Synchronization: Basics

Introduction to Computer Systems
Nov. 28st

叶炜宁

Today

- 多线程程序中的共享变量
- 信号量
- 生产者-消费者问题

线程内存模型

- 每个线程都有它自己独立的线程上下文，包括线程ID、栈、栈指针、程序计数器、条件码和通用目的寄存器值。

线程内存模型

- 每个线程都有它自己独立的线程上下文，包括线程ID、栈、栈指针、程序计数器、条件码和通用目的寄存器值。
- 每个线程与其他线程一起共享进程上下文的剩余部分。包括整个用户虚拟地址空间。线程也共享相同的打开文件的集合。

线程内存模型

- 每个线程都有它自己独立的线程上下文，包括线程ID、栈、栈指针、程序计数器、条件码和通用目的寄存器值。
- 每个线程与其他线程一起共享进程上下文的剩余部分。包括整个用户虚拟地址空间。线程也共享相同的打开文件的集合。
- 寄存器从不共享，而虚拟内存总是共享的。

线程内存模型

- 每个线程都有它自己独立的线程上下文，包括线程ID、栈、栈指针、程序计数器、条件码和通用目的寄存器值。
- 每个线程与其他线程一起共享进程上下文的剩余部分。包括整个用户虚拟地址空间。线程也共享相同的打开文件的集合。
- 寄存器从不共享，而虚拟内存总是共享的。
- 不同的线程栈不对其他线程设防，如果一个线程以某种方式得到一个指向其他线程栈的指针，那么他就可以读写这个栈的任何部分。

将变量映射到内存

- *全局变量*：定义在函数之外的变量，虚拟内存的读写区域只包含每个全局变量的一个实例，任何线程都可以引用。

将变量映射到内存

- *全局变量*：定义在函数之外的变量，虚拟内存的读写区域只包含每个全局变量的一个实例，任何线程都可以引用。
- *本地自动变量*：定义在函数内部，但是没有static属性的变量。每个线程的栈都包含它自己的所有本地自动变量的实例。

将变量映射到内存

- *全局变量*：定义在函数之外的变量，虚拟内存的读写区域只包含每个全局变量的一个实例，任何线程都可以引用。
- *本地自动变量*：定义在函数内部，但是没有static属性的变量。每个线程的栈都包含它自己的所有本地自动变量的实例。
- *本地静态变量*：定义在函数内部并有static属性的变量，和全局变量一样，虚拟内存的读写区域只包含每个全局变量的一个实例。

将变量映射到内存

- *全局变量*：定义在函数之外的变量，虚拟内存的读写区域只包含每个全局变量的一个实例，任何线程都可以引用。
- *本地自动变量*：定义在函数内部，但是没有static属性的变量。每个线程的栈都包含它自己的所有本地自动变量的实例。
- *本地静态变量*：定义在函数内部并有static属性的变量，和全局变量一样，虚拟内存的读写区域只包含每个全局变量的一个实例。
- *共享变量*：一个变量v是共享的，当且仅当他的一个实例被一个以上的线程引用。

共享变量

Variable instance	Referenced by main thread?	Referenced by peer thread 0?	Referenced by peer thread 1?
ptr	yes	yes	yes
cnt	no	yes	yes
i.m	yes	no	no
msgs.m	yes	yes	yes
myid.p0	no	yes	no
myid.p1	no	no	yes

```
char **ptr;  /* global var */
int main(int main, char *argv[]) {
  long i; pthread_t tid;
  char *msgs[2] = {"Hello from foo",
                   "Hello from bar" };
    ptr = msgs;
    for (i = 0; i < 2; i++)
        Pthread_create(&tid,
            NULL, thread,(void *)i);
    Pthread_exit(NULL);}
```

```
void *thread(void *vargp)
{
  long myid = (long)vargp;
  static int cnt = 0;

  printf("[%ld]:  %s (cnt=%d)\n",
         myid, ptr[myid], ++cnt);
  return NULL;
}
```

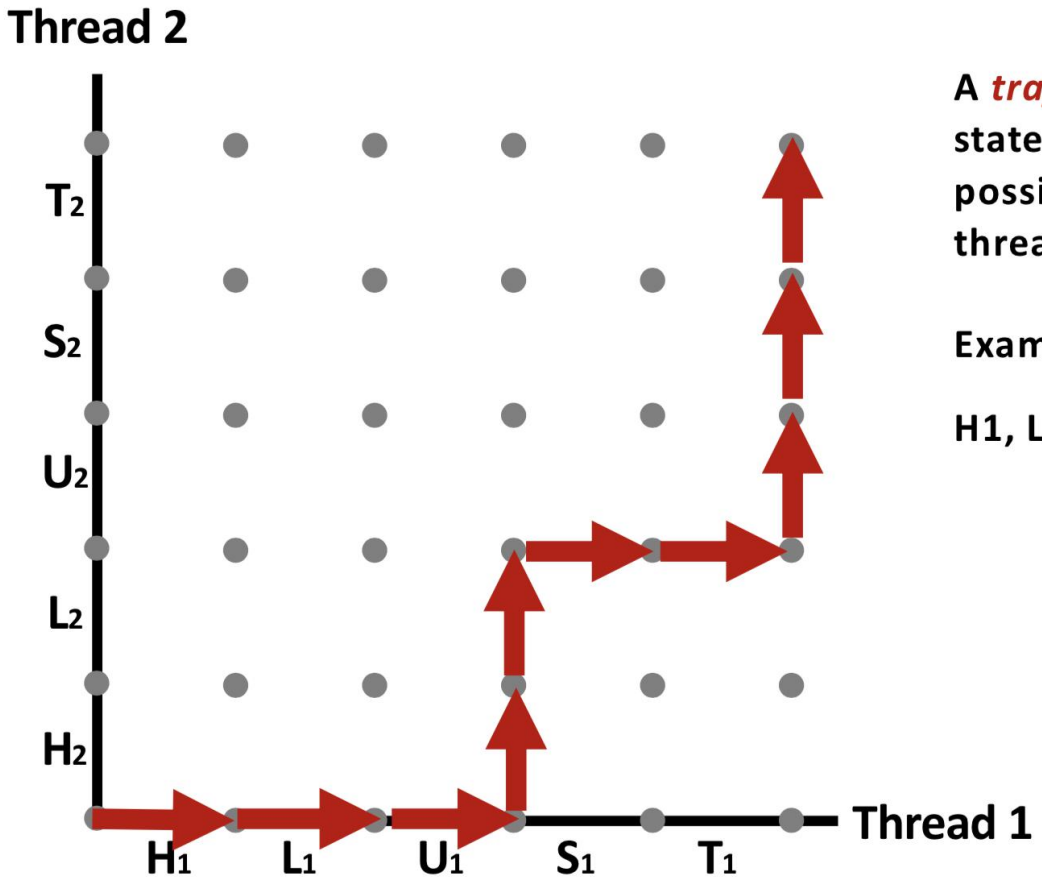
同步错误

- 一般而言，你没有办法预测操作系统是否将为你的线程选择一个正确的顺序。

i (thread)	instr$_i$	%rdx$_1$	%rdx$_2$	cnt
1	H_1	-	-	0
1	L_1	0	-	0
1	U_1	1	-	0
1	S_1	1	-	1
2	H_2	-	-	1
2	L_2	-	1	1
2	U_2	-	2	1
2	S_2	-	2	2
2	T_2	-	2	2
1	T_1	1	-	2

i (thread)	instr$_i$	%rdx$_1$	%rdx$_2$	cnt
1	H_1	-	-	0
1	L_1	0	-	0
1	U_1	1	-	0
2	H_2	-	-	0
2	L_2	-	0	0
1	S_1	1	-	1
1	T_1	1	-	1
2	U_2	-	1	1
2	S_2	-	1	1
2	T_2	-	1	1

进度图

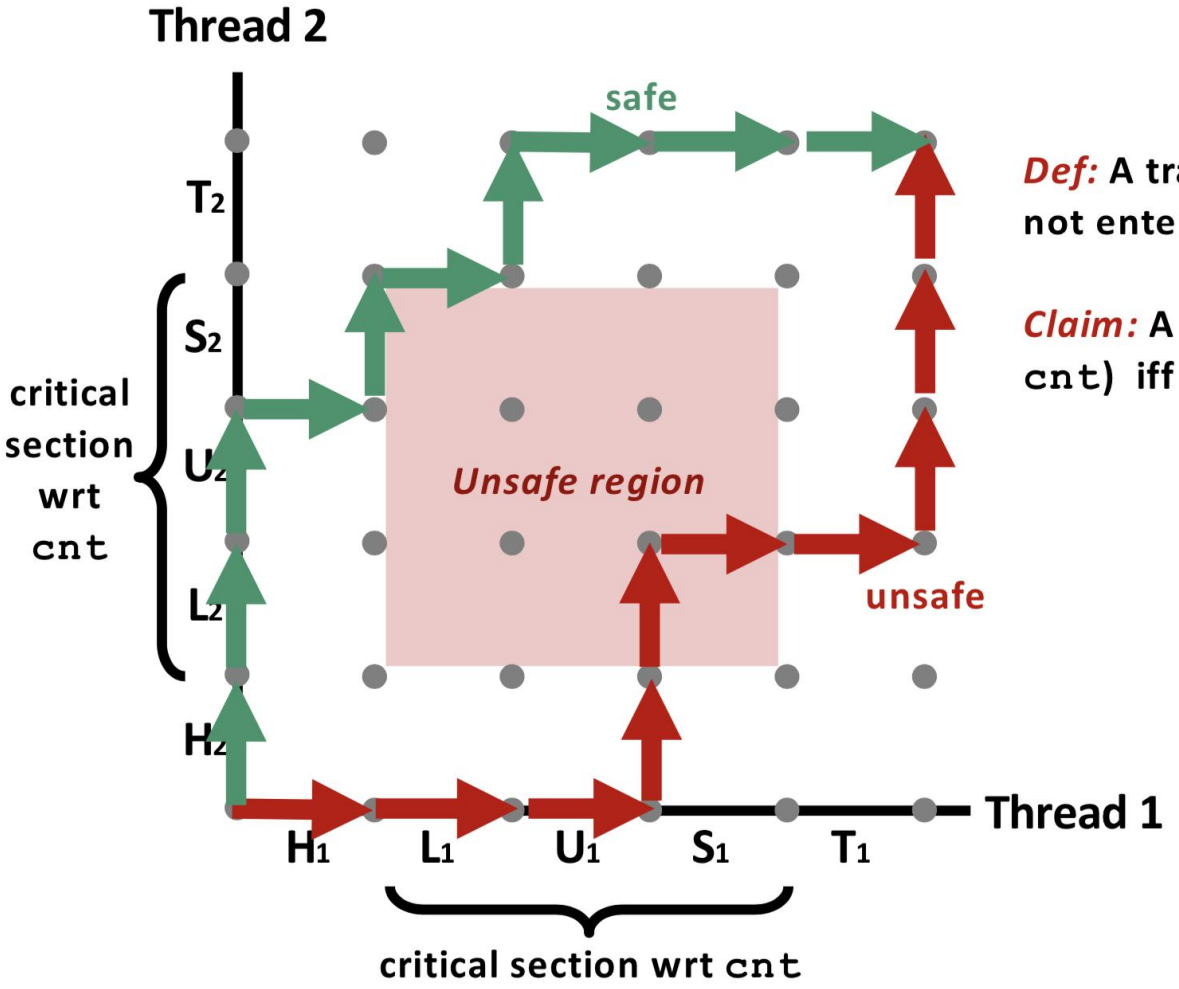


A *trajectory* is a sequence of legal state transitions that describes one possible concurrent execution of the threads.

Example:

H1, L1, U1, H2, L2, S1, T1, U2, S2, T2

进度图



Def: A trajectory is *safe* iff it does not enter any unsafe region

Claim: A trajectory is correct (wrt `cnt`) iff it is safe

信号量

- 信号量s是具有非负整数值的全局变量，只能由两种特殊的操作来处理。

信号量

- **信号量s是具有非负整数值的全局变量，只能由两种特殊的操作来处理。**
  - P(s)，如果s非零，那么就减1直接返回。否则挂起线程，直到s非零，V操作会重启这个线程，重启后，将s减1再返回。

信号量

- **信号量s是具有非负整数值的全局变量，只能由两种特殊的操作来处理。**
  - P(s)，如果s非零，那么就减1直接返回。否则挂起线程，直到s非零，V操作会重启这个线程，重启后，将s减1再返回。
  - V(s)，将s加1，如果有线程阻塞在P操作等待s变成非零，那么V操作会重启这些线程中的某一个，但你无法预测重启的是哪个线程。

信号量

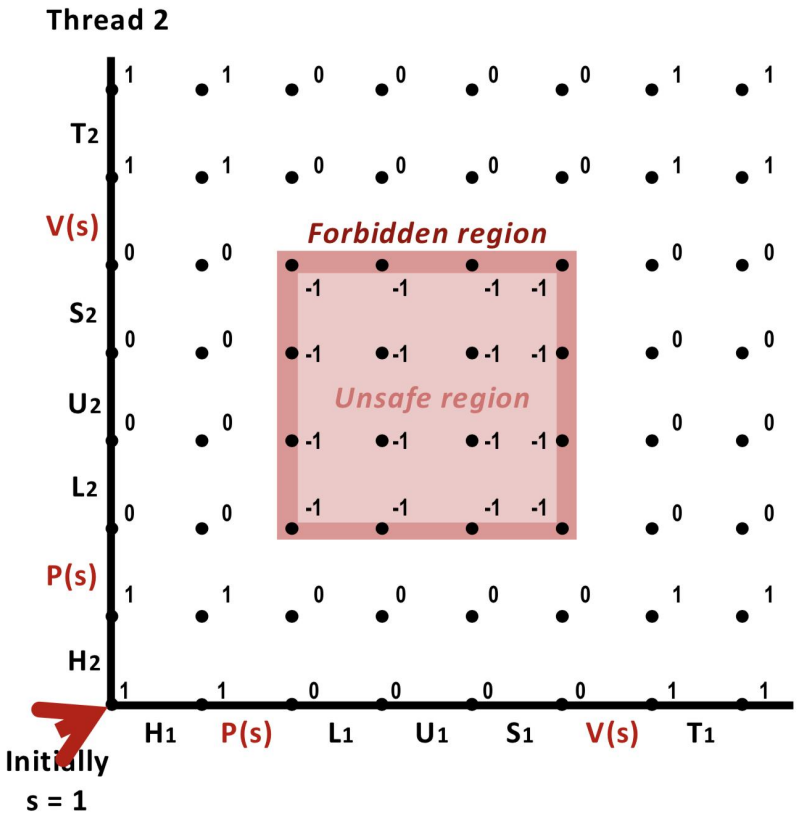
- **信号量s是具有非负整数值的全局变量，只能由两种特殊的操作来处理。**
  - P(s)，如果s非零，那么就减1直接返回。否则挂起线程，直到s非零，V操作会重启这个线程，重启后，将s减1再返回。
  - V(s)，将s加1，如果有线程阻塞在P操作等待s变成非零，那么V操作会重启这些线程中的某一个，但你无法预测重启的是哪个线程。
- **P、V操作都是原子的。**

信号量

- 信号量s是具有非负整数值的全局变量，只能由两种特殊的操作来处理。
  - P(s)，如果s非零，那么就减1直接返回。否则挂起线程，直到s非零，V操作会重启这个线程，重启后，将s减1再返回。
  - V(s)，将s加1，如果有线程阻塞在P操作等待s变成非零，那么V操作会重启这些线程中的某一个，但你无法预测重启的是哪个线程。
- P、V操作都是原子的。
- P、V的定义保证正在运行的程序不会让s变为负数。

使用信号量来实现互斥

- 将每个共享变量（或一组相关的共享变量）与一个信号量s（初识为1）联系起来。然后用P、V将相应的临界区包围起来。




```
for (i = 0; i < niters; i++) {
    P(&mutex);
    cnt++;
    V(&mutex);
}
```

goodcnt.c

生产者–消费者问题

- 生产者和消费者线程共享一个有n个槽的有限缓冲区。

生产者–消费者问题

- 生产者和消费者线程共享一个有n个槽的有限缓冲区。
- 生产者反复生成新的项目，并插入到缓冲区中；消费者不断从缓冲区取出这些项目，然后使用它们。

生产者–消费者问题

- 生产者和消费者线程共享一个有n个槽的有限缓冲区。
- 生产者反复生成新的项目，并插入到缓冲区中；消费者不断从缓冲区取出这些项目，然后使用它们。
- 插入和取出项目涉及更新共享变量，所以我们要保证对缓冲区的访问是互斥的。

生产者–消费者问题

- 生产者和消费者线程共享一个有n个槽的有限缓冲区。
- 生产者反复生成新的项目，并插入到缓冲区中；消费者不断从缓冲区取出这些项目，然后使用它们。
- 插入和取出项目涉及更新共享变量，所以我们要保证对缓冲区的访问是互斥的。
- 如果缓冲区是满的，那么生产者必须等待直到有一个槽位变为可用；如果缓冲区是空的，那么消费者必须等待直到有一个项目变为可用。

SBUF

```
#include "csapp.h"

typedef struct {
    int *buf;      /* Buffer array                         */
    int n;         /* Maximum number of slots              */
    int front;     /* buf[front+1 (mod n)] is first item */
    int rear;      /* buf[rear]   is last item             */
    sem_t mutex;   /* Protects accesses to buf             */
    sem_t slots;   /* Counts available slots               */
    sem_t items;   /* Counts available items               */
} sbuf_t;

void sbuf_init(sbuf_t *sp, int n);
void sbuf_deinit(sbuf_t *sp);
void sbuf_insert(sbuf_t *sp, int item);
int sbuf_remove(sbuf_t *sp);
```

```
/* Create an empty, bounded, shared FIFO buffer with n slots */
void sbuf_init(sbuf_t *sp, int n)
{
    sp->buf = Calloc(n, sizeof(int));
    sp->n = n;                      /* Buffer holds max of n items */
    sp->front = sp->rear = 0;       /* Empty buffer iff front == rear */
    Sem_init(&sp->mutex, 0, 1); /* Binary semaphore for locking */
    Sem_init(&sp->slots, 0, n); /* Initially, buf has n empty slots */
    Sem_init(&sp->items, 0, 0); /* Initially, buf has zero items */
}

/* Clean up buffer sp */
void sbuf_deinit(sbuf_t *sp)
{
    Free(sp->buf);
}
```

```
/* Insert item onto the rear of shared buffer sp */
void sbuf_insert(sbuf_t *sp, int item)
{
    P(&sp->slots);                  /* Wait for available slot */
    P(&sp->mutex);                  /* Lock the buffer         */
    if (++sp->rear >= sp->n)        /* Increment index (mod n) */
        sp->rear = 0;
    sp->buf[sp->rear] = item;       /* Insert the item         */
    V(&sp->mutex);                  /* Unlock the buffer       */
    V(&sp->items);                  /* Announce available item */
}
```

sbuf.c

```
/* Remove and return the first item from buffer sp */
int sbuf_remove(sbuf_t *sp)
{
    int item;
    P(&sp->items);                  /* Wait for available item */
    P(&sp->mutex);                  /* Lock the buffer         */
    if (++sp->front >= sp->n)       /* Increment index (mod n) */
        sp->front = 0;
    item = sp->buf[sp->front];      /* Remove the item         */
    V(&sp->mutex);                  /* Unlock the buffer       */
    V(&sp->slots);                  /* Announce available slot */
    return item;
}
```

sbuf.c