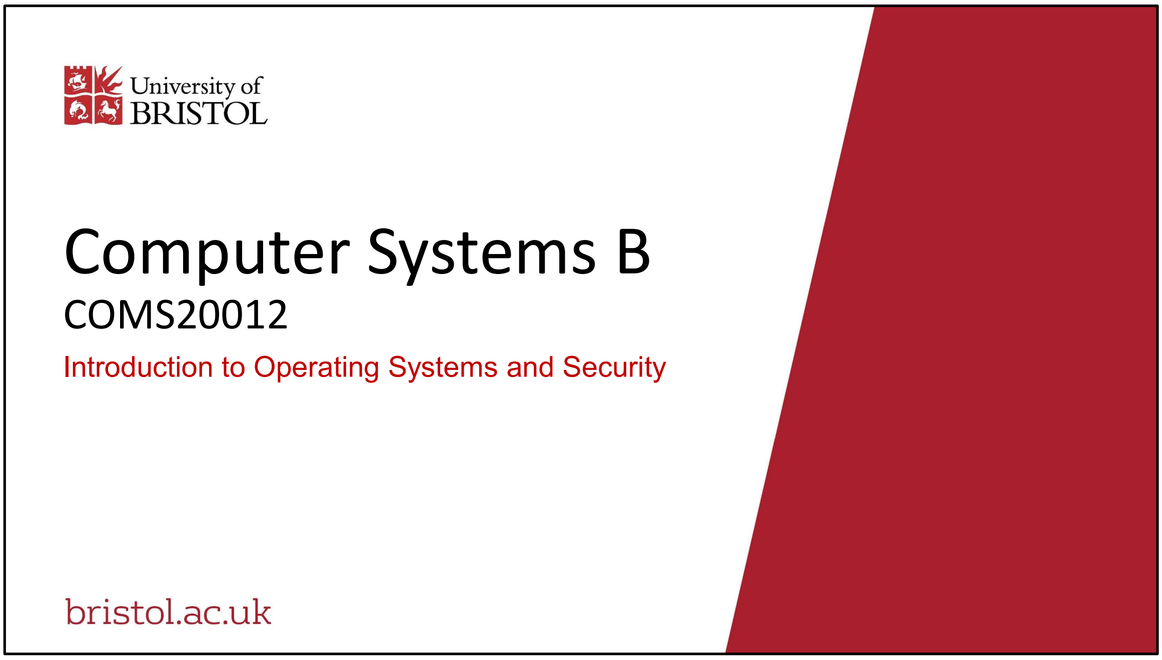


University of BRISTOL

Computer Systems B

COMS20012

Introduction to Operating Systems and Security

bristol.ac.uk

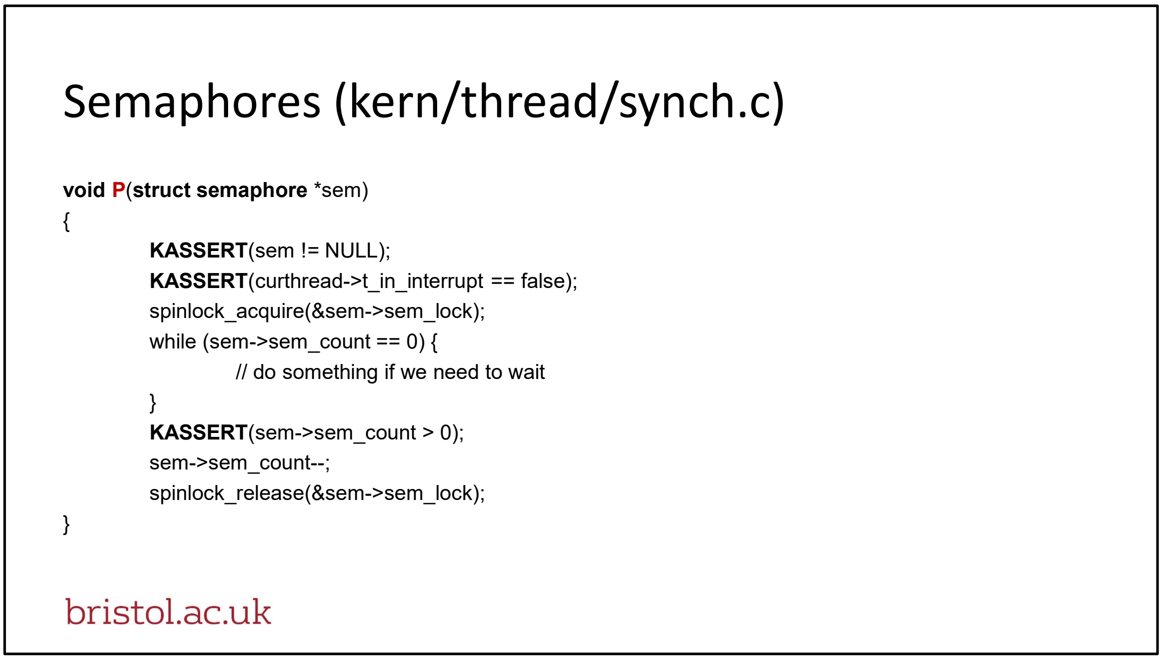
University of BRISTOL

Implementing semaphores

bristol.ac.uk

OS161 code

bristol.ac.uk



Semaphores (kern/thread/synch.c)

```
void P(struct semaphore *sem)
{
        KASSERT(sem != NULL);
        KASSERT(curthread->t_in_interrupt == false);
        spinlock_acquire(&sem->sem_lock);
        while (sem->sem_count == 0) {
                // do something if we need to wait
        }
        KASSERT(sem->sem_count > 0);
        sem->sem_count--;
        spinlock_release(&sem->sem_lock);
}
```

Semaphores (kern/thread/synch.c)

```
void P(struct semaphore *sem)
{
        KASSERT(sem != NULL);
        KASSERT(curthread->t_in_interrupt == false);
        spinlock_acquire(&sem->sem_lock);
        while (sem->sem_count == 0) {
                // do something if we need to wait
        }
        KASSERT(sem->sem_count > 0);
        sem->sem_count--;
        spinlock_release(&sem->sem_lock);
}
```

Conditions MUST be true

Conditions MUST be true

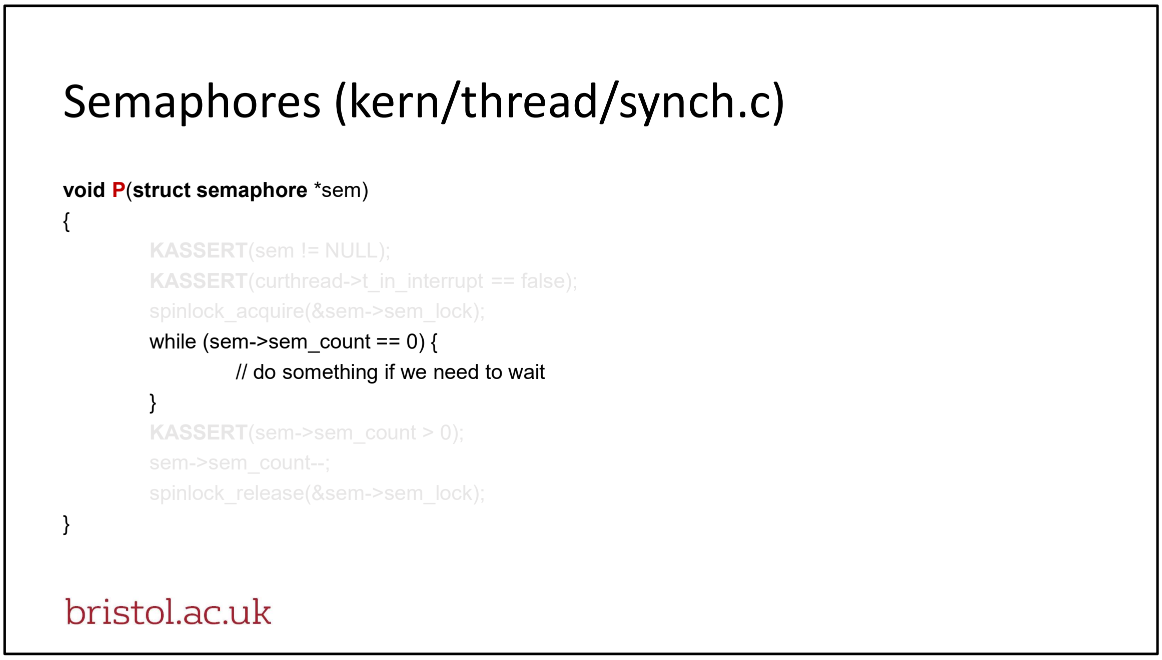
bristol.ac.uk

Semaphores (kern/thread/synch.c)

```
void P(struct semaphore *sem)
{
        KASSERT(sem != NULL);
        KASSERT(curthread->t_in_interrupt == false);
        spinlock_acquire(&sem->sem_lock);
        while (sem->sem_count == 0) {
                // do something if we need to wait
        }
        KASSERT(sem->sem_count > 0);
        sem->sem_count--;
        spinlock_release(&sem->sem_lock);
}
```

Check and Decrement
counter under lock

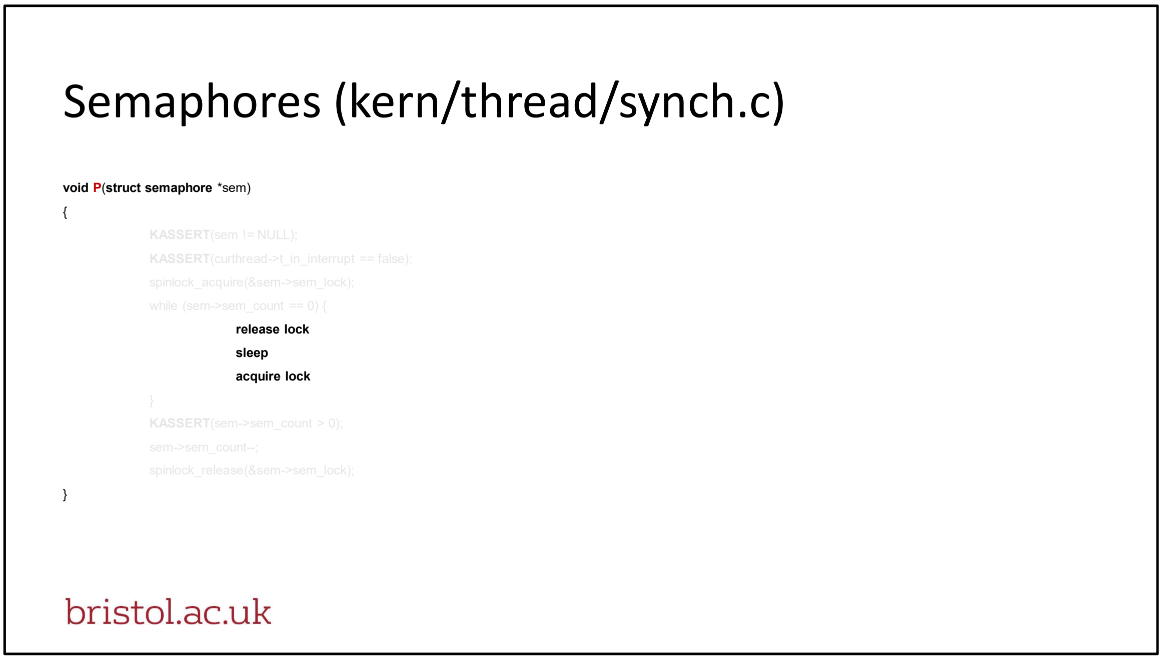
bristol.ac.uk



Semaphores (kern/thread/synch.c)

```
void P(struct semaphore *sem)
{
        KASSERT(sem != NULL);
        KASSERT(curthread->t_in_interrupt == false);
        spinlock_acquire(&sem->sem_lock);
        while (sem->sem_count == 0) {
                // do something if we need to wait
        }
        KASSERT(sem->sem_count > 0);
        sem->sem_count--;
        spinlock_release(&sem->sem_lock);
}
```

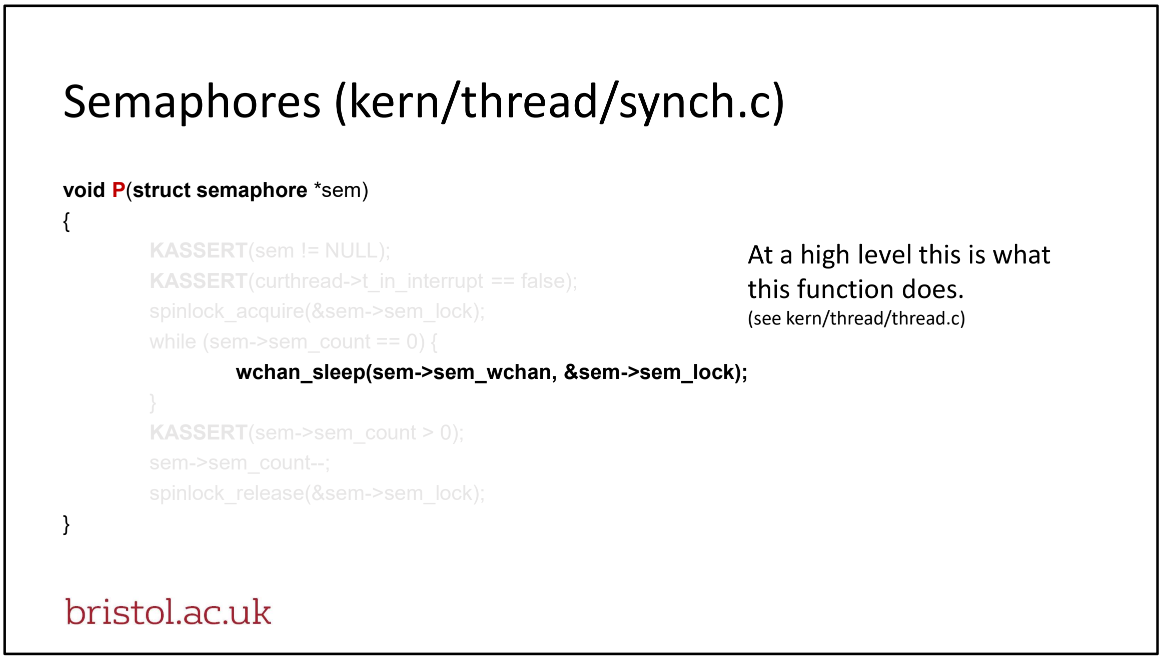
bristol.ac.uk



Semaphores (kern/thread/synch.c)

```
void P(struct semaphore *sem)
{
        KASSERT(sem != NULL);
        KASSERT(curthread->t_in_interrupt == false);
        spinlock_acquire(&sem->sem_lock);
        while (sem->sem_count == 0) {
                release lock
                sleep
                acquire lock
        }
        KASSERT(sem->sem_count > 0);
        sem->sem_count--;
        spinlock_release(&sem->sem_lock);
}
```

bristol.ac.uk



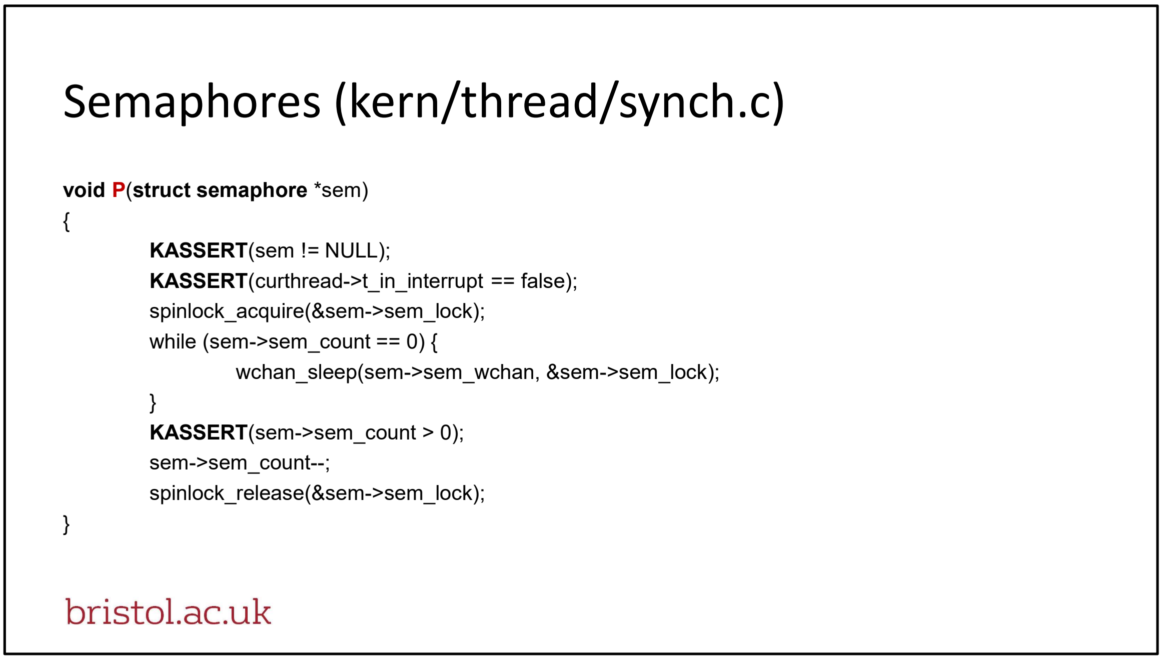
Semaphores (kern/thread/synch.c)

```
void P(struct semaphore *sem)
{
        KASSERT(sem != NULL);
        KASSERT(curthread->t_in_interrupt == false);
        spinlock_acquire(&sem->sem_lock);
        while (sem->sem_count == 0) {
                wchan_sleep(sem->sem_wchan, &sem->sem_lock);
        }
        KASSERT(sem->sem_count > 0);
        sem->sem_count--;
        spinlock_release(&sem->sem_lock);
}
```

At a high level this is what this function does.

(see kern/thread/thread.c)

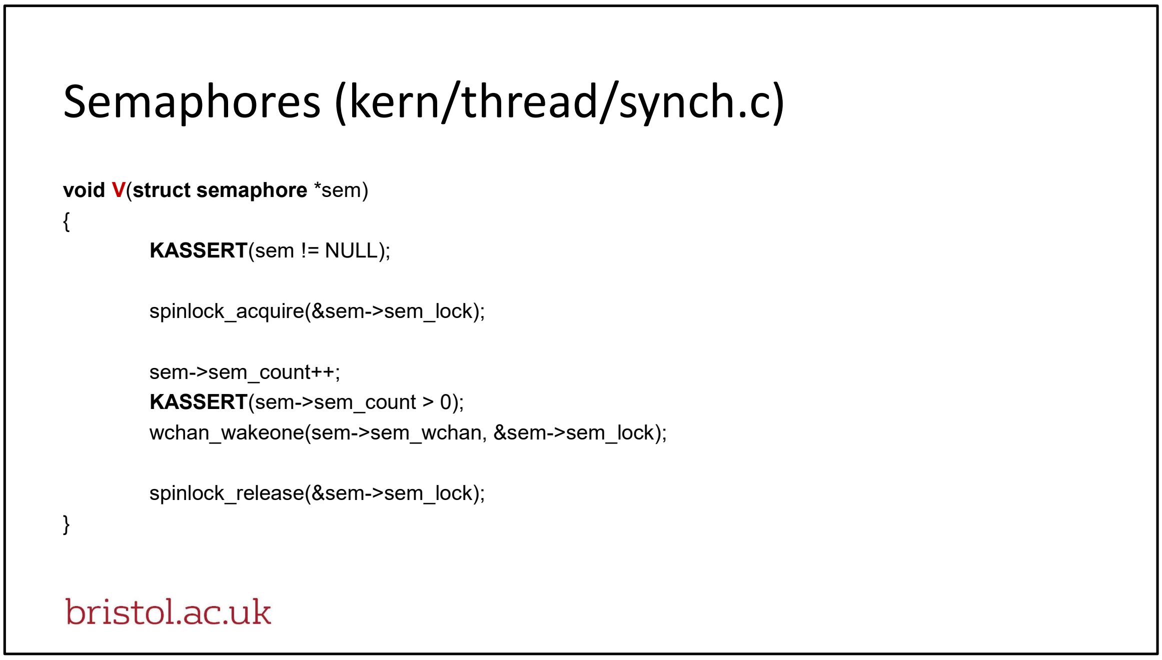
bristol.ac.uk



Semaphores (kern/thread/synch.c)

```
void P(struct semaphore *sem)
{
        KASSERT(sem != NULL);
        KASSERT(curthread->t_in_interrupt == false);
        spinlock_acquire(&sem->sem_lock);
        while (sem->sem_count == 0) {
                wchan_sleep(sem->sem_wchan, &sem->sem_lock);
        }
        KASSERT(sem->sem_count > 0);
        sem->sem_count--;
        spinlock_release(&sem->sem_lock);
}
```

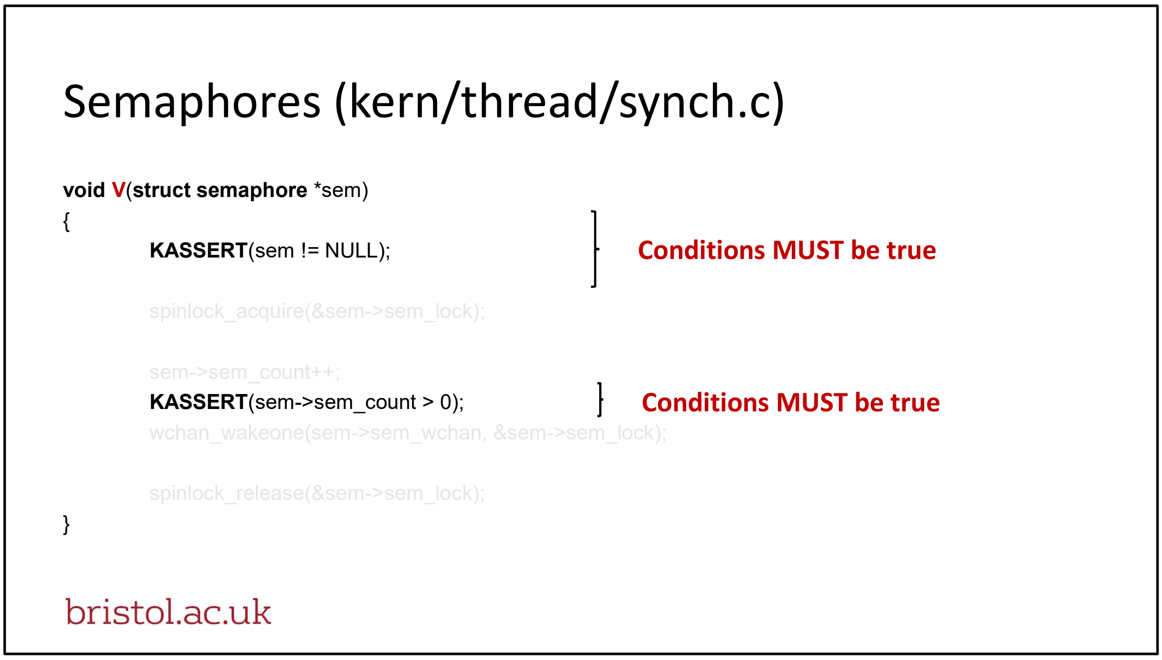
bristol.ac.uk



Semaphores (kern/thread/synch.c)

```
void V(struct semaphore *sem)
{
        KASSERT(sem != NULL);

        spinlock_acquire(&sem->sem_lock);

        sem->sem_count++;
        KASSERT(sem->sem_count > 0);
        wchan_wakeone(sem->sem_wchan, &sem->sem_lock);

        spinlock_release(&sem->sem_lock);
}
```

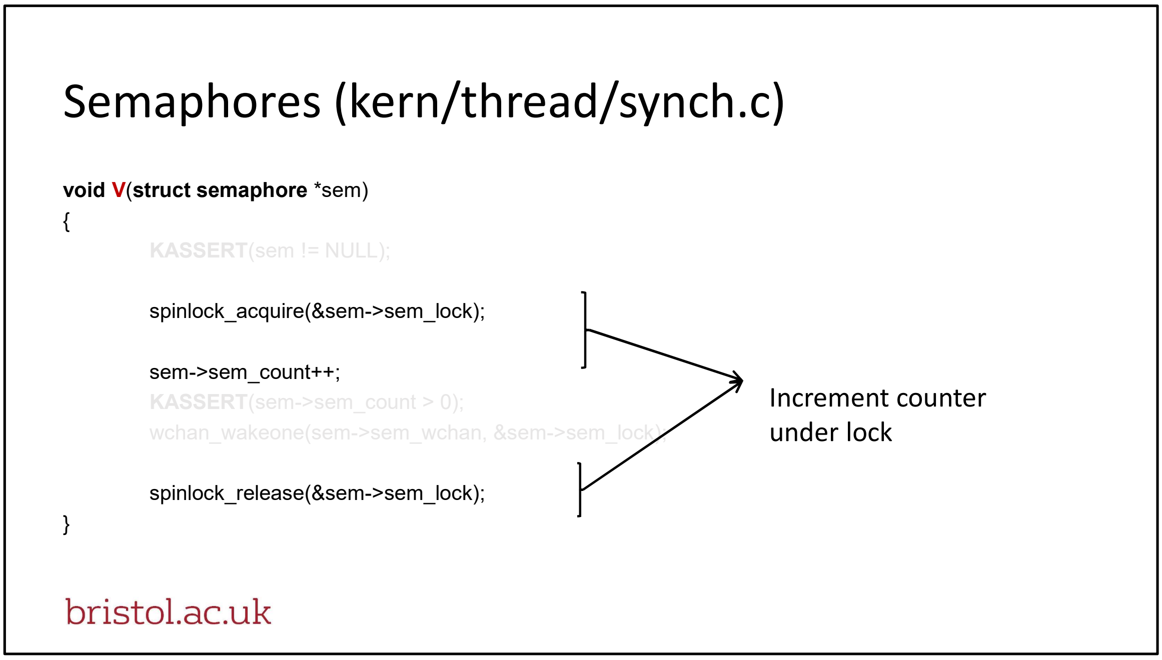
bristol.ac.uk



Semaphores (kern/thread/synch.c)

```
void V(struct semaphore *sem)
{
        KASSERT(sem != NULL);                              }  Conditions MUST be true

        spinlock_acquire(&sem->sem_lock);

        sem->sem_count++;
        KASSERT(sem->sem_count > 0);                       }  Conditions MUST be true
        wchan_wakeone(sem->sem_wchan, &sem->sem_lock);

        spinlock_release(&sem->sem_lock);
}
```

bristol.ac.uk

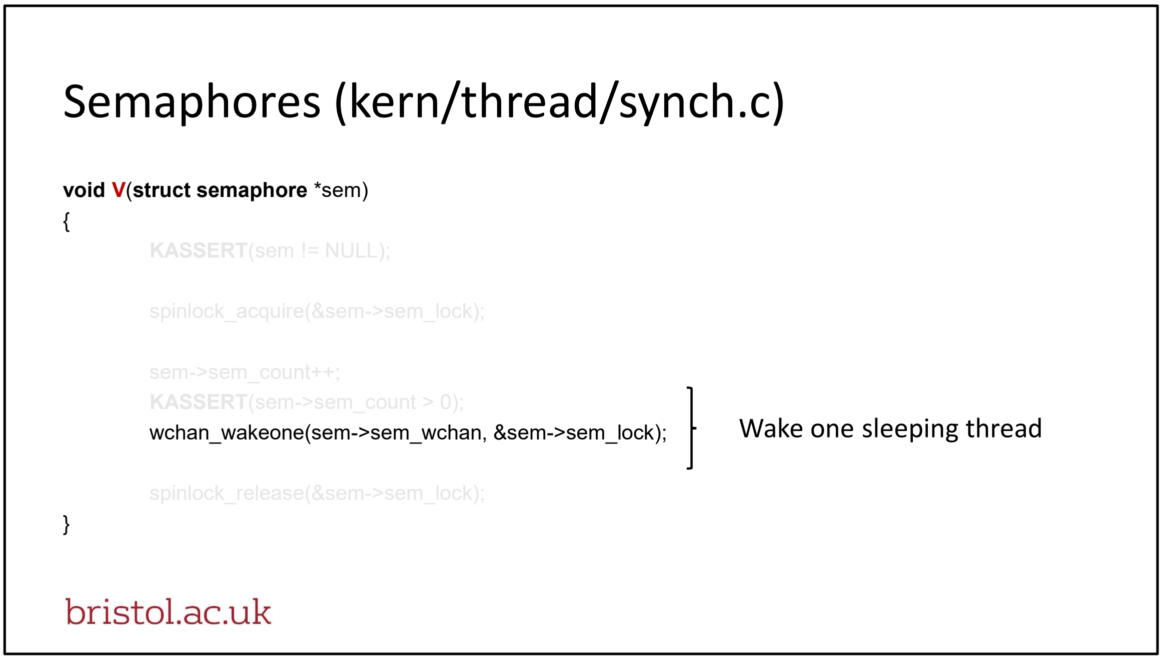


Semaphores (kern/thread/synch.c)

```
void V(struct semaphore *sem)
{
        KASSERT(sem != NULL);

        spinlock_acquire(&sem->sem_lock);

        sem->sem_count++;
        KASSERT(sem->sem_count > 0);
        wchan_wakeone(sem->sem_wchan, &sem->sem_lock);

        spinlock_release(&sem->sem_lock);
}
```

Increment counter under lock

bristol.ac.uk

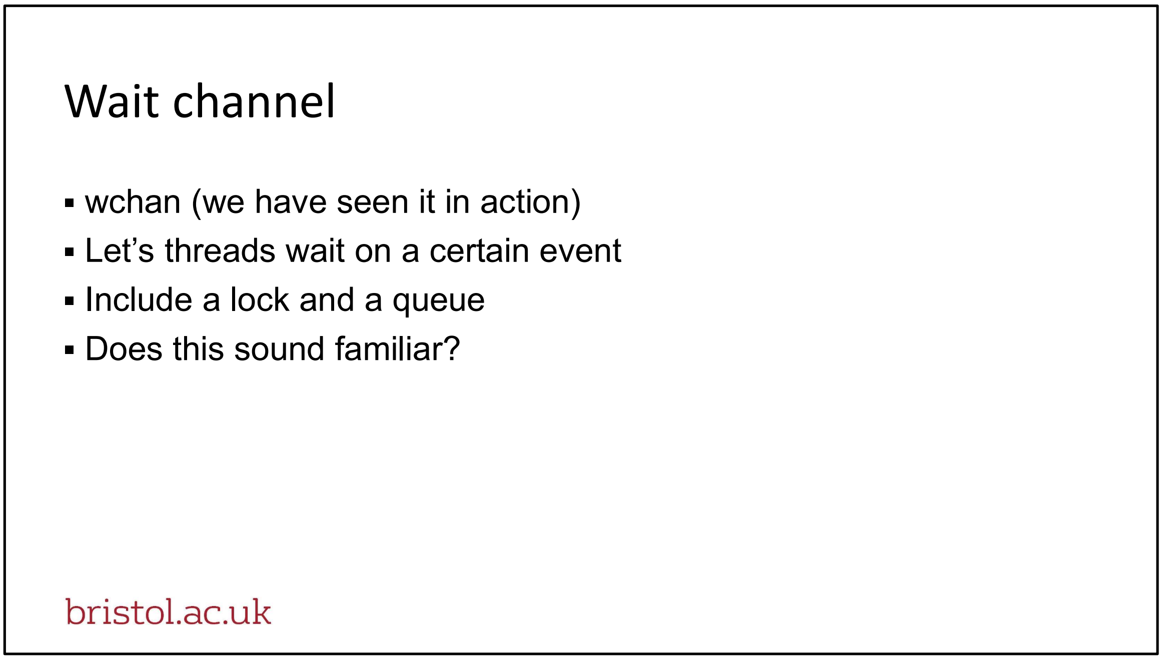


Semaphores (kern/thread/synch.c)

```
void V(struct semaphore *sem)
{
        KASSERT(sem != NULL);

        spinlock_acquire(&sem->sem_lock);

        sem->sem_count++;
        KASSERT(sem->sem_count > 0);
        wchan_wakeone(sem->sem_wchan, &sem->sem_lock);

        spinlock_release(&sem->sem_lock);
}
```

Wake one sleeping thread

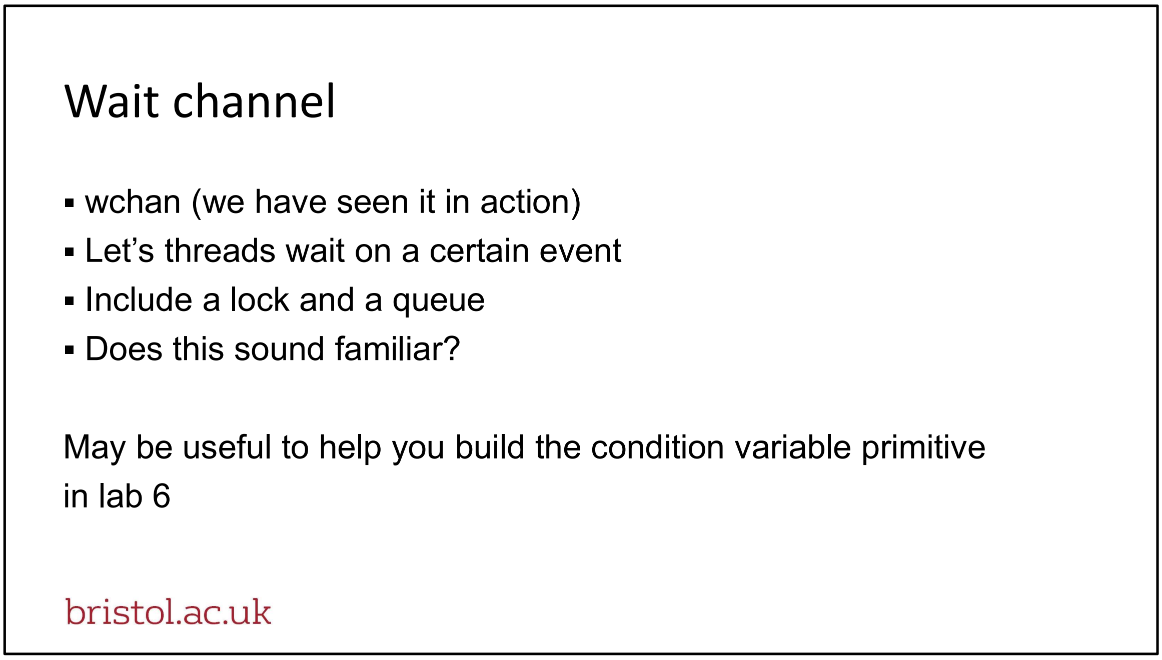
bristol.ac.uk



Wait channel

- wchan (we have seen it in action)
- Let’s threads wait on a certain event
- Include a lock and a queue
- Does this sound familiar?

bristol.ac.uk

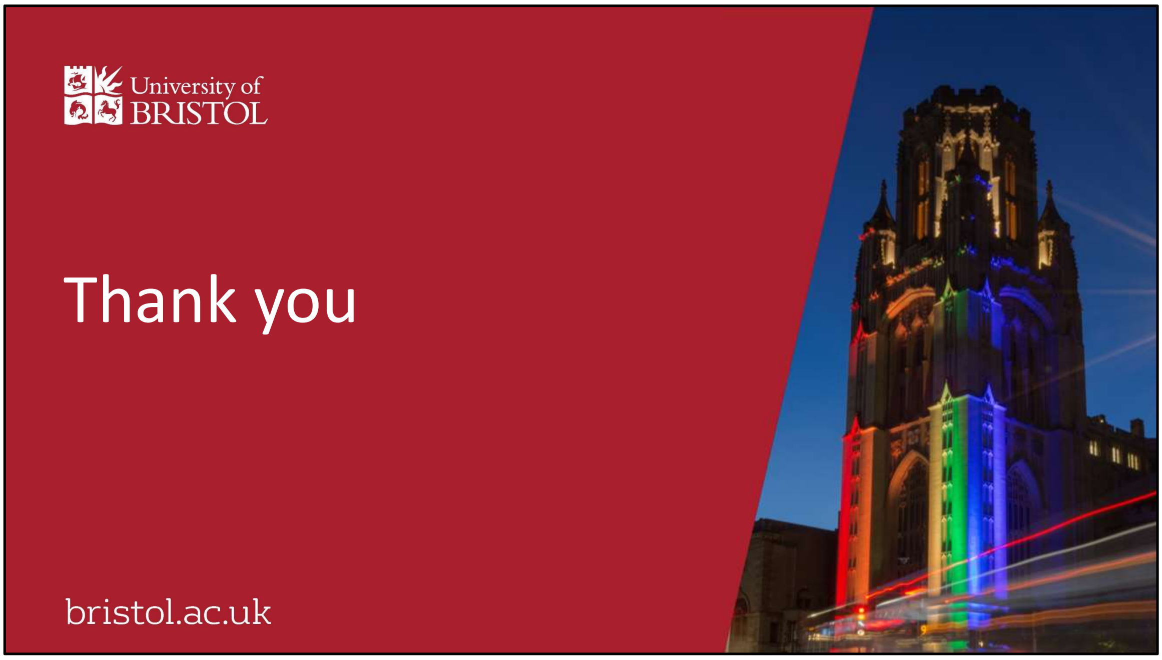


Wait channel

- wchan (we have seen it in action)
- Let’s threads wait on a certain event
- Include a lock and a queue
- Does this sound familiar?

May be useful to help you build the condition variable primitive in lab 6

bristol.ac.uk



University of BRISTOL

Thank you

bristol.ac.uk