OPERATING SYSTEMS

Monitors & Deadlocks

Operating Systems

郑贵锋 博士
中山大学计算机学院
zhenggf@mail.sysu.edu.cn
https://gitee.com/code_sysu

中山大學 SUN YAT-SEN UNIVERSITY

- **目录**
  - 同步硬件
  - 互斥锁
  - 信号量
  - 经典问题
  - 管程
  - 死锁
  - **同步示例**
    - Linux
    - POSIX
    - Solaris
    - Windows

Synchronization Examples

- **Linux**
  - Linux版本
    - 在内核版本2.6之前，禁用中断以实现短临界区（非抢占）。
    - 版本2.6及更高版本，完全抢占。
  - Linux在内核中提供了几种不同的同步机制：
    - __sync_fetch_** 类型的同步函数
    - 自旋锁
    - 互斥锁
    - 信号量
    - 自旋锁和信号量的读者写者版本
  - 在单CPU系统上，自旋锁被启用和禁用内核抢占取代。

Synchronization Examples

- **Linux**
  - Linux __sync_** fetch 类型的同步函数
    - __sync_**类型的操作以原子方式执行

```
type __sync_fetch_and_add (type *ptr, type value, ...)
type __sync_fetch_and_sub (type *ptr, type value, ...)
type __sync_fetch_and_or (type *ptr, type value, ...)
type __sync_fetch_and_and (type *ptr, type value, ...)
type __sync_fetch_and_xor (type *ptr, type value, ...)
type __sync_fetch_and_nand (type *ptr, type value, ...)

type __sync_add_and_fetch (type *ptr, type value, ...)
type __sync_sub_and_fetch (type *ptr, type value, ...)
type __sync_or_and_fetch (type *ptr, type value, ...)
type __sync_and_and_fetch (type *ptr, type value, ...)
type __sync_xor_and_fetch (type *ptr, type value, ...)
type __sync_nand_and_fetch (type *ptr, type value, ...)
```

    - 类型是1、2、4或8字节的整数：

```
int8_t / uint8_t
int16_t / uint16_t
int32_t / uint32_t
int64_t / uint64_t
```

Synchronization Examples

- **Linux**
  - Linux __sync_** fetch 类型的同步函数
    - 算法 18-1: syn-fetch.c

实现20个线程，同步计数

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
static int count = 0;

void *test_func(void *arg) {
    for (int i = 0; i < 10000; ++i)
        __sync_fetch_and_add(&count, 1);
    return NULL;
}

int main(int argc, const char *argv[])
{
    pthread_t tid[20];
    int i = 0;
    for (i = 0; i < 20; ++i)
         pthread_create(&tid[i], NULL, &test_func, NULL);
    for (i = 0; i < 20; ++i)
        pthread_join(tid[i],NULL);
    printf("%d\n", count);
    return 0;
}
```

Synchronization Examples

- **Linux**
  - Linux __sync_** fetch 类型的同步函数
    - 算法 18-1: syn-fetch.c

实现20个线程，同步计数

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
static int count = 0;

void *test_func(void *arg) {
    for (int i = 0; i < 10000; ++i)
        __sync_fetch_and_add(&count, 1);
    return NULL;
}
```

```
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ gcc syn-fetch.c -o syn-fetch.o -lpthread
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ ./syn-fetch.o
count = 200000

isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$
```

```
         pthread_create(&tid[i], NULL, test_func, NULL);
    for (i = 0; i < 20; ++i)
        pthread_join(tid[i],NULL);
    printf("%d\n", count);
    return 0;
}
```

Synchronization Examples

- **Linux**
  - 其他Linux __sync_** 类型的同步函数
    - __sync_**类型的操作以原子方式执行。

`bool __sync_bool_compare_and_swap (type *ptr, type oldval type newval, ...)`

- 如果*ptr==oldval，则设置*ptr=newval并返回TRUE，否则设置oldval=*ptr并返回FALSE

`type __sync_val_compare_and_swap (type *ptr, type oldval type newval, ...)`

- 如果*ptr==oldval，则设置*ptr=newval并返回oldval，否则返回*ptr

`__sync_synchronize (...)`

- 设置一个完整的内存屏障

`type __sync_lock_test_and_set (type *ptr, type value, ...)`

- 设置*ptr=value并返回*ptr的旧值

`void __sync_lock_release (type *ptr, ...)`

- 设置*ptr=0

Synchronization Examples

- **Linux**
  - Linux __sync_** 类型的同步函数
    - 算法18-2: syn-compare-test.c (1)

比较与交换同步操作

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, const char *argv[])
{
    int value, oldval, newval;
    int ret;

    value = 200000; oldval = 123456; newval = 654321;
    printf("value = %d, oldval = %d, newval = %d\n", value, oldval, newval);
    printf("bool __sync_bool_compare_and_swap(&value, oldval, newval)\n");
    ret =  __sync_bool_compare_and_swap(&value, oldval, newval);
    printf("return ret = %d, value = %d, oldval = %d, newval = %d\n\n", ret,
value, oldval, newval);

    value = 200000; oldval = 200000; newval = 654321;
    printf("value = %d, oldval = %d, newval = %d\n", value, oldval, newval);
    printf("bool __sync_bool_compare_and_swap(&value, oldval, newval)\n");
    ret =  __sync_bool_compare_and_swap(&value, oldval, newval);
    printf("return ret = %d, value = %d, oldval = %d, newval = %d\n\n", ret,
value, oldval, newval);
```

Synchronization Examples

- **Linux**
  - Linux __sync_** 类型的同步函数
    - 算法18-2: syn-compare-test.c (2)

比较与交换同步操作

```
    value = 200000; oldval = 123456; newval = 654321;
    printf("value = %d, oldval = %d, newval = %d\n", value, oldval, newval);
    printf("type ___sync_val_compare_and_swap(&value, oldval, newval)\n");
    ret =  __sync_val_compare_and_swap(&value, oldval, newval);
    printf("return ret = %d, value = %d, oldval = %d, newval = %d\n\n", ret,
value, oldval, newval);

    value = 200000; oldval = 200000; newval = 654321;
    printf("value = %d, oldval = %d, newval = %d\n", value, oldval, newval);
    printf("type ___sync_val_compare_and_swap(&value, oldval, newval)\n");
    ret =  __sync_val_compare_and_swap(&value, oldval, newval);
    printf("return ret = %d, value = %d, oldval = %d, newval = %d\n\n", ret,
value, oldval, newval);
```

Synchronization Examples

- **Linux**
  - Linux __sync_** 类型的同步函数
    - 算法18-2: syn-compare-test.c (3)

测试与设置同步操作

```
    value = 200000; newval = 654321;
    printf("value = %d, newval = %d\n", value,  newval);
    printf("type __sync_lock_test_and_set(&value, newval)\n");
    ret =  __sync_lock_test_and_set(&value, newval);
    printf("return ret = %d, value = %d, newval = %d\n\n", ret, value,
newval);

    value = 200000;
    printf("value = %d\n", value);
    printf("type __sync_lock_release(&value)\n");
    __sync_lock_release(&value);
    printf("value = %d\n\n",value);

    return 0;
}
```

Synchronization Examples

- **Linux**
  - Linux __sync_** 类型的同步函数
    - 算法18-2: syn-compare-test.c (3)

```
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ gcc syn-compare-test.c -o syn-compare-test.o
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ ./syn-compare-test.o
value = 200000, oldval = 123456, newval = 654321
bool __sync_bool_compare_and_swap(&value, oldval, newval)
return ret = 0, value = 200000, oldval = 123456, newval = 654321

value = 200000, oldval = 200000, newval = 654321
bool __sync_bool_compare_and_swap(&value, oldval, newval)
return ret = 1, value = 654321, oldval = 200000, newval = 654321

value = 200000, oldval = 123456, newval = 654321
type __sync_val_compare_and_swap(&value, oldval, newval)
return ret = 200000, value = 200000, oldval = 123456, newval = 654321

value = 200000, oldval = 200000, newval = 654321
type __sync_val_compare_and_swap(&value, oldval, newval)
return ret = 200000, value = 654321, oldval = 200000, newval = 654321

value = 200000, newval = 654321
type __sync_lock_test_and_set(&value, newval)
return ret = 200000, value = 654321, newval = 654321

value = 200000
type __sync_lock_release(&value)
value = 0

isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$
```

比较与交换

测试与设置

Synchronization Examples

- **Linux**
  - Linux自旋锁和内核抢占
    - Linux提供了自旋锁和信号量（以及这两种锁的读者写者版本）来在内核中加锁
      - 在SMP机器上，基本的锁定机制是自旋锁，内核的设计使得自旋锁只能保持很短的时间。
      - 在单处理器机器（如许多嵌入式系统）上，不适合使用自旋锁。它们被启用和禁用内核抢占取代。

单处理器	多处理器
禁用内核抢占	获取自旋锁
启用内核抢占	释放自旋锁

Synchronization Examples

- **Linux**
  - Linux自旋锁和内核抢占
    - Linux使用preempt_disable()和preempt_enable()系统调用来禁用和启用内核抢占。
    - 如果内核中运行的任务持有锁，那么内核是不可抢占的。
      - 系统中的每个任务都有一个线程信息结构，其中包含抢占计数preempt_count，以指示任务持有的锁的数量。抢占计数在获取锁时递增，在释放锁时递减。如果preempt_count的值大于0，则抢占在内核中运行的相应任务是不安全的。
    - 只有在锁（或禁用内核抢占）保持较短时间时，才会在内核中使用自旋锁以及启用和禁用内核抢占。当锁必须保持较长时间时，适合使用信号量或互斥锁。

Synchronization Examples

- **Linux**
  - Linux互斥锁
    - 原子整数的使用是受限的。在有多个变量导致可能的竞争条件的情况下，必须使用更复杂的加锁工具。
    - Linux中可以使用互斥锁来保护内核中的临界区。在这里，任务必须在进入临界区之前调用mutex_lock()函数，在退出临界区之后调用mutex_unlock()函数。如果互斥锁不可用，则调用mutex_lock()的任务将进入睡眠状态，并在锁的持有者调用mutex_unlock()时被唤醒。

Synchronization Examples

- **POSIX同步**
  - POSIX API/Pthreads
    - POSIX API可供程序员在用户级别使用，并且与操作系统无关。
      - 我们讨论的同步方法与内核内的同步有关，因此仅对内核开发者可用。
    - POSIX协议提供
      - 互斥锁
      - 信号量
      - 条件变量
      - 非便携扩展包括：
        - 读写锁
        - 自旋锁
    - 这些API最终是使用主机操作系统提供的工具实现的。它们被UNIX、Linux和macOS系统上的开发人员广泛用于线程创建和同步。

Synchronization Examples

- **POSIX同步**
  - POSIX互斥锁
    - 互斥锁用于保护代码的临界区，即线程在进入临界区之前获取锁，并在退出临界区时释放锁。
    - Pthreads将pthread_mutex_t数据类型用于互斥锁。使用pthread_mutex_init()函数创建互斥锁。

```
#include <pthread.h>

pthread_mutex_t mutex;

/*创建并初始化互斥锁*/
pthread_mutex_init (&mutex, NULL) ;
```

      - 第一个参数是指向互斥变量mutex的指针。通过将NULL作为第二个参数传递，我们将mutex初始化为其默认属性。

Synchronization Examples

- **POSIX同步**
  - POSIX互斥锁
    - 通过pthread_mutex_lock()和pthread_mutex_unlock()函数获取并释放互斥锁mutex。如果调用pthread_mutex_lock()时mutex不可用，则调用线程将被阻塞在 mutex 的等待队列中，直到持有者调用pthread_mutex_unlock()释放mutex
    - 以下代码说明了如何使用互斥锁保护临界区：

```
/*获取互斥锁*/
pthread_mutex_lock (&mutex) ;
临界区
/*释放互斥锁*/
pthread_mutex_unlock (&mutex) ;
剩余区
```

    - 所有互斥函数执行正确时将返回0；如发生错误，将返回非零错误代码。

Synchronization Examples

- **POSIX同步**
  - POSIX互斥锁
    - 其他一些pthread互斥锁函数

```
int pthread_mutex_trylock(pthread_mutex_t *mutex);

int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
                            const struct timespec *restrict abs_timeout);

int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Synchronization Examples

- **POSIX同步**
  - POSIX互斥锁
    - **算法** 18-3: syn-pthread-mutex.c (1)

```
/* Compiling:
   gcc syn-pthread-mutex.c -o syn-pthread-mutex.o -lpthread */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>

static int count = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

Synchronization Examples

- **POSIX同步**
  - POSIX互斥锁
    - **算法** 18-3: syn-pthread-mutex.c (2)

```
void *test_func_syn(void *arg) /* 同步测试函数 */
{
    for (int i = 0; i < 2; ++i) {
        pthread_mutex_lock(&mutex);
        count++;
        printf("count = %d\n", count);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}

void *test_func_asy(void *arg) /* 异步测试函数 */
{
    for (int i = 0; i < 2; ++i) {
        count++;
        printf("count = %d\n", count);
    }
    pthread_exit(NULL);
}
```

Synchronization Examples

- **POSIX同步**
  - POSIX互斥锁
    - 算法 18-3: syn-pthread-mutex.c (3)

```
int main(int argc, const char *argv[])
{
    int thread_num = 5;
    pthread_t tid[thread_num];
    int i = 0;

    if ((argc > 1) && (!strcmp(argv[1], "syn")))
        for (i = 0; i < thread_num; ++i)
            pthread_create(&tid[i], NULL, &test_func_syn, NULL);
    else
        for (i = 0; i < thread_num; ++i)
            pthread_create(&tid[i], NULL, &test_func_asy, NULL);

    for (i = 0; i < thread_num; ++i)
        pthread_join(tid[i],NULL);
    pthread_mutex_destroy(&mutex);
    printf("result count = %d\n\n", count);
    return 0;
}
```

Synchronization Examples

- **POSIX同步**
  - POSIX互斥锁

```
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ gcc syn-pthread-mutex.c -o syn-pthread-mutex.o -lpthread
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ ./syn-pthread-mutex.o syn
count = 1
count = 2
count = 3
count = 4
count = 5
count = 6
count = 7
count = 8
count = 9
count = 10
result count = 10

isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ ./syn-pthread-mutex.o asy
count = 1
count = 5
count = 3
count = 7
count = 2
count = 8
count = 6
count = 4
count = 9
count = 10
result count = 10

isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$
```

Synchronization Examples

- **POSIX同步**
  - POSIX信号量
    - POSIX SEM扩展指定了两种类型的命名和无名信号量。从内核的2.6版开始，Linux系统提供了对这两种类型的支持。
    - POSIX命名信号量
      - 函数sem_open()用于创建新的或打开现有的信号量：

```
#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>
sem_t *sem_open(const char *name, int oflag);
sem_t *sem_open(const char *name, int oflag, mode_t
mode, unsigned int value);
```

      - 例如：

```
sem_t *sem;
sem = sem_open("MYSEM", O_CREAT, 0666, 1);
```

        - 命名的信号量MYSEM被创建并初始化为1。它对其他进程具有读写访问权限。

Synchronization Examples

- **POSIX同步**
  - POSIX信号量
    - POSIX命名信号量
      - 多个不相关的进程可以通过简单地引用信号量的名称，轻松地使用一个通用的命名信号量作为同步机制。
      - 一旦创建了信号量MYSEM，其他进程使用相同参数调用sem = sem_open()，将获得该信号量的描述符sem。
      - 相应地，POSIX分别声明了信号量获取与释放操作sem_wait(sem)和sem_post(sem)
      - 以下说明了如何使用上面创建的命名信号量保护临界区：

```
/*获取信号量*/
sem_wait(sem);
临界区
/*释放信号灯*/
sem_post(sem);
...
sem_close(sem);
```

Synchronization Examples

- **POSIX同步**
  - POSIX信号量
    - POSIX无名信号量
      - 使用sem_init()函数创建并初始化无名信号量，3个参数
        (1) 指向信号量的指针
        (2) 表示共享级别的标志
        (3) 信号量的初始值

        ```
        int sem_init(sem_t*sem, int pshared, unsigned int value)
        ```
      - 例如：

        ```
        #include <semaphore.h>
        sem_t sem;
        /* 创建信号量并将其初始化为1 */
        sem_init(&sem, 0, 1);
        ```
        - 第2个参数pshared=0 表示此信号量只能由创建它的进程的线程共享。
        - 信号量设置初始值1

Synchronization Examples

- **POSIX同步**
  - POSIX信号量
    - POSIX无名信号量
      - POSIX无名信号量使用与命名信号量相同的sem_wait(sem)和sem_post(sem)操作对在描述符sem上的信号量进行获取与释放。
      - 以下说明了如何使用上面创建的无名信号量保护临界区：

        ```
        /*获取信号量*/
        sem_wait(&sem);
        临界区
        /*释放信号量*/
        sem_post(&sem);
        ...
        sem_destroy(&sem);
        ```
      - 通常在进程间同步时使用命名信号量，而无名信号量则用于线程间通信。

Synchronization Examples

- **POSIX同步**
  - POSIX信号量
    - **算法** 18-4: Syn-phread-sem-1.c: POSIX无名信号量演示

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <string.h>

sem_t mysem;
static int count = 0;

void *test_func_syn(void *arg)
{
    for (int i = 0; i < 2; ++i) {
        sem_wait(&mysem);
        count++;
        printf("count = %d\n", count);
        sem_post(&mysem);
    }
    pthread_exit(NULL);
}
```

Synchronization Examples

- **POSIX同步**
  - POSIX信号量
    - **算法** 18-4: Syn-phread-sem-1.c (2)

```
void *test_func_asy(void *arg)
{
    for (int i = 0; i < 2; ++i) {
        count++;
        printf("count = %d\n", count);
    }
    pthread_exit(NULL);
}

int main(int argc, const char *argv[])
{
    int thread_num = 5;
    pthread_t tid[thread_num];
    int i, ret;
    ret = sem_init(&mysem, 0, 1); /* initialize an unnamed
semaphore identified by global mysem for thread communication */
    if (-1 == ret) {
        perror("\nsem_init");
        return 1;
    }
```

Synchronization Examples

- **POSIX同步**
  - POSIX信号量
    - 算法 18-4: Syn-phread-sem-1.c (3)

```
if ((argc > 1) && (!strcmp(argv[1], "syn")))
    for (i = 0; i < thread_num; ++i)
        pthread_create(&tid[i], NULL, &test_func_syn, NULL);
else
    for (i = 0; i < thread_num; ++i)
        pthread_create(&tid[i], NULL, &test_func_asy, NULL);

for (i = 0; i < thread_num; ++i) {
    pthread_join(tid[i],NULL);
}

printf("result count = %d\n\n", count);

sem_destroy(&mysem);

return 0;
}
```

Synchronization Examples

- **POSIX同步**
  - POSIX信号量

```
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ gcc syn-pthread-sem-1.c -o syn-pthread-sem-1.o -lpthread
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ ./syn-pthread-sem-1.o syn
count = 1
count = 2
count = 3
count = 4
count = 5
count = 6
count = 7
count = 8
count = 9
count = 10
result count = 10

isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ ./syn-pthread-sem-1.o asy
count = 1
count = 4
count = 2
count = 5
count = 3
count = 6
count = 7
count = 8
count = 9
count = 10
result count = 10

isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$
```

Synchronization Examples

- **POSIX同步**
  - POSIX信号量
    - **算法** 18-5: Syn-phread-sem-2.c: POSIX命名信号量演示

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <string.h>
#include <fcntl.h>

sem_t *mysem;
static int count = 0;
void *test_func_syn(void *arg)
{
    for (int i = 0; i < 2; ++i) {
        sem_wait(mysem);
        count++;
        printf("count = %d\n", count);
        sem_post(mysem);
    }
    pthread_exit(NULL);
}
```

Synchronization Examples

- **POSIX同步**
  - POSIX信号量
    - **算法** 18-5: Syn-phread-sem-2.c (2)

```
void *test_func_asy(void *arg)
{
    for (int i = 0; i < 2; ++i) {
        count++;
        printf("count = %d\n", count);
    }
    pthread_exit(NULL);
}

int main(int argc, const char *argv[])
{
    int thread_num = 5;
    pthread_t tid[thread_num];
    int i, ret;
    mysem = sem_open("MYSEM", O_CREAT, 0666, 1); /* initialize a
named semaphore, "sem.MYSEM" is created in /dev/shm/ */
    if (SEM_FAILED == mysem) {
        perror("\nsem_open");
        return 1;
    }
```

Synchronization Examples

- **POSIX同步**
  - POSIX信号量
    - 算法 18-5: Syn-phread-sem-2.c (3)

```
if ((argc > 1) && (!strcmp(argv[1], "syn")))
    for (i = 0; i < thread_num; ++i)
        pthread_create(&tid[i], NULL, &test_func_syn, NULL);
else
    for (i = 0; i < thread_num; ++i)
        pthread_create(&tid[i], NULL, &test_func_asy, NULL);

for (i = 0; i < thread_num; ++i) {
    pthread_join(tid[i],NULL);
}

printf("result count = %d\n\n", count);
sem_close(mysem);
sem_unlink("MYSEM"); /* remove sem.MYSEM from /dev/shm/ when
its references is 0 */

return 0;
}
```

Synchronization Examples

- **POSIX同步**
  - POSIX信号量

```
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ gcc syn-pthread-sem-2.c -o syn-pthread-sem-2.o -lpthread
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ ./syn-pthread-sem-2.o syn
count = 1
count = 2
count = 3
count = 4
count = 5
count = 6
count = 7
count = 8
count = 9
count = 10
result count = 10

isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ ./syn-pthread-sem-2.o
count = 1
count = 3
count = 2
count = 4
count = 5
count = 8
count = 6
count = 9
count = 7
count = 10
result count = 10

isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$
```

Synchronization Examples

- **POSIX同步**
  - **算法** 18-6: syn-pc-con-5.h: **生产者消费者控制程序**

```
#define BASE_ADDR 10
/* first 10 units for control para, data starts from the unit indexed 10 */
/* circular queues shifted by (enqueue|dequeue) % buffer_size + BASE_ADDR */

struct ctln_pc_st {
    int BUFFER_SIZE;
    int MAX_ITEM_NUM; // number of items to be produced
    int THREAD_PRO; // number of producers
    int THREAD_CONS;  // number of consumers
    sem_t sem_mutex;
    sem_t stock;
    sem_t emptyslot;
    int item_num; // number of items having produced
    int enqueue, dequeue; // current positions of PRO and CONS in buffer
    int consume_num;  // number of items having consumed
    int END_FLAG; // producers met MAX_ITEM_NUM, finished their work
};

struct data_pc_st {
    int item_no; // the item_num when it is made
    int pro_no; // reserved
    long int pro_tid;
};
```

Synchronization Examples

- **POSIX同步**
  - **算法** 18-6: syn-pc-con-5.c (1)

```
/*  this version works well
    file list:syn-pc-con-5.h
              syn-pc-con-5.c
              syn-pc-producer-5.c
              syn-pc-consumer-5.c
    with process shared memory and semaphores
    BUFFER_SIZE, MAX_ITEM_NUM, THREAD_PRO and THREAD_CONS got from input */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/shm.h>
#include <semaphore.h>
#include <wait.h>
#include "syn-pc-con-5.h"

int shmid;
void *shm = NULL;
int detachshm(void);
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-6: syn-pc-con-5.c (2)

```
int main(int argc, char *argv[])
{
    pid_t childpid, pro_pid, cons_pid;
    struct stat statbuf;
    int buffer_size, max_item_num, thread_pro, thread_cons;

    if (argc < 2) {
        printf("\nshared file object undeclared!\nUsage: shmcon.o
/home/myshm\n");
        return EXIT_FAILURE;
    }
    if (stat(argv[1], &statbuf) == -1) {
        perror("\nshared file object stat error");
        return EXIT_FAILURE;
    }
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-6: syn-pc-con-5.c (3)

```
    while (1) {
        printf("Pls input the buffer size:(1-100, 0 quit) ");
        scanf("%d", &buffer_size);
        if (buffer_size <= 0) return 0;
        if (buffer_size > 100) continue;
        printf("Pls input the max number of items to be produced:(1-10000, 0
quit) ");
        scanf("%d", &max_item_num);
        if (max_item_num <= 0) return 0;
        if (max_item_num > 10000) continue;
        printf("Pls input the number of producers:(1-500, 0 quit) ");
        scanf("%d", &thread_pro);
        if (thread_pro <= 0) return 0;
        if (thread_pro < 0) continue;
        printf("Pls input the number of consumers:(1-500, 0 quit) ");
        scanf("%d", &thread_cons);
        if (thread_cons <= 0) return 0;
        if (thread_cons < 0) continue;
        break;
    }
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-6: syn-pc-con-5.c (4)

```
struct ctln_pc_st *ctln = NULL;  // shared control para, within BASE_ADDR
struct data_pc_st *data = NULL;  // shared data, beyond BASE_ADDR
key_t key;
int ret;

if ((key = ftok(argv[1], 0x28)) < 0) {
    perror("\nshmread key-gen failed");
    exit(EXIT_FAILURE);
}
  /* 'invalid argument': size exceeds the current shmmax, or the shmid
exists, its size is defined when it was firstly declared and can not be changed.
You have to alter a new key for a new shmid with larger size */
shmid = shmget((key_t)key, (buffer_size + BASE_ADDR)*sizeof(struct
data_pc_st), 0666 | IPC_CREAT);
if (shmid == -1) {
    perror("\nmain() shmget failed");
    exit(EXIT_FAILURE);
}

shm = shmat(shmid, 0, 0); /* attach shared memory to user space */
if (shm == (void *)-1) {
    perror("\nmain() shmat failed");
    exit(EXIT_FAILURE);
}
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-6: syn-pc-con-5.c (5)

```
  /* set the shared memory */
ctln = (struct ctln_pc_st *)shm;
data = (struct data_pc_st *)shm;

ctln->BUFFER_SIZE = buffer_size;
ctln->MAX_ITEM_NUM = max_item_num;
ctln->THREAD_PRO = thread_pro;
ctln->THREAD_CONS = thread_cons;
ctln->item_num = 0;
ctln->enqueue = 0;
ctln->dequeue = 0;
ctln->consume_num = 0;
ctln->END_FLAG = 0;
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-6: syn-pc-con-5.c (6)

```
    ret = sem_init(&ctln->sem_mutex, 1, 1); /* the second parameter of sem_init
must be set to non-zero for inter process sharing */
    if (-1 == ret) {
        perror("\nsem_init sem_mutex");
        return detachshm();
    }
    ret = sem_init(&ctln->stock, 1, 0);
    if (-1 == ret) {
        perror("\nsem_init stock");
        return detachshm();
    }
    ret = sem_init(&ctln->emptyslot, 1, ctln->BUFFER_SIZE);
    if (-1 == ret) {
        perror("\nsem_init emptyslot");
        return detachshm();
    }
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-6: syn-pc-con-5.c (7)

```
    printf("\nconsole pid = %d\n", getpid());

    char *argv1[3];
    char execname[] = "./";
    char shmidstring[10];
    sprintf(shmidstring, "%d", shmid);
    argv1[0] = execname;
    argv1[1] = shmidstring;
    argv1[2] = NULL;

    childpid = vfork(); /* vfork(), not fork() */
    if (childpid < 0) {
        perror("first fork error");
        return detachshm();
    }
    else if (childpid == 0) { /*  vfork return 0 in the child process */
          /* call the producer */
        pro_pid = getpid();
        printf("producer pid = %d, shmid = %s\n", pro_pid, argv1[1]);
        execv("./syn-pc-producer-5.o", argv1);
    }
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-6: syn-pc-con-5.c (8)

```
else { /* vfork another child */
    childpid = vfork();
    if (childpid < 0) {
        perror("second fork error");
        return detachshm();
    }
    else if (childpid == 0) { /*  vfork return 0 in the child process */
          /* call the consumer */
        cons_pid = getpid();
        printf("consumer pid = %d, shmid = %s\n", cons_pid, argv1[1]);
        execv("./syn-pc-consumer-5.o", argv1);
    }
}

if (waitpid(pro_pid, 0, 0) != pro_pid) /* block wait */
    perror("\npro_pid wait error");
else
    printf("waiting pro_pid %d success.\n", pro_pid);

if (waitpid(cons_pid, 0, 0) != cons_pid)
    perror("\ncons_pid wait error");
else
    printf("waiting cons_pid %d success.\n", cons_pid);
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-6: syn-pc-con-5.c (9)

```
    ret = sem_destroy(&ctln->sem_mutex);
    if (-1 == ret) perror("\nsem_mutex sem_destroy");
    ret = sem_destroy(&ctln->stock); /* sem_destroy() will not affect the
sem_wait() calling process */
    if (-1 == ret) perror("\nstock sem_destroy");
    ret = sem_destroy(&ctln->emptyslot);
    if (-1 == ret) perror("\nemptyslot sem_destroy");

    return detachshm();
}

int detachshm(void)
{
    if (shmdt(shm) == -1) {
        perror("\nsyn-pc-con shmdt() failed");
        exit(EXIT_FAILURE);
    }
    if (shmctl(shmid, IPC_RMID, 0) == -1) {
        perror("\nshmctl(IPC_RMID) failed");
        exit(EXIT_FAILURE);
    }
}
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-7: syn-pc-producer-5.c (1) 生产者程序

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/shm.h>
#include <semaphore.h>
#include <unistd.h>
#include <sys/syscall.h>
#include "syn-pc-con-5.h"

#define gettid() syscall(__NR_gettid)
void *producer(void *arg)
{
    void *shm = (void *)arg;
    struct ctln_pc_st *ctln = NULL;
    struct data_pc_st *data = NULL;
    ctln = (struct ctln_pc_st *)shm;
    data = (struct data_pc_st *)shm;
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-7: syn-pc-producer-5.c (2)

```
    while (ctln->item_num < ctln->MAX_ITEM_NUM) {
        sem_wait(&ctln->emptyslot);
        sem_wait(&ctln->sem_mutex);

        if (ctln->item_num < ctln->MAX_ITEM_NUM) {
            ctln->item_num++;
            ctln->enqueue = (ctln->enqueue + 1) % ctln->BUFFER_SIZE;
            (data + ctln->enqueue + BASE_ADDR)->item_no = ctln->item_num;
            (data + ctln->enqueue + BASE_ADDR)->pro_tid = gettid();
            printf("producer tid %ld prepared item no %d, now enqueue = %d\n",
(data + ctln->enqueue + BASE_ADDR)->pro_tid, (data + ctln->enqueue +
BASE_ADDR)->item_no, ctln->enqueue);
            if (ctln->item_num == ctln->MAX_ITEM_NUM)
                ctln->END_FLAG = 1;
            sem_post(&ctln->stock);
        }
        else {
            sem_post(&ctln->emptyslot);
        }
     sem_post(&ctln->sem_mutex);
    }
    pthread_exit(0);
}
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-7: syn-pc-producer-5.c (3)

```
int main(int argc, char *argv[])
{
    struct ctln_pc_st *ctln = NULL;  // shared control para, within BASE_ADDR
    struct data_pc_st *data = NULL;  // shared data, beyond BASE_ADDR

    int shmid;
    void *shm = NULL;
    shmid = strtol(argv[1], NULL, 10);
    shm = shmat(shmid, 0, 0);
    if (shm == (void *)-1) {
        perror("\nmain() shmat failed");
        exit(EXIT_FAILURE);
    }
    ctln = (struct ctln_pc_st *)shm;
    data = (struct data_pc_st *)shm;
    pthread_t tidp[ctln->THREAD_PRO];
    int i, ret;
    for (i = 0; i < ctln->THREAD_PRO; ++i) {
        ret = pthread_create(&tidp[i], NULL, &producer, shm);
        if (ret != 0) {
            perror("producer thread create error");
            break;
        }
    }
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-7: syn-pc-producer-5.c (4)

```
    for (i = 0; i < ctln->THREAD_PRO; ++i) {
        pthread_join(tidp[i], NULL);
    }

    for (i = 0; i < ctln->THREAD_CONS - 1; ++i) /* all producers stop working,
in case some consumer takes the last stock and no more than THREAD_CON-1
consumers stick in the sem_wait(&stock) */
        sem_post(&ctln->stock);

    if (shmdt(shm) == -1) {
        perror("\nshm-pc-producer shmdt() failed");
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

Synchronization Examples

- **POSIX同步**
  - **算法** 18-8: syn-pc-consumer-5.c (1) **消费者程序**

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/shm.h>
#include <semaphore.h>
#include <unistd.h>
#include <sys/syscall.h>
#include "syn-pc-con-5.h"

#define gettid() syscall(__NR_gettid)

void *consumer(void *arg)
{
    void *shm = (void *)arg;
    struct ctln_pc_st *ctln = NULL;
    struct data_pc_st *data = NULL;
    ctln = (struct ctln_pc_st *)shm;
    data = (struct data_pc_st *)shm;
```

Synchronization Examples

- **POSIX同步**
  - **算法** 18-8: syn-pc-consumer-5.c (2)

```
    while ((ctln->consume_num < ctln->item_num) || (ctln->END_FLAG == 0))  {
        sem_wait(&ctln->stock);  /* if stock is empty and all producers stop
working at this point, one or more consumers may wait forever */
        sem_wait(&ctln->sem_mutex);
        if (ctln->consume_num < ctln->item_num) {
            ctln->dequeue = (ctln->dequeue + 1) % ctln->BUFFER_SIZE;
            printf("                                consumer tid %ld taken item
no %d by pro %ld, now dequeue = %d\n", gettid(), (data + ctln->dequeue +
BASE_ADDR)->item_no, (data + ctln->dequeue + BASE_ADDR)->pro_tid, ctln-
>dequeue);
            ctln->consume_num++;
            sem_post(&ctln->emptyslot);
        }
        else {
            sem_post(&ctln->stock);
        }
        sem_post(&ctln->sem_mutex);
    }
    pthread_exit(0);
}
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-8: syn-pc-consumer-5.c (3)

```
int main(int argc, char *argv[])
{
    struct ctln_pc_st *ctln = NULL;  // shared control para, within BASE_ADDR
    struct data_pc_st *data = NULL;  // shared data, beyond BASE_ADDR

    int shmid;
    void *shm = NULL;
    shmid = strtol(argv[1], NULL, 10);
    shm = shmat(shmid, 0, 0);
    if (shm == (void *)-1) {
        perror("\nmain() shmat failed");
        exit(EXIT_FAILURE);
    }
    ctln = (struct ctln_pc_st *)shm;
    data = (struct data_pc_st *)shm;
    pthread_t tidc[ctln->THREAD_CONS];
    int i, ret;
    for (i = 0; i < ctln->THREAD_CONS; ++i) {
        ret = pthread_create(&tidc[i], NULL, &consumer, shm);
        if (ret != 0) {
            perror("consumer thread create error");
            break;
        }
    } /* Try this: shm and shared defined as global, used by all threads */
```

Synchronization Examples

- **POSIX同步**
  - 算法 18-8: syn-pc-consumer-5.c (4)

```
    for (i = 0; i < ctln->THREAD_CONS; ++i)
        pthread_join(tidc[i], NULL);

    if (shmdt(shm) == -1) {
        perror("\nshm-pc-producer shmdt() failed");
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

Synchronization Examples

- **POSIX同步**

```
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ gcc syn-pc-con-5.c -o syn-pc-con-5.o -lpthread
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ ./syn-pc-con-5.o /home/myshm
Pls input the buffer size:(1-100, 0 quit) 4
Pls input the max number of items to be produced:(1-10000, 0 quit) 8
Pls input the number of producers:(1-500, 0 quit) 2
Pls input the number of consumers:(1-500, 0 quit) 3

console pid = 52285
producer pid = 52286, shmid = 163882
consumer pid = 52287, shmid = 163882
producer tid 52288 prepared item no 1, now enqueue = 1
producer tid 52288 prepared item no 2, now enqueue = 2
producer tid 52288 prepared item no 3, now enqueue = 3
producer tid 52289 prepared item no 4, now enqueue = 0
                              consumer tid 52290 taken item no 1 by pro 52288, now dequeue = 1
                              consumer tid 52290 taken item no 2 by pro 52288, now dequeue = 2
producer tid 52288 prepared item no 5, now enqueue = 1
                              consumer tid 52292 taken item no 3 by pro 52288, now dequeue = 3
producer tid 52288 prepared item no 6, now enqueue = 2
                              consumer tid 52292 taken item no 4 by pro 52289, now dequeue = 0
                              consumer tid 52291 taken item no 5 by pro 52288, now dequeue = 1
producer tid 52289 prepared item no 7, now enqueue = 3
                              consumer tid 52291 taken item no 6 by pro 52288, now dequeue = 2
                              consumer tid 52290 taken item no 7 by pro 52289, now dequeue = 3
producer tid 52289 prepared item no 8, now enqueue = 0
                              consumer tid 52292 taken item no 8 by pro 52289, now dequeue = 0
waiting pro_pid 52286 success.
waiting cons_pid 52287 success.
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$
```

Synchronization Examples

- **POSIX同步**
  - POSIX条件变量
    - Pthreads中的条件变量的行为类似于管程上下文中使用的条件变量，提供了一种加锁机制来确保数据完整性。
    - Pthreads通常用于C程序中。由于C语言没有管程，互斥锁与条件变量关联以完成加锁。
    - Pthreads中的条件变量使用pthread_cond_t数据类型，并由pthread_cond_init()初始化。以下代码创建并初始化条件变量及其关联的互斥锁：

```
pthread_mutex_t mutex;
pthread_cond_t cond_var;
pthread_mutex_init(&mutex, NULL);
pthread_cond_init(&cond_var, NULL);
```

Synchronization Examples

- **POSIX同步**
  - POSIX条件变量
    - 例子：
      - 线程可以使用Pthread条件变量等待条件语句(a==b)变为true：

```
pthread_mutex_lock(&mutex);
while (a!=b)
    pthread_cond_wait(&cond_var, &mutex);
临界区
pthread_mutex_unlock(&mutex);
```

      - 在调用pthread_cond_wait()函数之前，必须锁定与con_var关联的互斥锁mutex，因为它用于保护条件语句中的数据不受可能的竞争条件的影响。
      - pthread_cond_wait()函数用于等待条件变量。
      - 获取锁后，当所需条件非真即(a!=b)时，线程将检查条件并将mutex和con_var作为参数调用pthread_cond_wait().

Synchronization Examples

- **POSIX同步**
  - POSIX条件变量
    - 例子：
      - 线程可以使用Pthread条件变量等待条件语句(a==b)变为true：

```
pthread_mutex_lock(&mutex);
while (a!=b)
    pthread_cond_wait(&cond_var, &mutex);
临界区
pthread_mutex_unlock(&mutex);
```

      - pthread_cond_wait()将调用线程放在CV队列的末尾，释放mutex以允许另一线程访问共享数据，并可能更新其值，以便条件语句(a==b)的计算结果为true。当调用线程被激活时，它将锁定mutex并再次检查该条件。
        - 为什么用while而不用if判断条件？因为当条件语句为true时，可能会调度CV队列中调用线程之前的另一个线程。

Synchronization Examples

- **POSIX同步**
  - POSIX条件变量
    - **例子：**
      - **线程可以调用**pthread_cond_signal()**函数，向一个等待条件变量的线程发送信号**

```
pthread_mutex_lock(&mutex);
if (a == b)
    pthread_cond_signal(&cond_var);
pthread_mutex_unlock(&mutex);
```

    - **需要注意的是：**
      - pthread_cond_signal()**不会释放互斥锁**mutex
      - pthread_mutex_unlock()**释放互斥锁。**
      - **释放互斥锁后，收到信号的线程将成为互斥锁的持有者，并从**pthread_cond_wait()**调用中返回，重获控制权。**

Synchronization Examples

- **POSIX同步**
  - POSIX条件变量
    - **算法** 18-9: pthread-cond-wait.c (1) **基于条件变量的计数器**

```
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  cond  = PTHREAD_COND_INITIALIZER;

int count = 0;

void *decrement(void *arg){
    for (int i = 0; i < 4; i++) {
        pthread_mutex_lock(&mutex);
        while (count == 0)
            pthread_cond_wait(&cond, &mutex);
        count--;
        printf("                    count = %d.\n", count);
     printf("                  Unlock decrement.\n");
     pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

减法线程等待计数非0的条件

Synchronization Examples

- **POSIX同步**
  - POSIX条件变量
    - 算法 18-9: pthread-cond-wait.c (2)

加法线程当计数非0时发送条件信号

```
void *increment(void *arg) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 10000; j++);
        pthread_mutex_lock(&mutex);
        count++;
        printf("count = %d.\n", count);
        if (count != 0) pthread_cond_signal(&cond);
        printf("Unlock increment.\n");
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t tid_in, tid_de;
    pthread_create(&tid_de, NULL, &decrement, NULL);
    pthread_create(&tid_in, NULL, &increment, NULL);
    pthread_join(tid_de, NULL); pthread_join(tid_in, NULL);
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond);
    return 0;
}
```

Synchronization Examples

- **POSIX同步**
  - POSIX条件变量
    - 算法 18-9: pthread-cond-wait.c (2)

加法线程当计数非0时发送条件信号

```
void *increment(void *arg) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 10000; j++);
```

```
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ gcc pthread-cond-wait.c -o pthread-cond-wait.o -lpthread
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ ./pthread-cond-wait.o
count = 1.
Unlock increment.
count = 2.
Unlock increment.
                    count = 1.
                    Unlock decrement.
                    count = 0.
                    Unlock decrement.
count = 1.
Unlock increment.
count = 2.
Unlock increment.
                    count = 1.
                    Unlock decrement.
                    count = 0.
                    Unlock decrement.
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$
```

```
        return 0;
}
```

Synchronization Examples

- **POSIX同步**
  - POSIX条件变量–演示程序
    - 编程练习
      - 使用pthread_mutex和pthread_cond解决生产者-消费者问题。

Synchronization Examples

- **Solaris同步**
  - 实现各种锁以支持多任务、多线程（包括实时线程）和多处理器。
  - 在保护数据不受短代码段影响时，使用自适应互斥锁以提高效率：
    - 初始化为标准信号量自旋锁
    - 如果锁被另一个CPU上运行的线程持有，则自旋
    - 如果锁被非运行状态线程持有，则阻塞并休眠以等待锁释放的信号
  - 使用条件变量。
  - 当较长的代码段需要访问数据时，使用读者写者锁。
  - 使用旋转栅门（Turnstile）对等待获取自适应互斥锁或读者写者锁的线程列表进行排序
    - 旋转栅门队列是基于每个持有锁的线程，而不是基于每个对象
  - 每个旋转栅门的优先级继承为正在运行的线程提供旋转栅门中线程的最高优先级，以防止优先级反转（一个低优先级的线程持有一个被高优先级线程所需要的共享资源）。

Synchronization Examples

- **Windows同步**
  - 使用中断屏蔽保护对单处理器系统上全局资源的访问。
  - 在多处理器系统上使用自旋锁：
    - 自旋锁线程永远不会被抢占。
  - 还提供了调度程序对象用户区，它可以充当互斥体、信号量、事件和计时器：
    - 事件
      - 事件的行为非常类似于条件变量。
    - 计时器在时间过期时通知一个或多个线程。
    - 调度程序对象可以是有信号状态（对象可用）或无信号状态（线程将阻塞）。