CS 240
Computer Systems

#10: Deadlock and Producer-Consumer

Feb. 17, 2022 · Wade Fagen-Ulmschneider

Solving Deadlock

On Tuesday, we explored the four necessary conditions for deadlock. In the context of the dining philosophers problem, how do we remove each of the four?

1. Mutual Exclusion

2. Circular Wait

3. Hold and Wait

4. No Preemption

Deadlock Solution Considerations

1. Fairness:

2. Livelock:

10/producer-consumer-2.c

```c
#define THINGS_MAX 10
#define THREAD_CT 5

int things[THINGS_MAX];
int things_ct = 0;



```

```c
void *producer(void *vptr) {
 while (1) {

  assert(things_ct <
                THINGS_MAX);

  // Produce a thing:
  things[things_ct] =
              rand() % 100;
  printf("Produced [%d]: %d
     -> ", things_ct,
           things[things_ct]);
  things_ct++;
  print_things_as_list();

 }
}
```

```c
void *consumer(void *vptr) {
 while (1) {

  assert(things_ct > 0);

  // Consume a thing:
  things_ct--;
  int value =
           things[things_ct];
  printf("Consumed [%d]: %d
           <- ", things_ct,
                  value);
  print_things_as_list();

 }
}
```

```c
int main() {
  int i;

  // Create `thread_ct` threads of each producer and consumer:
  pthread_t tid_consumer[THREAD_CT];
  pthread_t tid_producer[THREAD_CT];
  for (i = 0; i < THREAD_CT; i++) {
    pthread_create(&tid_consumer[i], NULL, producer, NULL);
    pthread_create(&tid_producer[i], NULL, consumer, NULL);
  }

  // Join threads:
  for (i = 0; i < THREAD_CT; i++) {
    pthread_join(tid_consumer[i], NULL);
    pthread_join(tid_producer[i], NULL);
  }
}
```