CS 107 Lecture 16

Semaphore : "sync counter var >= 0" (1)

```
void SellTickets (int agent, int *numTicketsp, Semaphore lock)
{
    while (true){
        Semaphore Wait (lock);
        if (*numTicketsp == 0){Break;}
        (*numTicketsp)--;
        Semaphore Signal (lock);
        print ( ... );
    }
    Semaphore Signal (lock);
}
```

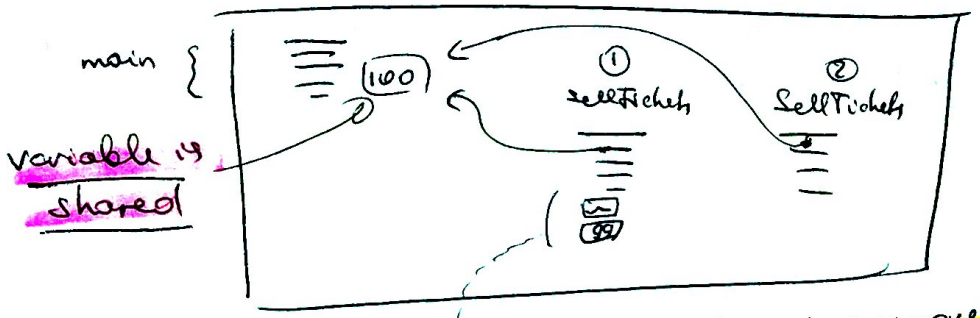
Binary lock pattern

[redacted]

critical region

make it as small as possible; only one thread in the region at any time

The problem is at any stage of decrementing, not only at *numTickets == 1



Let's let

① be interrupted prior to RET

R1 [redacted]

R2 100 → 99

when a thread is removed from a [redacted] processor, the registers values are copied to the bottom of activation record

already commited to decrementing to 99, when the thread is put on processor again it will set global variable to 99, but other threads decremented it further already.

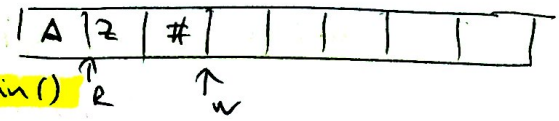
need critical region

lock

better declare [redacted] initialize Semaphore variable in main, not globally, as well as the shared resource variable, if practical

```
char buffer[8];
```

ex: rendering [redacted] in the browser cannot be ahead of data from server




```
Semaphore emptyBuffers [redacted] (8);
Semaphore fullBuffers [redacted] (0);
```

↖ global variables but could be inside main

```
int main()
{
  ITP(false);
  ThreadNew("Writer", Writer, 0);
  ThreadNew("Reader", Reader, 0);
  RunAllThreads();
}
```

↙ producer ↙ consumer

Rendezvous pattern

here binary rendezvous: one thread to one thread

```
void Writer()  // should be ahead of reader
               // and should not destroy data that has not been read
{
  for (int i=0; i<40; i++) {  // cycling 5 times through buffer
    char c = PrepareRandomChar();  // assume thread-safe
    SemaphoreWait(emptyBuffers);  // blocked if written one cycle ahead of reader
    buffer[i%8] = c;
    SemaphoreSignal(fullBuffers);
  }
}

void Reader()
{
  for (int i=0; i<40; i++) {
    SemaphoreWait(fullBuffers);  // blocked if read upto portion of writer
    char c = buffer[i%8];
    SemaphoreSignal(emptyBuffers);
    ProcessChar(c);
  }
}
```

my comment

counters

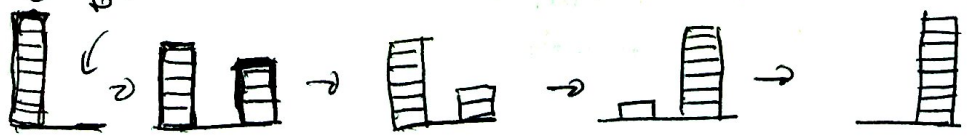
start

to write

to read

buffer is shared

e.g. if more than 1 writer, then would also need a binary lock



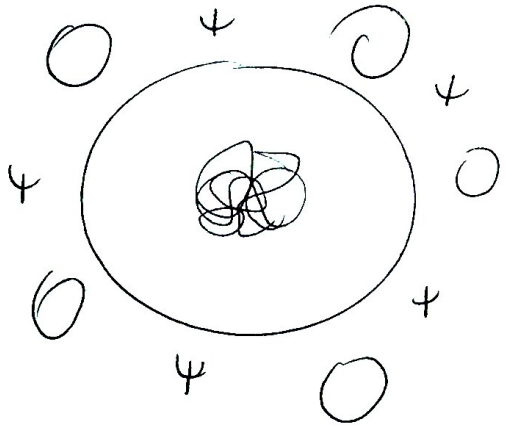
deadlock:

```
Semaphore emptyBuffers(0);
Semaphore fullBuffers(0);
```

CS107 Lecture 16

Dining Philosophers Problem

— 5 philosophers
— 5 forks
— a philosopher can eat only with 2 forks



```
Semaphore forks[] = {1,1,1,1,1};
Semaphore numAllowedToEat(4);
heuristic used to solve deadlock
void Philosopher(int id)
{
  for (int i=0; i<3; i++){
    Think();
    Wait(forks[id])
    Wait(forks[(id+1)%5]);
    Eat();
    [redacted]
    SemaphoreSignal(forks[id]);
    SemaphoreSignal(forks[(id+1)%5]);
  }
}
```

deadlock occurs when each philosopher of 5 grabs one fork to the e.g. right

heuristic:
if only 4 Philosophers are allowed to [redacted] eat at any time, then at least one will be able to grab 2 forks

⇓

at least one thread is guaranteed to run at any given time

⇓

no deadlock