

进程 联系

Linux
Android
Linux
OpenStack
Mac OS
Windows



进程联系

本讲内容

1. 顺序程序与顺序环境
2. 并发环境与并发进程
3. 与时间有关的不确定
4. 相交进程与无关进程
5. 进程同步与进程互斥

顺序程序与顺序环境

1 顺序环境

- 程序的指令或语句序列是顺序的
- 在计算机系统中 **只有一个程序** 在运行
- 一个程序独占系统中所有资源
- 一个程序执行不受外界影响

顺序程序与顺序环境

2 顺序特征

顺序性执行

封闭独占资源

确定可再现性

进程联系

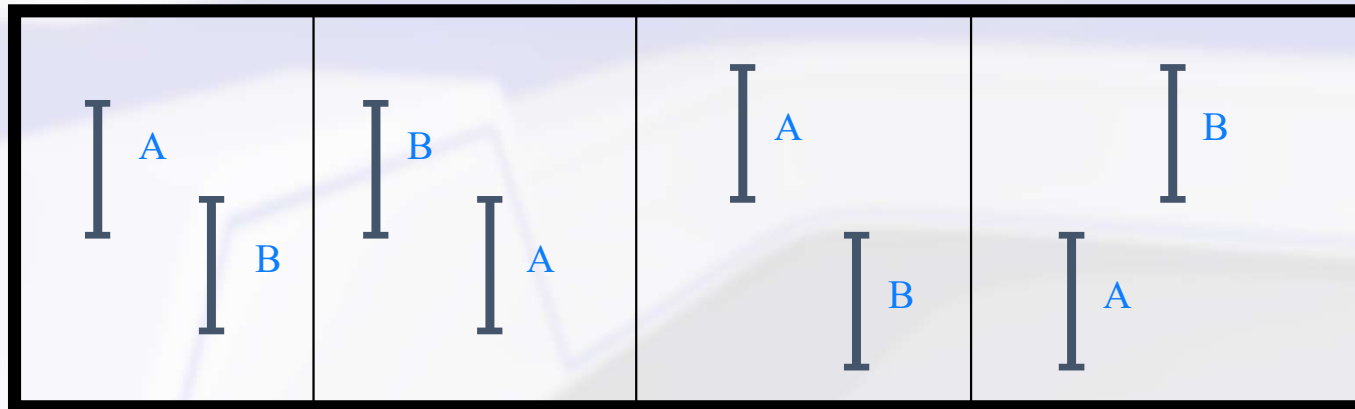
本讲内容

1. 顺序程序与顺序环境
2. 并发环境与并发进程
3. 与时间有关的不确定
4. 相交进程与无关进程
5. 进程同步与进程互斥

并发环境与并发进程

1 并发环境

- 在一定时间内物理机器上有两个或两个以上的程序
- 程序处于开始运行但尚未结束的状态
- 程序执行次序不是事先确定的



并发环境与并发进程

2 并发特征



程序结果的不可再现性



程序的执行呈现间断性



系统中各类资源共享



独立性和制约性



程序和计算不再对应

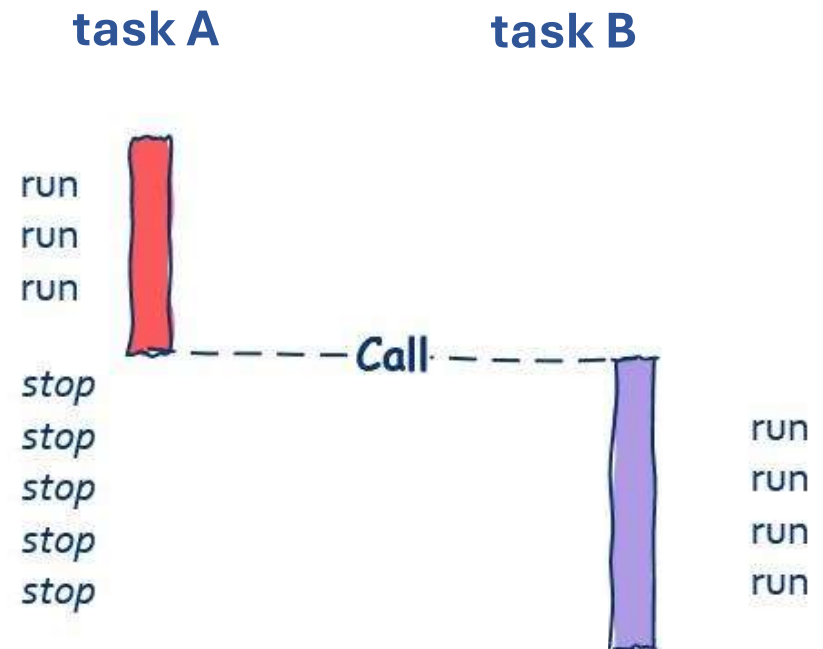
进程联系

本讲内容

1. 顺序程序与顺序环境
2. 并发环境与并发进程
3. 与时间有关的不确定
4. 相交进程与无关进程
5. 进程同步与进程互斥

1 程序同步执行

```
test1.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 void taskA() {
6     printf("Starting task A...\n");
7     sleep(2);
8     printf("Task A completed\n");
9 }
10
11 void taskB() {
12     printf("Starting task B...\n");
13     sleep(3);
14     printf("Task B completed\n");
15 }
16
17 int main() {
18     taskA();
19     taskB();
20     return 0;
21 }
```

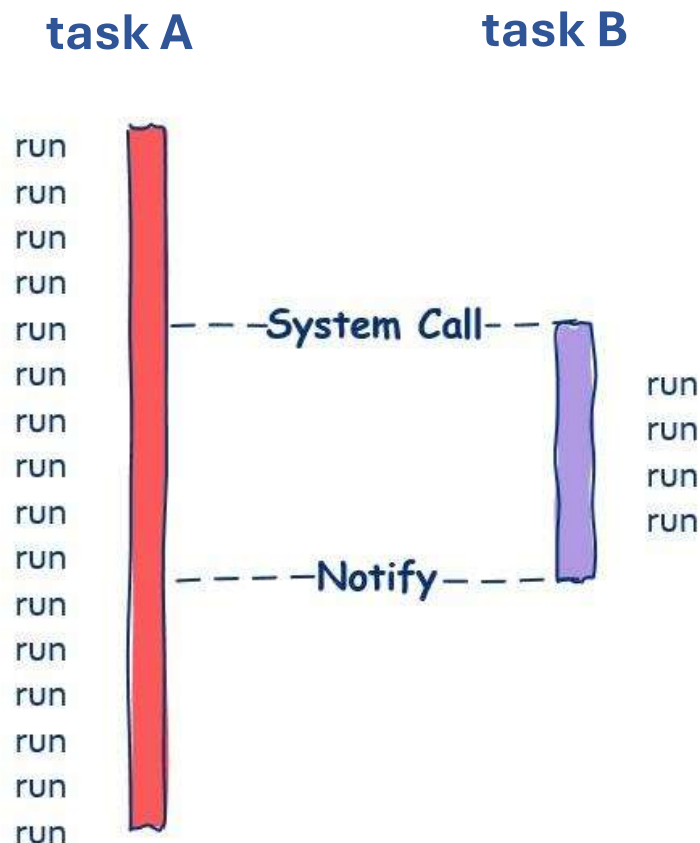


同步执行与异步

- ❏ "程序的运行是同步的" 意味程序中的操作按照顺序依次执行，每操作都要等待上一个操作的完成才能开始。
- ❏ 这种同步执行的可以确保操作的有序性和可控性。在这种情况下，(单个)程序**执行是线性的和顺序的**。

2 程序异步执行

```
test1.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <unistd.h>
5
6 void *taskA(void *arg) {
7     printf("Starting task A...\n");
8     sleep(2);
9     printf("Task A completed\n");
10    return NULL;
11 }
12
13 void *taskB(void *arg) {
14     printf("Starting task B...\n");
15     sleep(3);
16     printf("Task B completed\n");
17    return NULL;
18 }
19
20 int main() {
21     pthread_t threadA, threadB;
22     pthread_create(&threadA, NULL, taskA, NULL);
23     pthread_create(&threadB, NULL, taskB, NULL);
24
25     pthread_join(threadA, NULL);
26     pthread_join(threadB, NULL);
27     return 0;
28 }
```



- 在异步编程中，某些操作可以独立于主流程而执行，并且不会阻塞主程序的执行。
- 如果不使用回调函数、事件驱动、多线程或异步、I/O等机制，每个线程会独立执行，各自随机顺序运行。
- 异步操作可以提高程序的效率，特别是在需要等待 I/O 操作完成时。

2 程序异步执行

test1.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <unistd.h>
5
6 void *taskA(void *arg) {
7     printf("Starting task A...\n");
8     sleep(2);
9     printf("Task A completed\n");
10    return NULL;
11 }
12
13 void *taskB(void *arg) {
14     printf("Starting task B...\n");
15     sleep(3);
16     printf("Task B completed\n");
17    return NULL;
18 }
19
20 int main() {
21     pthread_t threadA, threadB;
22     pthread_create(&threadA, NULL, taskA, NULL);
23     pthread_create(&threadB, NULL, taskB, NULL);
24
25     pthread_join(threadA, NULL);
26     pthread_join(threadB, NULL);
27     return 0;
28 }
```

test2.c

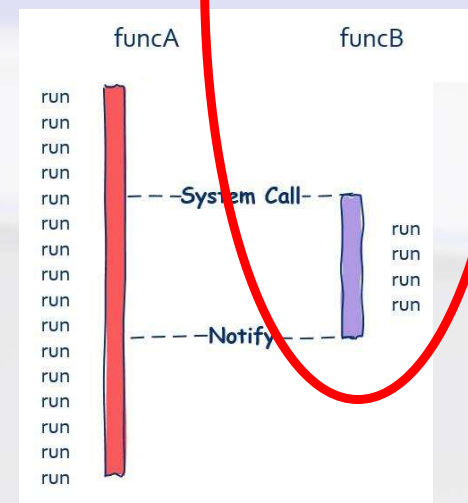
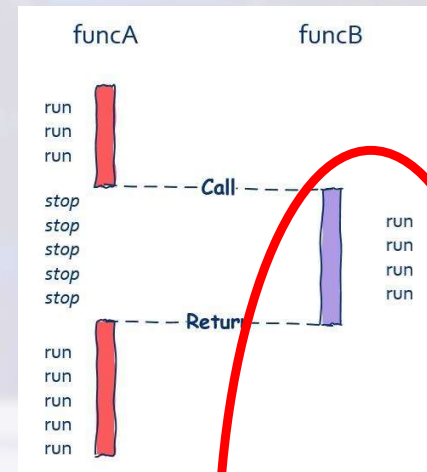
```
douyi@douyi-virtual-machine: ~
Task A completed
Task B completed
douyi@douyi-virtual-machine:~$ ./a.out
Starting task A...
Starting task B...
Task A completed
Task B completed
douyi@douyi-virtual-machine:~$ gcc test.c
douyi@douyi-virtual-machine:~$ ./a.out
Starting task A...
Starting task B...
Task A completed
Task B completed
douyi@douyi-virtual-machine:~$ ./a.out
Starting task A...
Starting task B...
Task A completed
Task B completed
douyi@douyi-virtual-machine:~$ ./a.out
Starting task B...
Starting task A...
Task A completed
Task B completed
douyi@douyi-virtual-machine:~$
```

同步执行与异步

“**进程同步与异步**”则是指在并发系统中，多个进程之间的**消息通信机制**

同步：当一个同步调用发出去后，调用者要一直等待调用结果的返回后，才能进行后续的操作

异步：当一个异步调用发出去后，调用者不用管被调用方法是否完成，都会继续执行后面的代码

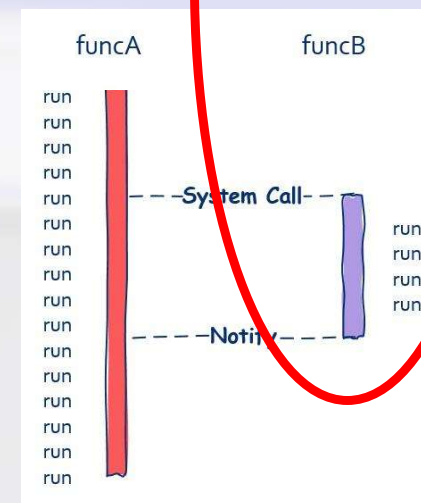
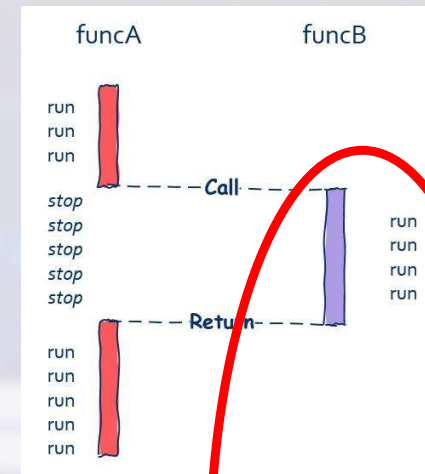


进程同步与异步


“同步或者异步”执行取决于被带调用的进程，是否不主动(同步)/主动(异步)通知调用进程

同步：调用进程就需要每隔一定时间检查一次，效率就很低

异步：使用通知和回调的方式，效率则很高



与时间有关的不确定

 **例2** P1和P2是并发的终端订票进程，x是票数

P1: ① Read(x);

② if $x \geq 1$ then $x := x - 1$

③ write(x)


P2: ④ Read(x)

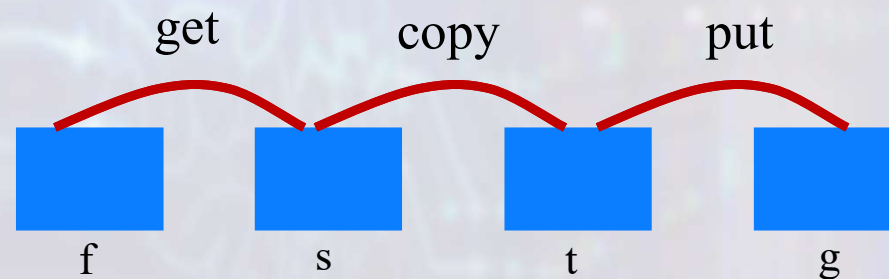
⑤ if $x \geq 1$ then $x := x - 1$

⑥ write(x)

在并发环境中多进程或多线程异步执行，因为竞争引起的数据不一致问题

与时间有关的不确定

 **例3** get、copy、put是3个并发进程



	f	s	t	g
初始状态	3,4, ..., m	2	2	(1,2)
g, c, p g, p, c	4, 5, ..., m	3 3	3 3	$\begin{pmatrix} 1, & 2, & 3 \\ 1, & 2, & 2 \end{pmatrix}$ $\begin{matrix} \checkmark \\ \times \end{matrix}$
c, g, p	4, 5, ..., m	3	2	(1, 2, 2) \times
c, p, g	4, 5, ..., m	3	2	(1, 2, 2) \times
p, c, g	4, 5, ..., m	3	2	(1, 2, 2) \times
p, g, c	4, 5, ..., m	3	2	(1, 2, 2) \times

某个进程可能依赖于另一个进程的状态或结果，而异步执行会导致该依赖关系不再可靠

进程联系

本讲内容

1. 顺序程序与顺序环境
2. 并发环境与并发进程
3. 与时间有关的不确定
4. 相交进程与无关进程
5. 进程同步与进程互斥

相交进程与无关进程

1 间接式与直接式制约



直接作用

两个具有合作关系的进程



间接作用

两个没有合作关系的进程竞争资源

相交进程与无关进程

2 相交与无关进程



相交进程

并发进程在逻辑上有某种联系



无关进程

逻辑上无任何联系的并发进程

- 直接作用只发生在相交进程之间
- 间接作用可以发生在相交进程或无关进程之间

进程联系

本讲内容

1. 顺序程序与顺序环境
2. 并发环境与并发进程
3. 与时间有关的不确定
4. 相交进程与无关进程
5. 进程同步与进程互斥

进程同步与进程互斥

- ❏ **并发的好处**：可以充分利用多核CPU的计算能力提高程序的性能和响应能力
- ❏ **并发的坏处、困难**：增加了程序的复杂性，带来与时间有关的不确定性，例如竞态条件、死锁和数据不一致等安全问题
- ❏ 人类（sequential creatures）是顺序思维，因此为了避免这些问题，要**阻止并发的发生**

进程同步与进程互斥



解决方法:

- ❏ 在局部的地方（需要访问共享信息）取消并发，提高正确性，在剩下的地方依然采用并发的结构
- ❏ 把互相独立的没有互相影响的任务，分派给处理器独立计算，这样就可以进行可靠的并行计算

进程同步与进程互斥

- ❏ 进程同步与互斥，主要解决异步问题，阻止并发
- ❏ “进程同步”则是指在并发系统中，多个进程之间因直接制约而互相等待，彼此相互发送消息进行合作，使得各进程按一定的速度执行
- ❏ 根据一定的时序关系合作完成一项任务
- ❏ 进程间的相互联系是有意识的安排的，直接作用只发生在相交进程间

进程同步与进程互斥



进程的同步（直接作用）

司机 P1

```
while (true)
```

```
{
```

```
    启动车辆;
```

```
    正常运行;
```

```
    到站停车;
```

```
}
```

售票员 P2

```
while (true)
```

```
{
```

```
    关门;
```

```
    售票;
```

```
    开门;
```

```
}
```

- 管道是一种用于进程间通信（IPC）的机制，主要用于在相关进程之间传递数据，可以用于进程同步
- 当一个进程试图从管道中读取数据时，如果管道为空，它会阻塞（等待），直到有数据可读
- 当一个进程向管道写入数据时，如果管道已满，它也会阻塞，直到有足够的空间可用于写入。

写进程

写数据

管道

读数据

读进程

进程同步与进程互斥

进程互斥（间接作用）

- ❏ 多进程之间互相排斥，竞争使用临界资源
- ❏ 临界资源：一次只允许一个进程使用的系统资源
- ❏ 进程间通过中介发生联系，是无意识安排的

进程同步与进程互斥



进程互斥（间接作用）

- ❏ 可发生在相交进程间，也可发生在无关进程间
- ❏ 进程互斥是进程同步的一种特殊形式，专注于协调和控制多个进程对共享资源的访问

进程联系

本讲内容

1. 顺序程序与顺序环境
2. 并发环境与并发进程
3. 与时间有关的不确定
4. 相交进程与无关进程
5. 进程同步与进程互斥