Barriers

• Use of a barrier
  – processes approaching a barrier
  – all processes but one blocked at barrier
  – last process arrives, all are let through
Dining Philosophers (1)

- Philosophers eat/think
- Eating needs 2 forks
- Pick one fork at a time
- How to prevent deadlock

Posix Library Implementation
Based F. Mueller’s Paper

Language Application

Language Interface

C Language Application

Posix thread library

Unix libraries

User Level

Kernel Level

Unix Kernel

To answer Hanumanth’s question.
Dining Philosophers Example

```c
monitor dp
{
    enum {thinking, hungry, eating} state[5];
    condition self[5];
    void pickup(int i) // following slides
    void putdown(int i) // following slides
    void test(int i) // following slides
    void init() {
        for (int i = 0; i < 5; i++)
            state[i] = thinking;
    }
}
```

Dining Philosophers

```c
void pickup(int i) {
    state[i] = hungry;
    test[i];
    if (state[i] != eating)
        self[i].wait();
}

void putdown(int i) {
    state[i] = thinking;
    // test left and right neighbors
    test((i+4) % 5);
    test((i+1) % 5);
}
```
Dining Philosophers

```c
void test(int i) {
    if ( (state[(i + 4) % 5] != eating) &&
        (state[i] == hungry) &&
        (state[(i + 1) % 5] != eating)) {
        state[i] = eating;
        self[i].signal();
    }
}
```

The Sleeping Barber Problem

(1)
Sleeping Barber Problem

The barber shop has one barber, one barber chair, and n waiting chairs.
If there are no customers, barber falls asleep. When customer arrives he/she has to wake up sleeping barber.
If additional customer arrive when barber is cutting they wait (if any empty chair) or leave (if all chairs are taken).