CS302
Operating System
Lab 5
Mutual Exclusion

Xinxun Zeng, Shiqi Zhang, Dongping Zhang

Race Condition

- The outcome of an execution depends on a particular order in which the shared resource is accessed.

- A simple example: The too much milk problem

The too much milk problem

- Description:

Mom and Dad are all used to checking the fridge when they arrive home. If milk run out, he or she will leave home to buy milk. The fridge is small in your home that only one bottle of milk can be put in it at a time. Mom and Dad always arrive home at different time.

The too much milk problem

Time	Dad	Mom
3:00	Arrive Home	
3:05	Look in fridge, no milk	
3:10	Leave for supermarket	
3:15		Arrive Home
3:20	Arrive at Supermarket	Look in fridge, no milk
3:25	Buy Milk	Leave for supermarket
3:30	Arrive home, put milk into fridge	
3:35		Arrive at Supermarket
3:40		Buy milk
3:45		Arrive home, put milk in fridge
3:50		Oh .....

The too much milk problem

- Compile and run like this:
  - gcc milk.c -o milk
  - ./milk
- Result:

```
Mom comes home.
Mom checks the fridge.
Mom goes to buy milk...
Dad comes home.
Mom puts milk in fridge and leaves.
Dad checks the fridge.
Dad closes the fridge and leaves.
```

```
Dad comes home.
Mom comes home.
Mom checks the fridge.
Mom goes to buy milk...
Dad checks the fridge.
Dad goes to buy milk...
Mom puts milk in fridge and leaves.
Dad puts milk in fridge and leaves.
What a waste of food! The fridge can not hold so much milk!
```

http://www.cse.cuhk.edu.hk/~ericlo/teaching/os/lab/7-IPC2/race_con.html

The too much milk problem

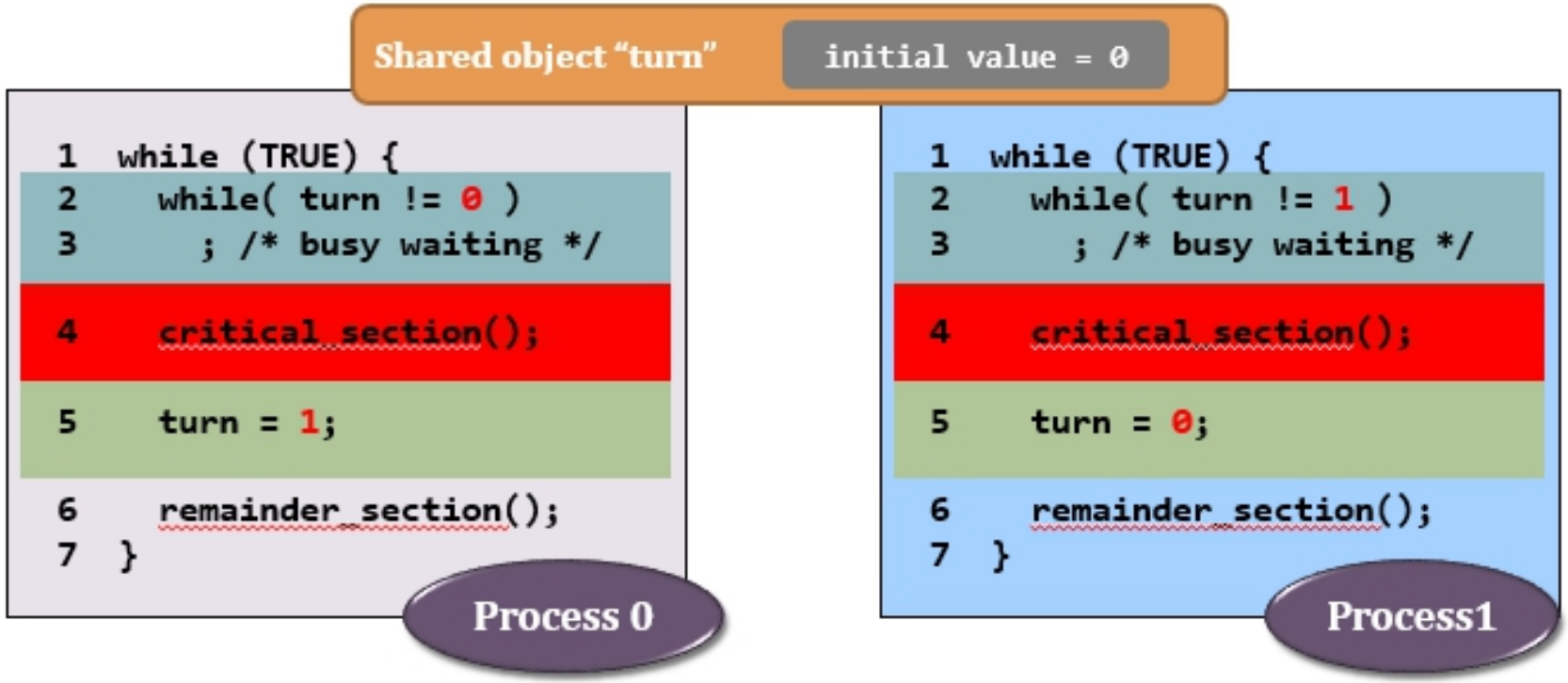
- mom

```c
void *mom(){
    int fd;
    printf("Mom comes home.\n");
    sleep(rand()%2+1);
    printf("Mom checks the fridge.\n");
    fd=open("fridge", O_CREAT|O_RDWR|O_APPEND, 0777);
    if(lseek(fd,0,SEEK_END)==0){
        printf("Mom goes to buy milk...\n");
        sleep(rand()%2+1);
        printf("Mon comes back.\n");
        if(lseek(fd,0,SEEK_END)>0)
            printf("What a waste of food! The fridge can not hold so much milk!\n");
        else{
            write(fd,"milk",4);
            printf("Mom puts milk in fridge and leaves.\n");
        }
    }else{
        printf("Mom closes the fridge and leaves.\n");
    }
    close(fd);
}
```

http://www.cse.cuhk.edu.hk/~ericlo/teaching/os/lab/7-IPC2/race_con.html

Busy Waiting

- Loop on yet another shared object ***turn*** to detect the status of other processes
- Example: spin_lock



Practice

- How to fix **the too much milk problem** by busy waiting?
  (Hint: you could use pthread_spin_lock)

Pthread_spin_lock API

- int pthread_spin_init (pthread_spinlock_t *, int);
- int pthread_spin_lock(pthread_spinlock_t *)
- int pthread_spin_unlock(pthread_spinlock_t *)

Sleep Waiting

- During waiting, the CPU would work on other task and no CPU are wasted.
- E.g. Mutex Lock, Semaphore

Practice

- How to fix **the too much milk problem** by spin-based waiting?
  (Hint: use pthread_mutex_lock)

Pthread_mutex_lock API

- int pthread_mutex_lock(pthread_mutex_t *)
- int pthread_mutex_unlock(pthread_mutex_t *)

Spin_lock vs Mutex

- Is mutex always better than spin_lock?
- Not exactly. Why?

```
lock
i++
unlock
```

```
lock
…
…
…
…
…
unlock
```

Better way

- Mom see the fridge is empty.
- Mom leave a note and then go buy milk.
- Dad open the fridge and see the note.
- Dad just go away.

- new API:
- int pthread_mutex_trylock (pthread_mutex_t *)
- int pthread_spin_trylock (pthread_spinlock_t *)

Semaphores

- Semaphore is a variable that has an integer value

  Initialize: a nonnegative integer value

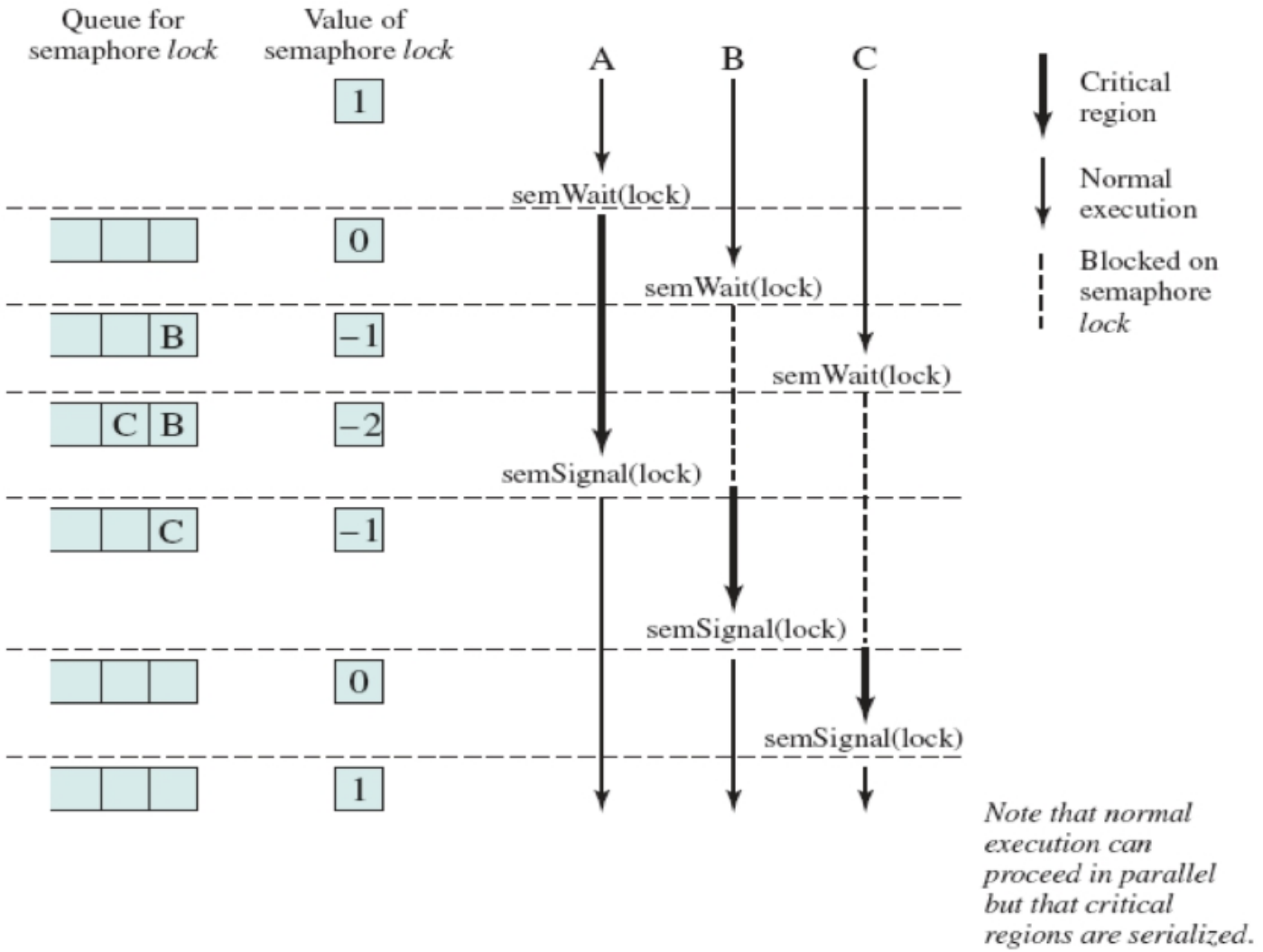
  semWait (P): decreases the semaphore value. the value becomes negative, then the process executing the semWait is blocked.

  semSignal (V): increases semaphore value. If the resulting value is less than or equal to zero, then a process is blocked by a semWait operation, if any, is unblocked.

Semaphores

```
struct semaphore {
        int count;
        queueType queue;
};
void semWait(semaphore s)
{
        s.count--;
        if (s.count < 0) {
          /* place this process in s.queue */;
          /* block this process */;
        }
}
void semSignal(semaphore s)
{
        s.count++;
        if (s.count<= 0) {
          /* remove a process P from s.queue */;
          /* place process P on ready list */;
        }
}
```

Mutual Exclusion using Semaphores



Semaphore API in Linux

Function	Description
sem_open	Opens/creates a named semaphore for use by a process
sem_wait	lock a semaphore
sem_post	unlock a semaphore
sem_close	Deallocates the specified named semaphore
sem_unlink	Removes a specified named semaphore
sem_getvalue	get semaphore value

- **semaphore.c** shows how to use these functions to create, operate and remove named semaphore.
- compile semaphore.c like this:
  - **gcc semaphore.c -pthread -o semaphore**

http://www.cse.cuhk.edu.hk/~ericlo/teaching/os/lab/7-IPC2/race_con.html

Practice

- Remember the milk problem.
- What if the fridge has space to store 2 bottles of milk?
- Besides mom and dad, you are the third people will buy milk.

- How to handle **the too much milk problem** by semaphore ?

Mutex vs Semaphore

- Is binary semaphore equals to mutex?
- Not exactly.

- Mutex is more about resource **protection**.
- Semaphore is more about resource **assignment**.
- Mutex can only **be unlock by container**.
- Semaphore can **be assigned by anyone**, including caller itself.
- Mutex lock **will be released**, if the holder is terminate.
- Semaphore **will not add up**, if the holder is terminate.

Your home has new problem

- Your family buy a new big fridge, which can put in 100 bottles of milk.
- Dad and you always take milk but never buy.
- Mom is very frequently checking the fridge is empty or not.
- If fridge is empty, she will go buy milk. Otherwise do nothing.

```
while (1){
    lock
    int num=check_fridge()
    if(num>0)
        take milk
    unlock
}
```

Dad and you

```
while (1){
    lock
    int num=check_fridge()
    if(num==0)
        go buy milk
    unlock
}
```

Mom

Condition variable

- used to **wait** for a particular **condition to become true**.
- Cond = a condition + a mutex

- When we need condition variable?
  - when If/else is unbalance

Pthread_cond API

- int pthread_cond_wait (pthread_cond_t *, pthread_mutex_t *);
  - release lock, put thread to sleep until condition is signaled;
  - when thread wakes up again, re-acquire lock before returning

- int pthread_cond_signal (pthread_cond_t *);
  - Wake up at least one of the threads that are blocked on the specified condition variable.
  - If more than one thread is blocked on a condition variable, the scheduling policy shall determine the order in which threads are unblocked.

Solution

Try to realize it

```
while (1){
    lock
    int num=check_fridge()
    if(num>0)
        take milk
    else  cond_signal
    unlock
}                                   Mon and you
```

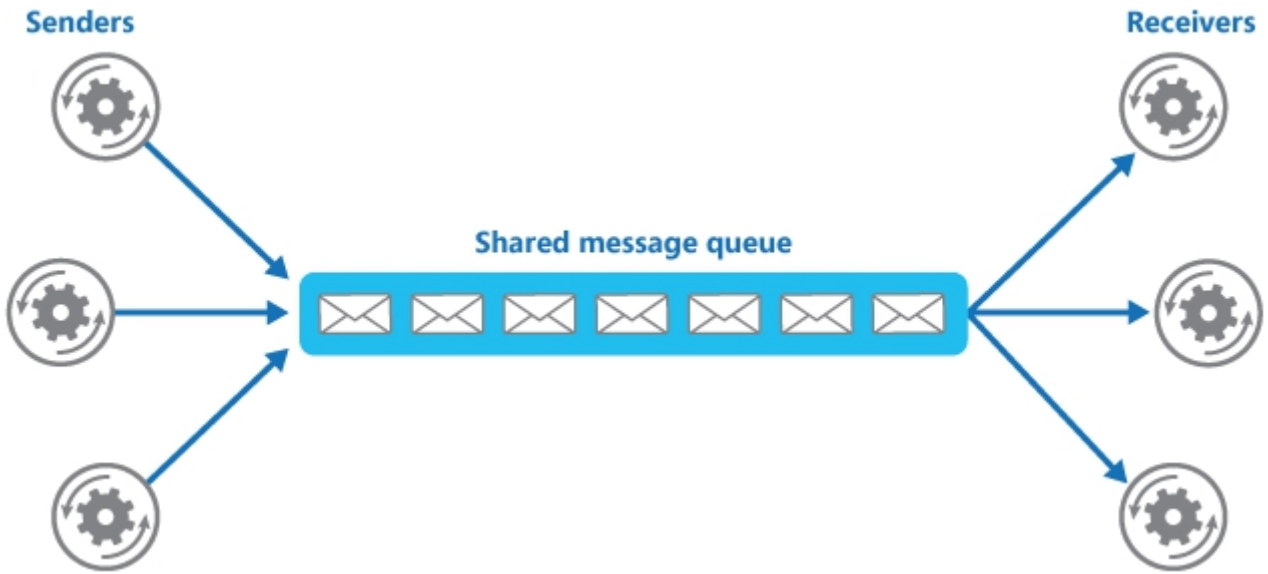
```
while (1){
    lock//lock mutex
    while(check_fridge()>0)
        cond_wait  //wait for cond_signal, unlock mutex
    go buy milk
    unlock
}                                   Brother
```

```
while (1){
    lock//lock mutex
    while(check_fridge()>0)
        cond_wait  //wait for cond_signal and unlock mutex
    go buy milk
    unlock
}                                   Dad
```

Three typical problem

- Producers and Consumers
- Readers and Writers
- Dining philosopher(Not in this lab)

Producers and Consumers



Producers and Consumers

- There is a queue, has a fixed size n.
- Producer can produce when queue is not full
- Consumer can consume when queue is not empty
- At any time, only one person can access the queue

- How?

Readers-Writers Problem

- There is a data area shared among a number of processes.
- Data area: a file, a block of main memory, a bank of processor registers and etc.

- The conditions that must be satisfied are as follows:
  - Any number of readers can **simultaneously read** the file
  - Only **one writer** at a time may write to the file
  - If a **writer is writing** to the file, **no reader** can read it

Scenarios:

- When new reader coming,
  - If no reader and writer, then new reader can read.
  - If writer is waiting and other readers are reading, then new reader can read.
  - If writer is writing, then new reader waits.

- When new writer coming,
  - If no reader, then new writer can write.
  - If reader is reading, then new writer waits.
  - If writer is writing, then new writer waits.

Assignment: Readers-Writers Problem

- Please complete the report
- Please complete ”read.h” and “write.h”, and
  You should implement according to the output format in "output_sample.txt“
- Check the **blackboard** for deadline