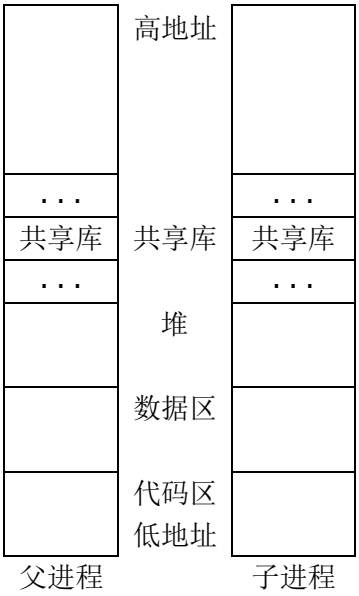
ICS 第十二章

1. volatile 保证定义的变量存放在内存中，而不总是在寄存器里。右侧为两个进程的地址空间。请在合适的位置标出变量 gCount、vCount 与 lCount 的位置。如果一个量出现多次，那么就标多次。

```
long gCount = 0;
void *thread(void *vargp) {
    volatile long vCount = *(long *)vargp;
    static long lCount = 0;
    gCount++; vCount++; lCount++;
    printf("%ld\n", gCount+vCount+lCount);
    return NULL;
}
int main() {
    long var; pthread_t tid1, tid2;
    scanf("%ld", &var);
    fork();
    pthread_create(&tid1,NULL,thread,&var);
    pthread_create(&tid2,NULL,thread,&var);
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    return 0;
}
```



2. 下面的程序会引发竞争。一个可能的输出结果为 2 1 2 2。解释输出这一结果的原因。

```
long foo = 0, bar = 0;

void *thread(void *vargp) {
    foo++; bar++;
    printf("%ld %ld ", foo, bar); fflush(stdout);
    return NULL;
}

int main() {
    pthread_t tid1, tid2;
    pthread_create(&tid1, NULL, thread, NULL);
    pthread_create(&tid2, NULL, thread, NULL);
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    return 0;
}
```

3. 信号量 w,x,y,z 均被初始化为 1。下面的两个线程运行时可能会发生死锁。给出发生死锁的执行顺序。

线程 1	①P(w) ②P(x) ③P(y) ④P(z) ⑤V(w) ⑥V(x) ⑦V(y) ⑧V(z)
线程 2	ⅠP(x) ⅡP(z) ⅢP(y) ⅣP(w) ⅤV(x) ⅥV(y) ⅦV(w) ⅧV(z)

4. 某次考试有 30 名学生与 1 名监考老师，该教室的门很狭窄，每次只能通过一人。考试开始前，老师和学生进入考场（有的学生来得比老师早），当人来齐以后，老师开始发放试卷。拿到试卷后，学生就可以开始答卷。学生可以随时交卷，交卷后就可以离开考场。当所有的学生都上交试卷以后，老师才能离开考场。

请用信号量与 PV 操作，解决这个过程中的同步问题。所有空缺语句均为 PV 操作。

全局变量：
stu_count: int 类型，表示考场中的学生数量，初值为 0
信号量：
mutex_stu_count: 保护全局变量，初值为 1
mutex_door: 保证门每次通过一人，初值为_____
mutex_all_present: 保证学生都到了，初值为_____
mutex_all_handin: 保证学生都交了，初值为_____
mutex_test[30]: 表示学生拿到了试卷，初值均为_____

```
Teacher:  // 老师
    (1)
    从门进入考场
    (2)

    (3) // 等待同学来齐
    for (i = 1; i <= 30; i++)
        (4) // 给 i 号学生发放试卷

    (5) // 等待同学将试卷交齐
    (6)
    从门离开考场
    (7)
```

```
Student(x):  // x 号学生
    (8)
    从门进入考场
    (9)

    P(mutex_stu_count);
    stu_count++;
    if (stu_count == 30)
        (10)
    V(mutex_stu_count);

    (11) // 等待拿自己的卷子
    学生答卷

    P(mutex_stu_count);
    stu_count--;
    if (stu_count == 0)
        (12)
    V(mutex_stu_count);

    (13)
    从门离开考场
    (14)
```

5. 竞争

以下几段代码创建两个对等线程，并希望第一个线程输出 0，第二个输出 1；但有些代码会因为变量 myid 的竞争问题导致错误，请你判断哪些代码会在 myid 上存在竞争。如果不存在竞争，请你判断这段代码是否一定先输出 0 再输出 1？

A.

```
void *foo(void *vargp) {
    int myid;
    myid = *(int *)vargp;
    free(vargp);
    printf("Thread %d\n", myid);
}
```

```
int main() {
    pthread_t tid[2];
    int i, *ptr;
    for (i = 0; i < 2; ++i) {
        ptr = malloc(sizeof(int));
        *ptr = i;
        pthread_create(&tid[i], 0,
foo, ptr);
    }
    pthread_join(tid[0], 0);
    pthread_join(tid[1], 0);
}
```

B.

```
void *foo(void *vargp) {
    int myid;
    myid = *(int *)vargp;
    printf("Thread %d\n", myid);
}
```

```
int main() {
    pthread_t tid[2];
    int i;
    for (i = 0; i < 2; ++i) {
        pthread_create(&tid[i], 0,
foo, &i);
    }
    pthread_join(tid[0], 0);
    pthread_join(tid[1], 0);
}
```

C.

```
void *foo(void *vargp) {
    int myid;
    myid = (int)vargp;
    printf("Thread %d\n", myid);
}
```

```
int main() {
    pthread_t tid[2];
    int i;
    for (i = 0; i < 2; ++i) {
        pthread_create(&tid[i], 0,
foo, i);
    }
    pthread_join(tid[0], 0);
    pthread_join(tid[1], 0);
}
```

D.

```
sem_t s;

void *foo(void *vargp) {
    int myid;
    P(&s);
    myid = *(int *)vargp;
    printf("Thread %d\n", myid);
    V(&s);
}
```

```
int main() {
    pthread_t tid[2];
    int i;
    sem_init(&s, 0, 1);
    for (i = 0; i < 2; ++i) {
        pthread_create(&tid[i], 0,
foo, &i);
    }
    pthread_join(tid[0], 0);
    pthread_join(tid[1], 0);
}
```

E.

```
sem_t s;

void *foo(void *vargp) {
    int myid;
    myid = *(int *)vargp;
    printf("Thread %d\n", myid);
    V(&s);
}
```

```
int main() {
    pthread_t tid[2];
    int i;
    sem_init(&s, 0, 0);
    for (i = 0; i < 2; ++i) {
        pthread_create(&tid[i], 0,
foo, &i);
        P(&s);
    }
    pthread_join(tid[0], 0);
    pthread_join(tid[1], 0);
}
```

6. 读者写者问题

一组并发的线程想要访问一个共享对象，有无数的读者和写者想要访问共享对象，读者可以和其它读者同时访问，而写者必须独占对象。以下是第一类读者写者问题的代码。

```
void reader() {
    P(&mutex);
    readcnt++;
    if (readcnt == 1)
        P(&w); /* line a */
    V(&mutex);
    /* reading... line b */
    P(&mutex);
    readcnt--;
    if (readcnt == 0)
        V(&w);
    V(&mutex);
}
```

```
void writer() {
    P(&w); /* line c */
    /* writing... line d */
    V(&w);
}
```

(1) 假设在时刻 0~4 分别有五个读、写者到来；它们的顺序为 R1, R2, W1, R3, W2；已知读操作需要等待 3 个周期，写操作需要等待 5 个周期；假设忽略其他语句的执行时间、线程的切换/调度的时间开销，因此在任意时刻，每个读者、写者只能处在上面标注好的 abcd 四处语句，请你分析这五个读者/写者线程终止的顺序？

(2) 基于(1)的发现，这段代码容易导致饥饿，于是一位同学规定：当有写者在等待时，后来的读者不能进行读操作，写出了第二类读者写者问题的代码如下（所有信号量初始化为 1）：

```
void reader() {
    P(&r); /* line a */
    P(&mutex);
    readcnt++;
    if (readcnt == 1)
        P(&w); /* line b */
    V(&mutex);
    V(&r);
    /* reading... line c */
    P(&mutex);
    readcnt--;
    if (readcnt == 0)
        V(&w);
    V(&mutex);
}
```

```
void writer() {
    P(&mutex);
    writecnt++;
    if (writecnt == 1)
        P(&r); /* line d */
    V(&mutex);
    P(&w); /* line e */
    /* writing... line f */
    V(&w);
    P(&mutex);
    writecnt--;
    if (writecnt == 0)
        V(&r);
    V(&mutex);
}
```

这段代码会导致死锁，请你列举一种可能导致死锁的线程控制流，并提出一种改进的方案。

(3) 在修改了(2)中的问题后，请你基于第二类读者写者问题的代码再回答(1)中的题目。

7. 线程安全函数

吴用功同学找了一个找素数的函数 next_prime，ta 在实现这个函数的线程安全版本 ts_next_prime 的时候出现了问题，请你帮助 ta。

```
struct big_number *next_prime(struct big_number current_prime) {
    static struct big_number next;
    next = current_prime;
    addOne(next);
    while(!isNotPrime(next))
        addOne(next);
    return &next;
```

```
}

struct big_number *ts_next_prime(struct big_number current_prime) {
    return next_prime(current_prime);
}
```

A. 现在的 ts_next_prime 为什么线程不安全？

B. 下面的代码是否线程安全？

```
struct big_number *ts_next_prime(struct big_number current_prime)
{
    struct big_number *value_ptr;

    P(&mutex); /* mutex is initialized to 1*/
    value_ptr = next_prime(current_prime);
    V(&mutex);

    return value_ptr;
}
```

C. 请使用 lock&copy 技术实现线程安全的 ts_next_prime

```
sem_t mutex;
struct big_number *ts_next_prime(struct big_number current_prime)
{
    struct big_number *value_ptr;
    struct big_number *ret_ptr;
    P(&mutex); /* mutex is initiallized to 1*/
    ret_ptr = _____________________;
    value_ptr = next_prime(current_prime);
    ____________________________________;
    V(&mutex);

    return ret_ptr;
}
```