[537] Semaphores

Chapter 31

Tyler Harter

10/20/14

Producer/Consumer Problem

Producers generate data (like pipe writers).

Consumers grab data and process it (like pipe readers).

Producer/consumer problems are frequent in systems.

Producer/Consumer Problem

Producers generate data (like pipe writers).

Consumers grab data and process it (like pipe readers).

Producer/consumer problems are frequent in systems.
- examples?
- what primitives did we use?

Condition Variables

wait(cond_t *cv, mutex_t *lock)
- assumes the lock is held when wait() is called
- puts caller to sleep + releases the lock (atomically)
- when awoken, reacquires lock before returning

signal(cond_t *cv)
- wake a single waiting thread (if >= 1 thread is waiting)
- if there is no waiting thread, just return, doing nothing

broadcast(cond_t *cv)
- wake **all** waiting threads (if >= 1 thread is waiting)
- if there are no waiting thread, just return, doing nothing

Example: Bounded Buffer

```
void *producer(void *arg) {
    for (int i=0; i<loops; i++) {
        Mutex_lock(&m);
        while(numfull == max)
            Cond_wait(&empty, &m);
        do_fill(i);
        Cond_signal(&fill);
        Mutex_unlock(&m);
    }
}
```

```
void *consumer(void *arg) {
    while(1) {
        Mutex_lock(&m);
        while(numfull == 0)
            Cond_wait(&fill, &m);
        int tmp = do_get();
        Cond_signal(&empty);
        Mutex_unlock(&m);
        printf(“%d\n”, tmp);
    }
}
```

Example: Bounded Buffer

```
void *producer(void *arg) {
    for (int i=0; i<loops; i++) {
        Mutex_lock(&m);
        while(numfull == max)
            Cond_wait(&empty, &m);
        do_fill(i);
        Cond_signal(&fill);
        Mutex_unlock(&m);
    }
}
```

```
void *consumer(void *arg) {
    while(1) {
        Mutex_lock(&m);
        while(numfull == 0)
            Cond_wait(&fill, &m);
        int tmp = do_get();
        Cond_signal(&empty);
        Mutex_unlock(&m);
        printf(“%d\n”, tmp);
    }
}
```

Discuss

Can we do producer/consumer with only locks (no CVs)?

Discuss

Can we do producer/consumer with only locks (no CVs)?

Do you like CVs?

Discuss

Can we do producer/consumer with only locks (no CVs)?

Do you like CVs? No!

Discuss

Can we do producer/consumer with only locks (no CVs)?

Do you like CVs? No!

Why are CVs hard to use?

Discuss

Can we do producer/consumer with only locks (no CVs)?

Do you like CVs? No!

Why are CVs hard to use?

What rules of thumb should we follow with CVs?

CV rules of thumb

Keep state in addition to CV's

Always do wait/signal with lock held

Whenever you acquire a lock, recheck state

Design Tip

If it’s always recommended to
use an abstraction the same way…

Design Tip

If it's always recommended to
use an abstraction the same way…

…build a better abstraction
over your first abstraction.

More Concurrency Abstractions

Linux Workqueues: list of function ptr's to call later.

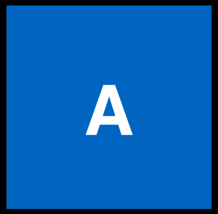
Semaphores: today's topic.

Condition Variable

Queue:

Condition Variable

Queue:



wait()

Condition Variable

Queue:

A

Condition Variable

Queue:

wait()

Condition Variable

Queue:

Condition Variable

Queue:

B

signal()

Condition Variable

Queue:

B

Condition Variable

Queue:

signal()

Condition Variable

Queue:

Condition Variable

Queue:

signal()

Condition Variable

Queue:

nothing to do!

signal()

Condition Variable

Queue:

Condition Variable

Queue:

wait()

Condition Variable

Queue:

C

Condition Variable

Queue:

C

If we weren’t careful, C may sleep forever.

Semaphore

Thread Queue: Signal Queue:

Semaphore

Thread Queue: Signal Queue:

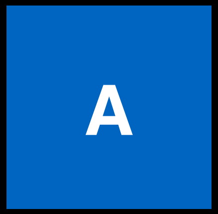
A

wait()

Semaphore

Thread Queue:

Signal Queue:



Semaphore

Thread Queue: Signal Queue:

signal()

Semaphore

Thread Queue:

Signal Queue:

Semaphore

Thread Queue:

Signal Queue:

signal

signal()

Semaphore

Thread Queue: Signal Queue:

signal

Semaphore

Thread Queue: Signal Queue:

A **signal**

wait()

Semaphore

Thread Queue:

Signal Queue:

wait()

Semaphore

Thread Queue: Signal Queue:

signal was not lost do to some race condition!

wait()

Semaphore

Thread Queue:

Signal Queue:

Actual Implementation

Use counter instead of Signal Queue
 - all signals are the same

If the counter is positive, don't bother to queue a thread upon wait().

Actual Implementation

Use counter instead of Signal Queue
 - all signals are the same

If the counter is positive, don't bother to queue a thread upon wait().

CV's don't keep extra state, so CV users must.
Semaphores keep extra state, so users sometimes don't.

Actual Definition (see handout)

```
sem_init(sem_t *s, int initval) {
   s->value = initval
}

sem_wait(sem_t *s) {
   s->value -= 1
   wait if s->value < 0
}

sem_post(sem_t *s) {
   s->value += 1
   wake one waiting thread (if there are any)
}
```

Actual Definition (see handout)

```
sem_init(sem_t *s, int initval) {
   s->value = initval
}

sem_wait(sem_t *s) {
   s->value -= 1
   wait if s->value < 0
}

sem_post(sem_t *s) {
   s->value += 1
   wake one waiting thread (if there are any)
}
```

wait and post are atomic

Actual Definition (see handout)

```
sem_init(sem_t *s, int initval) {
   s->value = initval
}

sem_wait(sem_t *s) {
   s->value -= 1
   wait if s->value < 0
}

sem_post(sem_t *s) {
   s->value += 1
   wake one waiting thread (if there are any)
}
```

value = 4: 4 waiting signals
value = -3: 3 waiting threads

Join example

Join is simpler with semaphores than CV's.

Join w/ CV

```
int done = 0;
mutex_t m = MUTEX_INIT;
cond_t c = COND_INIT;
void *child(void *arg) {
    printf("child\n");
    Mutex_lock(&m);
    done = 1;
    cond_signal(&c);
    Mutex_unlock(&m);
}

int main(int argc, char *argv[]) {
    pthread_t c;
    printf("parent: begin\n");
    Pthread_create(c, NULL, child, NULL);
    Mutex_lock(&m);
    while(done == 0)
        Cond_wait(&c, &m);
    Mutex_unlock(&m);
    printf("parent: end\n");
}
```

Join w/ Semaphore

```
sem_t s;
void *child(void *arg) {
    printf("child\n");
    sem_post(&s);
}

int main(int argc, char *argv[]) {
    sem_init(&s, ?);
    pthread_t c;
    printf("parent: begin\n");
    Pthread_create(c, NULL, child, NULL);
    sem_wait(&s);
    printf("parent: end\n");
}
```

Join w/ Semaphore

```
sem_t s;
void *child(void *arg) {
   printf("child\n");
   sem_post(&s);
}

int main(int argc, char *argv[]) {
   sem_init(&s, ?);
   pthread_t c;
   printf("parent: begin\n");
   Pthread_create(c, NULL, child, NULL);
   sem_wait(&s);
   printf("parent: end\n");
}
```

Join w/ Semaphore

```
sem_t s;
void *child(void *arg) {
   printf("child\n");
   sem_post(&s);
}

int main(int argc, char *argv[]) {
   sem_init(&s, ?);
   pthread_t c;
   printf("parent: begin\n");
   Pthread_create(c, NULL, child, NULL);
   sem_wait(&s);
   printf("parent: end\n");
}
```

What is this int?

Join w/ Semaphore

```
sem_t s;
void *child(void *arg) {
   printf("child\n");
   sem_post(&s);
}

int main(int argc, char *argv[]) {
   sem_init(&s, ?);
   pthread_t c;
   printf("parent: begin\n");
   Pthread_create(c, NULL, child, NULL);
   sem_wait(&s);
   printf("parent: end\n");
}
```

Join w/ Semaphore

```
sem_t s;
void *child(void *arg) {
   printf("child\n");
   sem_post(&s);
}

int main(int argc, char *argv[]) {
   sem_init(&s, 0);
   pthread_t c;
   printf("parent: begin\n");
   Pthread_create(c, NULL, child, NULL);
   sem_wait(&s);
   printf("parent: end\n");
}
```

Join w/ Semaphore

```
sem_t s;
void *child(void *arg) {
   printf("child\n");
   sem_post(&s);
}

int main(int argc, char *argv[]) {
   sem_init(&s, 0);
   pthread_t c;
   printf("parent: begin\n");
   Pthread_create(c, NULL, child, NULL);
   sem_wait(&s);
   printf("parent: end\n");
}
```

Run it!
(sem-join.c)

Worksheet

Problem 1: building locks with semaphores

Problem 2: building semaphores with locks and CV's

Equivalence Claim

Semaphores are equally powerful to Locks+CVs.
 - what does this mean?

Equivalence Claim

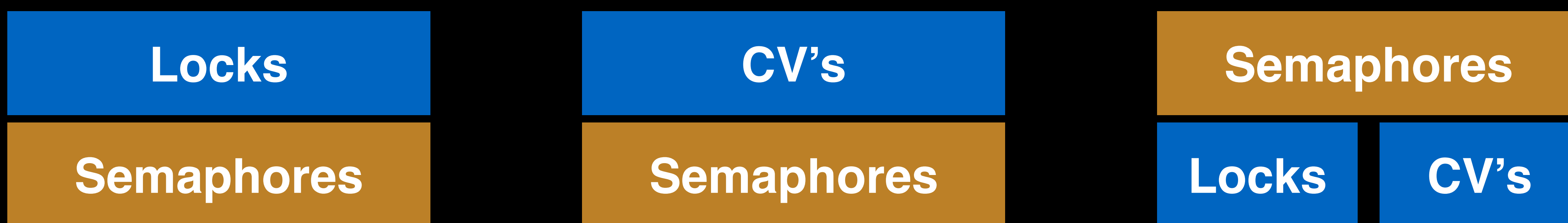
Semaphores are equally powerful to Locks+CVs.
 - what does this mean?

Either may be more convenient, but that's not relevant.

Equivalence means we can build each over the other.

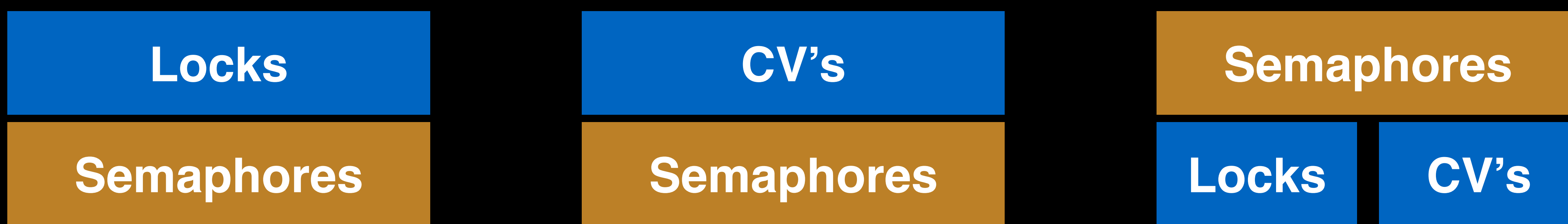
Proof Steps

Want to show we can do these three things:



Proof Steps

Want to show we can do these three things:

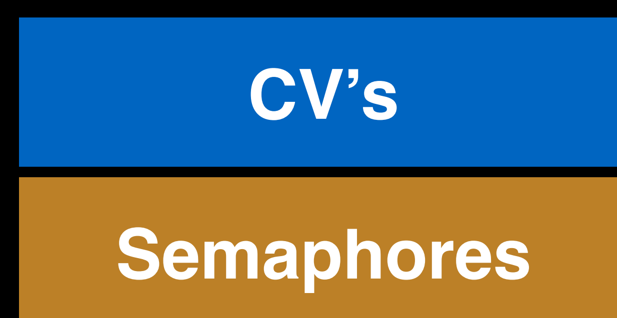


done!
(problem 1)

done!
(problem 2)

Building CV’s over Semaphores

Possible, but really hard to do right.

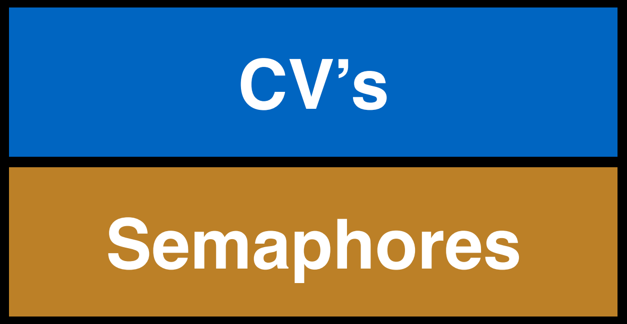


Building CV's over Semaphores

Possible, but really hard to do right.

Read about Microsoft Research's attempts:
- http://research.microsoft.com/pubs/64242/ImplementingCVs.pdf

We won't go beyond our simple join example.



Bounded-Buffer w/ Semaphores

Write code.

R/W Lock w/ Semaphores

Worksheet, Problem 3.

Summary

Locks+CVs are good primitives, but not always convenient.

Possible to build other abstractions such as semaphores.

Advice: if you always use an abstraction the same way,
build another abstraction over the first!