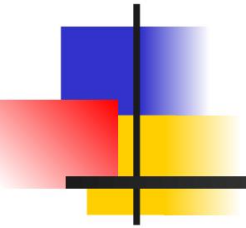
Machine Translated by Google

操作系统原理

操作系统原理



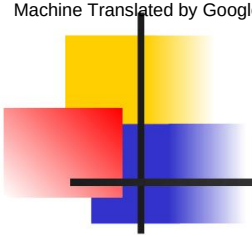
进程间通信

李旭东

leexudong@nankai.edu.cn南

开大学

Machine Translated by Google



目标

竞争条件

互斥

同步通信

Machine Translated by Google

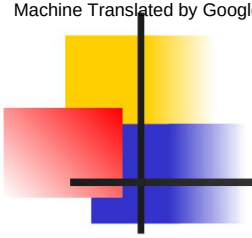
竞争条件

竞争条件

竞動

leexudong@nankai.edu.cn

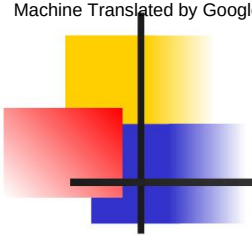
Machine Translated by Google



墨菲定律

如果某件事可能出错,它就会出错

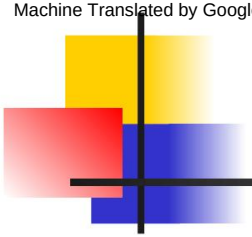
Machine Translated by Google



竞赛条件竞态

两个或多个进程正在读取或写入一些共享数据,最终结果取决于谁精确运行

Machine Translated by Google



流程关系

资源共享

合作

…

Machine Translated by Google

关键区域

- Critical Resource 临界资源

  文件,内存,信号量，...

- 临界区

  有时一个进程需要访问共享

  内存或文件,或者执行其他可能导致竞争的关键操作

  程序中共享的部分

  被访问的内存被称为临界区或临界段

Machine Translated by Google

四个前提条件

避免竞争条件

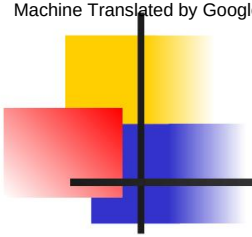
1. 任何两个进程都不能同时在其关键区域内。

2. 不能对速度或 CPU 数量做出任何假设。

3. 任何在临界区外运行的进程都不得阻塞其他进程。

4. 任何进程都不应该永远等待进入其临界区域。

Machine Translated by Google



互斥

互斥互斥访问

Machine Translated by Google

互斥

如何访问关键资源

重复

入口部分

关键部分；

出口部分

余数部分；

直到错误

Machine Translated by Google

互斥解决方案

禁止中断

中断禁用时,不会发生任何时钟中断发生。

CPU 只会从进程切换到

毕竟,由于时钟或其他中断而导致的进程中断,并且关闭中断后 CPU 不会切换到其他进程。

因此,一旦某个进程禁用了中断,它就可以检查和更新共享内存,而不必担心任何其他进程会干预。

Machine Translated by Google

互斥解决方案

锁定变量

竞争条件

a :软件解决方案

考虑有一个单一的、共享的（锁）变量，最初为 0。

当一个进程想要进入临界区时，它首先测试锁。如果锁为 0，则进程将其设置为 1 并进入临界区。如果锁已经为 1，则进程只需等待，直到它变为 0。

因此，

a=0 表示没有进程处于其临界区，　　a=1 表示某个进程处于其临界区。

Machine Translated by Google

互斥解决方案

- Strict alternation 严格轮换法

整数变量 turn 初始为 0,保持

跟踪谁该进入临界区并检查或更新共享内存

Machine Translated by Google

严格交替（续）

问题 1

忙等待

持续测试一个变量直到出现某个值

问题2

运行速度不匹配 速度不匹配

问题3

进程 0 被不在其关键区域中的进程阻塞

Machine Translated by Google

互斥解决方案

彼得森的解决方案

```
#define FALSE 0
#define TRUE 1
#define N 2 /*进程数*/ shared int turn; /*轮到谁了?*/ shared
int interested[N]; /*所有值最初均为 0*/ void enter_region(int
process) {


      int other;
      other=1-process;
      interested[process]=TRUE;
       turn=process;
       while(turn==process && interested[other]==TRUE); ?while(turn!
=process && interested[other]==TRUE); 可以吗?
}
void leave_region（int 进程){

      感兴趣[过程]=FALSE;
}
```

Machine Translated by Google

互斥解决方案

TSL 指令

测试并设置锁定

TSL 注册、锁定

输入区域：

TSL 注册,锁定1

CMP 寄存器,#0

JNE 进入区域

回转窑

离开区域：

移动锁1,#0

回转窑

Machine Translated by Google

互斥解决方案

TSL 指令

测试并设置锁定

TSL 注册、锁定

它将内存字 lock 的内容读入寄存器 RX,然后在内存地址 lock 处存储一个非零值。 读取字和存储字的操作保证不可分割 - 在指令完成之前,任何其他处理器都不能

访问该内存字。

Machine Translated by Google

互斥解决方案

交换指令　XCHG

```
程序交换（var a,b:boolean）
变量温度:布尔值;
开始温
  度:=a;a:=b;
  b:
  =temp;
结尾;
```

Machine Translated by Google

互斥解决方案

总结

忙等待

纺纱

进程不断检查某个条件是否为真

优先级反转问题

高优先级任务被中优先级任务间接抢占,从而有效地“颠倒”了两个任务的相对优先级

优先级较高的任务:时间充足,但无关键资源

低优先级任务:使用关键资源,但没有时间

Machine Translated by Google

spinlock 自旋锁

自旋锁是操作系统中使用的一种同步机制,用于保护共享资源不被多个线程或进程单独访问。与信号量或

互斥锁,

自旋锁使用忙等待方法,其中一个线程不断选择一个锁,直到它可用。

自旋锁是一种公平有序的机制。它可以防止冲突和潜在的损害。

```
spinlock_t 锁;
spin_lock_init(&lock);
spin_lock(&lock);
...
spin_unlock(&lock);
```

Machine Translated by Google

互斥问题的新解决方案

信号量

1965 年,EW Dijkstra (EWD 信 迪科斯彻)

号量,在字典中的定义是,是一种机械信号装置或一种视觉信号手段

通常使用的类比是列车信号的铁路机制， 机械臂会向下摆动以阻止一列火车驶离另一列火车正在使用的轨道区段。

当轨道空闲时,臂就会向上摆动,等候的火车就可以继续前行。

Machine Translated by Google

互斥问题的新解决方案

信号量

变量

具有两个原子操作： 原始语

向下,向上

- P (Proberen)、V (Verhagen);等待()、信号()　向下()

如果是,则减少该值并继续。 如果该值为 0,则进程将进入睡眠状态，暂时不完成关闭。

Up()

up 操作增加信号量的值

解决。如果一个或多个进程在该信号量上处于休眠状态,无法完成先前的关闭操作,则系统将选择其中一个进程并允许其完成其关闭操作

Machine Translated by Google

IPC 原语 原文

进程间通信原语

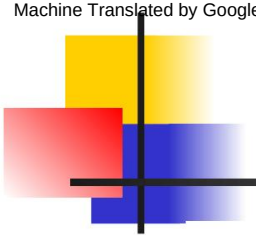
当不允许进入临界区时,阻塞而不是浪费 CPU 时间

示例

睡眠

唤醒

Machine Translated by Google



生产者-消费者问题

生产者-消费者问题（PCP）

- 缓冲区受限问题
- 多进程共享一个固定大小的缓冲

Machine Translated by Google

第一个解决方案：

生产者-消费者问题

```
#define N 100 /*缓冲
区中的插槽数*/ int count=0; /*缓冲区中的
项目数*/ void
produce(void){ int item; while(TRUE)
{ produce_item(&item); if
    (count==N)
    sleep();
        enter_item(item); count=count+1;
        if (count==1) wakeup(consumer);




} }
```

?竞争条件

添加：

唤醒等待位

```
void comsumer(void)
    { int item;
    while(TRUE)
      { if(count==0) sleep();
      remove_item(&item);
      count=count-1;
      if (count==N-1)
          wakeup(producer);
      consumer_item(item); } }
```

Machine Translated by Google

PCP 使用信号量

#define N 100 /*缓冲区中的插槽数*/ typedef int semaphore;

semaphore mutex=1; semaphore empty=N; semaphore full=0; void produce(void) { int item; while(TRUE) { produce_item(&item);

down(&empty);
down(&mutex);
enter_item(item);
up(&mutex);
up(&full); }

}

void consumer(void){ int item;

while(TRUE)

{ down(&full);

down(&mutex);

remove_item(&item);

up(&mutex); up(&empty);

consumer_item(item);

} } ?交换两行down(&mutex);

down(&empty);

leexudong@nankai.edu.cn

Machine Translated by Google

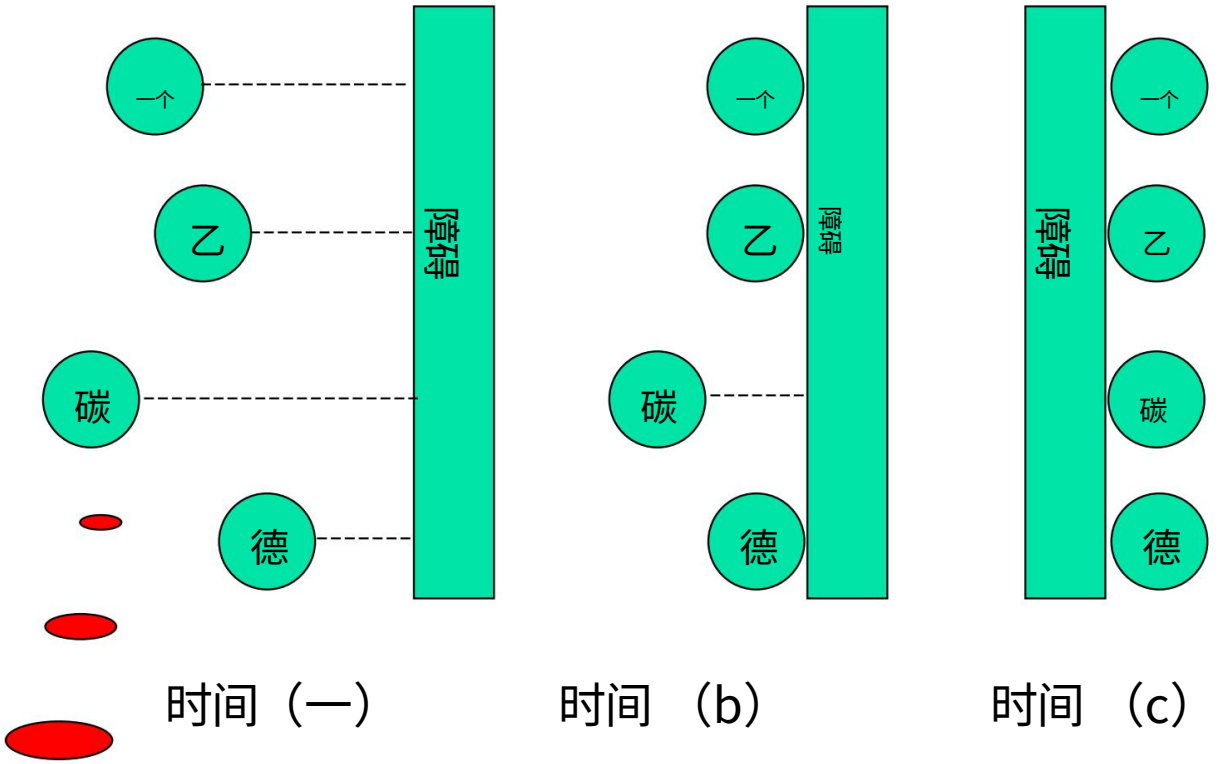
信号量场景

互斥

信号量互斥=1； 同步信号量空=N； 信号量满=0；

Machine Translated by Google

屏障 (Barrier)



如何实现

Machine Translated by Google

互斥锁的实现

互斥

值:0,1

互斥锁（mutex_lock）：

```
TSL 寄存器,互斥
CMP 寄存器,#0
JZE 好的
调用thread_yield          ;安排另一个线程
JMP mutex_lock 确
```

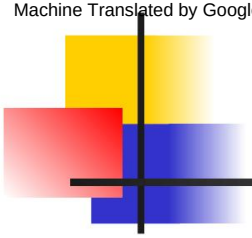
定:RET

无需忙碌等待

互斥解锁：

```
移动互斥锁,#0
回转窑
```

Machine Translated by Google



Pthread 调用:信号量

Machine Translated by Google

信号量集

为什么

进程A：

wait(mutex1);

wait(mutex2);

进程B：

wait(mutex2);

wait(mutex1);

信号量集合

Swait(S1,S2,…,Sn)

S信号(S1,S2,…,Sn)

Machine Translated by Google

条件变量

条件变量

typedef boolean cond;

true,false 使用

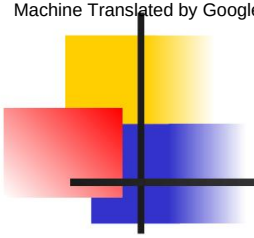
信号量函数

条件初始化 (cond) 条件等待

(cond,互斥)条件信号 (cond) 条件广播

(cond)

Machine Translated by Google



示例:条件变量

Machine Translated by Google

示例:条件变量

是 while() ,而不
是 if()
为什么?

Machine Translated by Google

如何 Imp 条件等待

```
pthread_mutex_t the_mutex;
pthread_cond_t condc， condp;
/*互斥量:1 */ /*互斥
量:1->0*/
pthread_mutex_lock(&the_mutex);

/*the_mutex:0->1*/ /
*cond:等待*/ /
*the_mutex:1->0*/
while(buffer!=0) pthread_cond_wait(&condp,&the_mutex);

/*the_mutex:0 */
XXXXX /*执行到临界区*/    pthread_cond_signal(&condc); /
*the_mutex:0 */


/* 互斥锁:0->1 */
pthread_mutex_unlock(&the_mutex); /* 互斥锁:1
*/
```

Machine Translated by Google

如何 Imp 条件等待

```
pthread_cond_wait（&condp,&the_mutex)int {
pthread_unlock_mutex （&互斥）；
while(循环spin时间){
判断是否有信号来,若有则goto done;
}
while(1){
futex_wait_cancelable（cond,private）；
判断是否有信号来,若有则goto done;
}
完毕:
__condvar_confirm_wakeup（cond,私人）；
线程锁 （&互斥）；
}
```

Machine Translated by Google

条件等待语义

Machine Translated by Google

获取帮助和源代码

·须藤 apt 更新 ·须藤

apt 安装 man-db manpages-posix ·须藤 apt 安

装 manpages-dev manpages-posix-dev

·男人

- man pthread_cond_wait
- sudo apt source glibc

Machine Translated by Google

条件变量:POSIX

```
#include <pthread.h>
int pthread_cond_init(pthread_cond_t *restrict cond,
                      const pthread_condattr_t *限制属性);
pthread_cond_t cond =
PTHREAD_COND_INITIALIZER;
int pthread_cond_destroy（pthread_cond_t *cond）；
int pthread_cond_timedwait（pthread_cond_t *restrict cond，
                      pthread_mutex_t *限制互斥锁,const
                      struct timespec *限制abstime);
int pthread_cond_wait(pthread_cond_t *restrict cond，
                      pthread_mutex_t *restrict mutex)；
int pthread_cond_signal（pthread_cond_t *cond）；
int pthread_cond_broadcast（pthread_cond_t *cond）；
```

Machine Translated by Google

事件计数器

Reed 和 Kanodia,1979

三种操作

阅读(E) 进步

(E)

E=E+1 （原子作用）

等待（E,v）

等待直到 E>=v

Machine Translated by Google

PCP:事件计数器

```
#define N 100
typedef int event_counter;
event_counter in =0;
event_counter out =0; void
productioner(void){
    int item,sequence=0;
    while(TRUE)
      { produce_item(&item);
      serial=sequence+1;
      await(out,sequence-N);
      enter_item(item);
      advance(&in); }

}
```

```
void consumer(void)
        { int item,sequence=0;
        while(TRUE)
          { serial=sequence+1;
          await(in,sequence);
          remove_item(&item);
          advance(&out);
          consumer_item(item); }

}
```

Machine Translated by Google

监视器

使用信号量的一些错误

错误1：

信号（互斥）

临界区

等待（互斥）

错误2：

等待（互斥)关

键部分

等待（互斥）

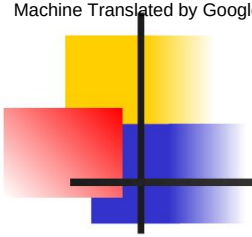
错误3：

忽略等待（互斥）

错误4：

忽略信号（互斥）

Machine Translated by Google



监控管程

霍尔(1974)、布林奇·汉森(1975)

Machine Translated by Google

PCP:监控

监控生产者消费者条件满,空;整数计数;程序

插入（项目:整数）;开始如果计
数=N则等待（满）;插
入项目（项目）;计数:=计数+1;如果计数=1则信号
（空）;
结束;函数删除:整数;开始如果计数=0则等
待（空）;删除=删除项目;计
数:=计数-1;如果计数=N-1
则信号（满）;结束;计数:=0;结束监控;

生产者程序;开始直至 true 才开
始
item=produce_item;

ProducerConsumer.insert(item);
结束结束;消费者程序;开始直至 true 才开始

item=ProducerConsumer.remove;consume_item(item);
结束结束;

Machine Translated by Google

监控器的实现

类型监视器名称=监视器变量声明过程entryP1
（…）;开始

…

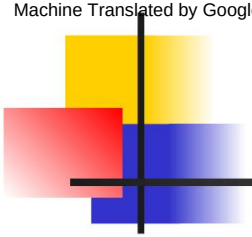
结束;
程序 entryP2(…);开始…结束;

…

程序 entryPn(…); 开始 … 结束;

开始初
始化代码结束

Machine Translated by Google



经典 IPC 问题

哲学家就餐问题（DPP)*

读者和作者问题（RWP)*

沉睡的理发师问题(SBP)*

Machine Translated by Google

哲学家就餐问题

1965 年,迪克斯特拉

哲学家吃饭/思考

吃饭需要2把叉子

每次选择一个叉子

如何防止死锁

Machine Translated by Google

哲学家就餐问题

```
#define N 5 void
philosopher(int i){ while(TRUE)
  { think();

      take_fork(i);
      take_fork((i+1)%N); eat();

      put_fork(i);
      put_fork((i+1)%N);

}}
```

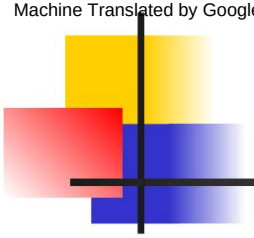
Machine Translated by Google

哲学家就餐问题

```
#define N 5
#define LEFT (i+N-1)%N #define
RIGHT (i+1)%N #define THINKING
0 #define HUNGRY 1 #define
EATING 2 typedef int
semaphore; int state[N]; 信
号量 mutex=1; 信号量 s[N]; void
philosopher(int)
{ while(TRUE){ think();
take_forks(i); eat();
put_forks(i);
}
}
```

```
void take_forks(int i)
  { down(&mutex);
  state[i]=HUNGRY; test(i);
  up(&mutex);
  down(&s[i]); }

void put_forks(int i)
  { down(&mutex);
  state[i]=THINKING;
  test(LEFT); test(RIGHT);
  up(&mutex); }

void test(int i){
  if(state[i]==HUNGRY && state[LEFT]!=EATING &&
     state[RIGHT]!=EATING)
    { state[i]=EATING;
    向上
(&s[i]); }}
```

Machine Translated by Google



读者和作者的问题

1971年,库尔图瓦:

民航订票系统

Machine Translated by Google

RWP:解决方案 1

```
typedef int semaphore; 信
号量 mutex=1; 信号量
db=1; int rc=0; void

reader(void)
   { while(TRUE)
      { down(&mutex);
      rc=rc+1;
      if(rc==1) down(&db);
      up(&mutex);
      read_data_base();
       down(&mutex);
      rc=rc-1;
      if (rc==0) up(&db);
      up(&mutex);
      use_data_read();
}}
```

```
void writer(void)
   { while(TRUE)
      { think_up_data();
      down(&db);
      write_data_base();
      up(&db);

} }
```

Machine Translated by Google

RWP:解决方案 2

程序读者和作家;var readcount，writecount:整数;x,y,z,wsem,rsem:信号量（1）;程序读者;开始重复等待（z）;等待（rsem）;等待（x）;readcount:=

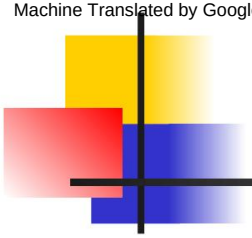
readcount + 1;如果 readcount = 1 则等待（wsem）;信号（x）;信号（rsem）;信号（z）;

READUNIT;等待（x）;readcount:= readcount-1;如果 readcount=0则信号（wsem）;信号（x）;永远结束;

z 是否必须?

程序编写器;开始重复等待（y）;写入计数:=写入计数+1;如果写入计数=1则等待（rsem）;信号（y）;等待（wsem）;WRITEUNIT;信号（wsem）;等待（y）;写入计数:=写入计数-1;如果写入计数=0则信号（rsem）;信号（y）;永远结束;开始读取计数,写入计数:=0;parbegin reader;writer;parend 结束。

Machine Translated by Google



睡着的理发师问题

Machine Translated by Google

睡着的理发师问题

```
#define CHAIRS 5 typedef
int semaphore; 信号量
customers=0; 信号量 barbers=0; 信
号量 mutex=1; int waiting =0;
void barber(void){ while(TRUE)
{ down(&customers);
down(&mutex);
      waiting=waiting-1;
            up(&barbers); up(&mutex);
              cut_hair();
} }
```

Machine Translated by Google

睡着的理发师问题

```
#define CHAIRS 5 typedef
int semaphore; 信号量
customers=0; 信号量 barbers=0; 信
号量 mutex=1; int waiting =0;
void barber(void){ while(TRUE)
{ down(&customers);
down(&mutex);
      waiting=waiting-1;
           up(&barbers); up(&mutex);
            cut_hair();
} }
```

```
void customer(void)
  { down(&mutex);
  if(waiting<CHAIRS)
    { waiting=waiting+1;
    up(&customers);
    up(&mutex);
    down(&barbers);
    get_haircut(); }

  else{ up(&mutex); } }
```

Machine Translated by Google

同步方法：概括

互斥

信号量信号量集

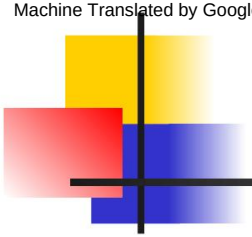
条件变量

事件计数器

监控

消息传递

Machine Translated by Google



替代同步方法

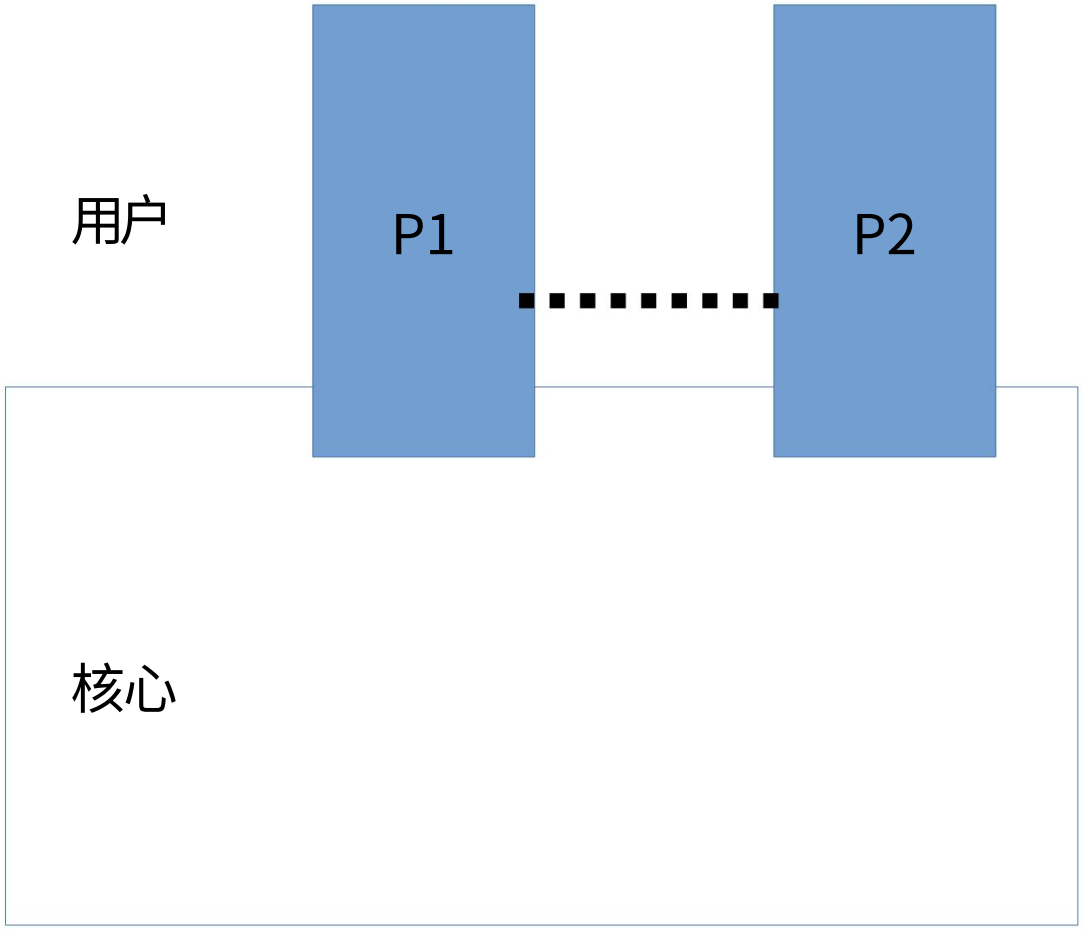
事务内存

函数式编程语言

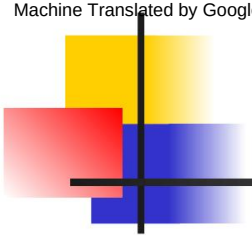
…

Machine Translated by Google

进程间数据通信



Machine Translated by Google



进程间数据通信

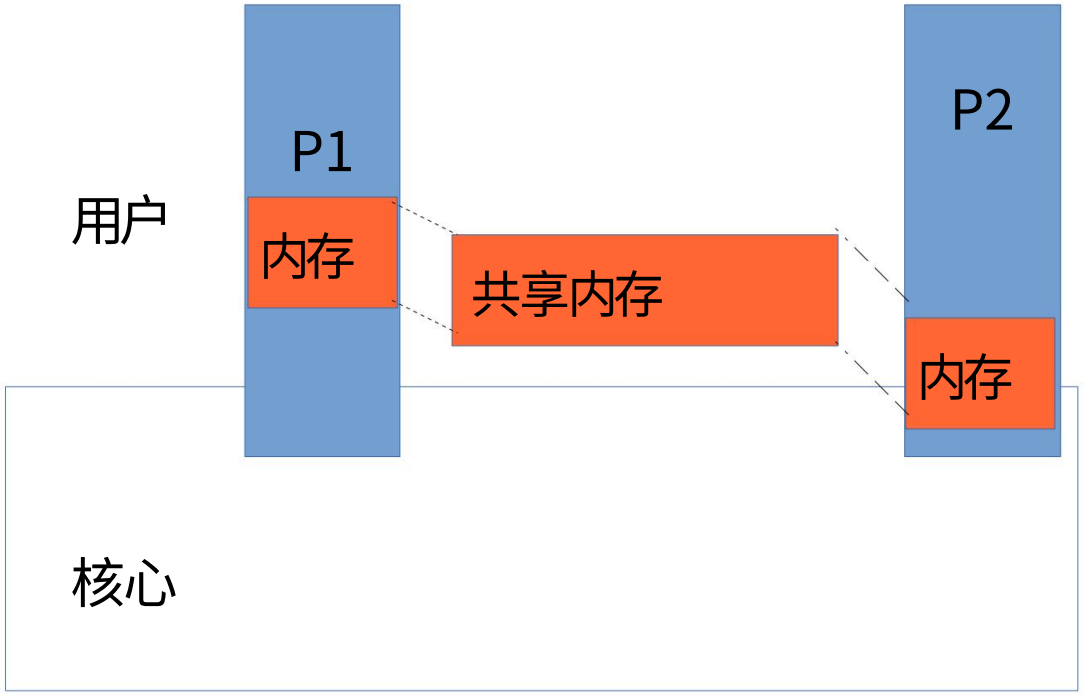
共享内存系统消息传递系统管道

系统远程过程调用

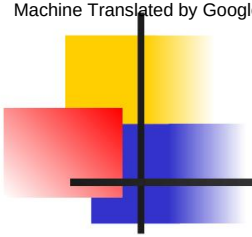
Machine Translated by Google

共享内存系统

可由多个程序同时访问的内存，目的是提供它们之间的通信或避免冗余副本



Machine Translated by Google



信息传递系统

发送（目标,&消息） 接收（源,&消息）

Machine Translated by Google

PCP:消息传递

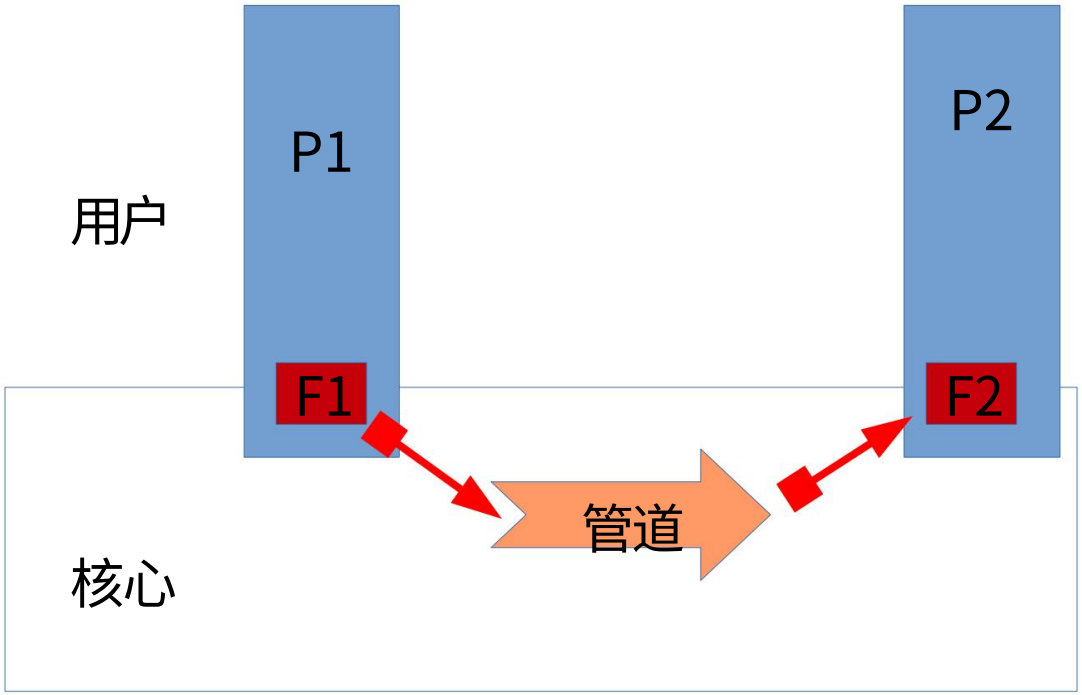
```
#define N 100 void
production(void){ int item;
    message
    m; while(TRUE)

    { item=produce_item(); 接收
        (consumer,&m);
        build_message(&m,item); 发送
        (consumer,&m); } }
```

```
void consumer(void){ int
    item,i; 消息
    m; for(i=0;i<N;i+
    +) send(producer,&m);
      while(TRUE){ 接收
    (producer,&m);
        item=extract_item(&m);
    send(producer,&m); 消费
    _item(item); } }
```

Machine Translated by Google

管道系统

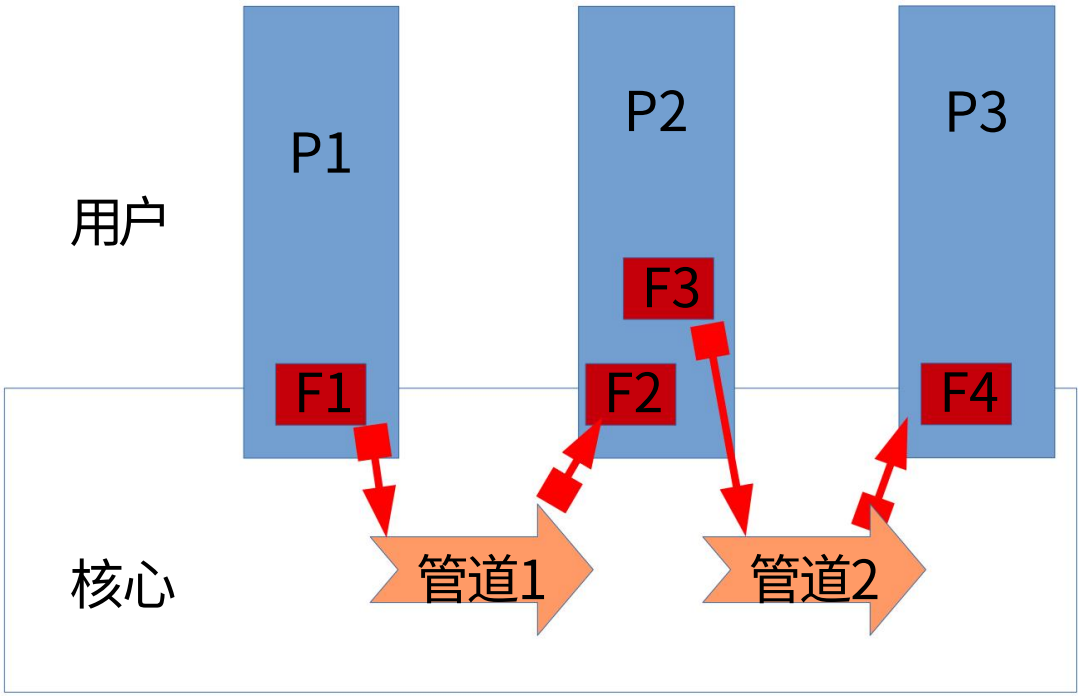
一组串联的数据处理单元,其中一个单元的输出是下一个单元的输入



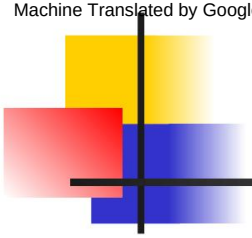
Machine Translated by Google

管道系统

cat 文件1 文件2 文件3 | grep “root” | wc -l

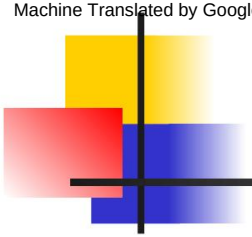


Machine Translated by Google



远程过程调用

Machine Translated by Google



概括

竞争条件

互斥

同步通信

经典的 IPC 问题

Machine Translated by Google

问答？

Machine Translated by Google