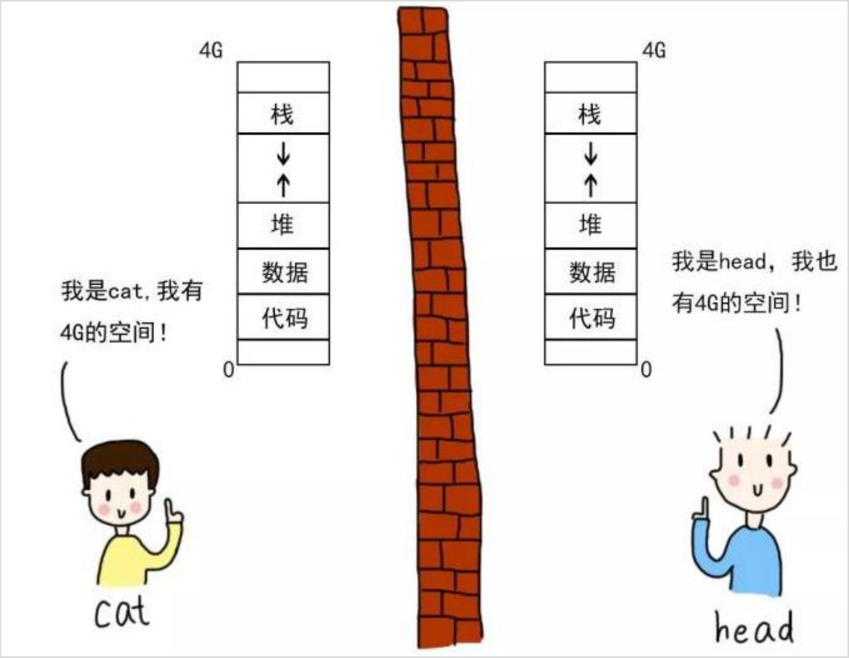
E.2 Linux下IPC实践初步

- **Linux内，每个进程都有自己的独立地址空间，进程之间如同横亘着一堵墙，他们如何通信?**



E.2 Linux下IPC实践初步

- **Linux内，每个进程都有自己的独立地址空间，进程之间如同横亘着一堵墙，他们如何通信？**



E.2 Linux下IPC实践初步

- **Linux进程间通信解决方案1：在内核为需要通信的进程建立一条管道**