



# Homework Assignment #4

## Race Condition & Mutex



# Outline

- **Race Condition**
- **Mutex Locks**
- **Application Programming Interface**
  - Exercise 3.20 API
  - Exercise 4.28 API
  - Pthread Mutex
- **Homework Assignment #3**
- **Reference**

# Race Condition

- A situation that several threads access the same data concurrently and the outcome depends on the uncontrollable sequence.

| Thread 1                  | Thread 2                  | bitmap/300/ | Pid_th1 | Pid_th2 |
|---------------------------|---------------------------|-------------|---------|---------|
| if( bitmap/300/ == 0<br>) |                           | 0           | NULL    | NULL    |
|                           | if( bitmap/300/ == 0<br>) | 0           | NULL    | NULL    |
| bitmap/300/ = 1           |                           | 1           | 300     | NULL    |
|                           | bitmap/300/ = 1           | 1           | 300     | 300     |

**Race  
Condition**

# Race Condition

```
tid is 140172315629312
pid is 0, will sleep 7 seconds
tid is 140172324022016
pid is 1, will sleep 2 seconds
tid is 140172307236608
pid is 0, will sleep 7 seconds
tid is 140172374378240
tid is 140172357592832
pid is 4, will sleep 8 seconds
pid is 6, will sleep 6 seconds
tid is 140172365985536
pid is 2, will sleep 10 seconds
tid is 140172349200128
pid is 3, will sleep 5 seconds
tid is 140172298843904
pid is 0, will sleep 1 seconds
tid is 140172340807424
pid is 5, will sleep 8 seconds
tid is 140172332414720
pid is 7, will sleep 6 seconds
```

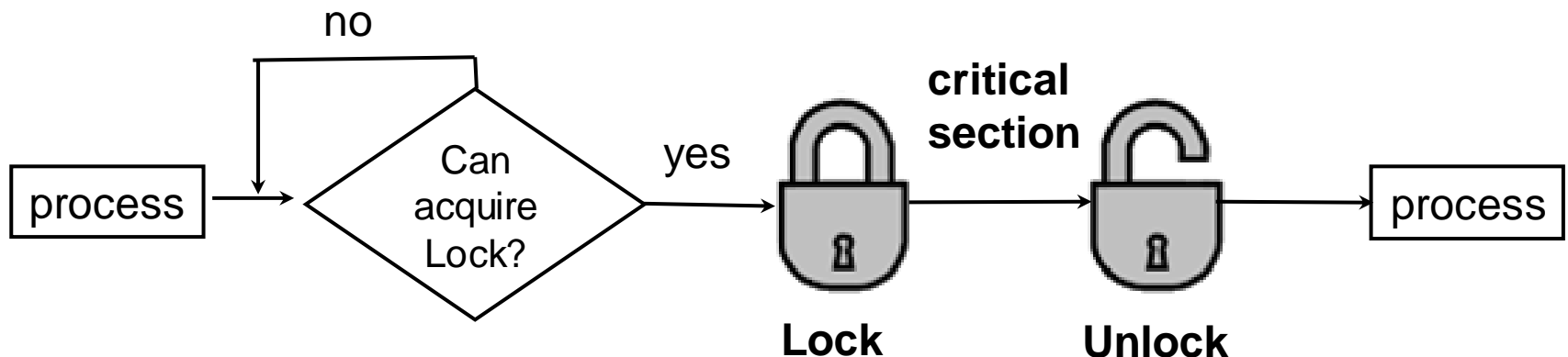


# Outline

- **Race Condition**
- **Mutex Locks**
- **Application Programming Interface**
  - Exercise 3.20 API
  - Exercise 4.28 API
  - Pthread Mutex
- **Homework Assignment #3**
- **Reference**

# Mutex Locks

- We use the mutex lock to protect critical sections and thus prevent **race conditions**.
- A process must acquire the lock before entering a critical section; it releases the lock when it exits the critical section.
- A mutex lock has a boolean variable whose value indicates if the lock is available or not.





# Outline

- **Race Condition**
- **Mutex Locks**
- **Application Programming Interface**
  - Exercise 3.20 API
  - Exercise 4.28 API
  - Pthread Mutex
- **Homework Assignment #3**
- **Reference**

# Exercise 3.20 API

- We have used three APIs in homework#1.

- *int allocate map( void ) :*

Initializes a data structure for representing pids;  
returns -1 if unsuccessful, 1 if successful

- *int allocate pid( void ) :*

Allocates and returns a pid; returns -1 if unable to allocate a pid  
(all pids are in use)

- *void release pid( int pid ) :*

Releases a pid



# Exercise 4.28 API

- We have used three Pthreads APIs in homework#3.
  - *#include <pthread.h>*
  - *int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine) (void \*), void \*arg);*

Create a thread

- *int pthread\_join(pthread\_t thread, void \*\*value\_ptr);*  
Wait for a thread

Causes the caller to wait for the specified thread to exit

- *int pthread\_exit(void \*value\_ptr);*

Exit a thread without exiting process

# Pthread Mutex Locks

- There is a whole set of library calls associated with mutex locks, most of whose names start with **pthread\_mutex**
- To use these library calls, we must include the file **pthread.h**, and link with the pthread library using **-pthread**

```
luke@luke-VirtualBox:~$ vim hw4.c  
luke@luke-VirtualBox:~$ gcc hw4.c -o hw4 -pthread
```

```
#include <pthread.h>  
pthread_mutex_t mutex;
```

# Pthread Mutex Locks

- We will use the following four functions
  - *int pthread\_mutex\_init()*
    - Initialize a mutex.
  - *int pthread\_mutex\_lock()*
    - Lock the critical section.
  - *int pthread\_mutex\_unlock()*
    - Unlock the critical section.
  - *int pthread\_mutex\_destroy()*
    - Release the resource and destroy a mutex.

# Pthread Mutex Locks

- **pthread\_mutex\_init**
  - Initializes the mutex lock.

```
#include<pthread.h>
int pthread_mutex_init(pthread_mutex_t *mutex,
const pthread_mutexattr_t *mattr);
```

```
EX: pthread_mutex_init(& mutex , NULL);
```

**mutex**: Pointer to the mutex to be initialized.

**mattr**: Use the attributes to initialize the mutex. NULL for the default values.

On success, returns 0. On error, one of the following values is returned : EAGAIN, EINVAL , EFAULT , ENOMEM

# Pthread Mutex Locks

- **pthread\_mutex\_lock**
  - Lock the critical section.

```
int pthread_mutex_lock( pthread_mutex_t* mutex );
```

```
EX: pthread_mutex_lock(& mutex );
```

**mutex**: A pointer to the pthread\_mutex\_t object that you want to lock. The ***pthread\_mutex\_lock()*** locks the mutex object referenced by ***mutex***.

If the mutex is already locked, then the calling thread blocks until it has acquired the mutex.

On success, returns 0. On error, one of the following values is returned : EAGAIN, EINVAL , EFAULT , ENOMEM

# Pthread Mutex Locks

- **pthread\_mutex\_unlock**

- Unlock the critical section.

```
int pthread_mutex_unlock( pthread_mutex_t* mutex );
```

```
EX: pthread_mutex_unlock(& mutex );
```

**mutex**: A pointer to the pthread\_mutex\_t object that you want to unlock.  
The **pthread\_mutex\_unlock()** unlocks the **mutex**.

If **mutex** has been locked more than once, it must be unlocked the same number of times before the next thread is given ownership of the mutex.

On success, returns 0. On error, one of the following values is returned :  
EAGAIN, EINVAL , EFAULT , ENOMEM

# Pthread Mutex Locks

- **pthread\_mutex\_destroy**

- Destroys a previously declared mutex.

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

```
EX: pthread_mutex_destroy (& mutex );
```

**mutex:** Pointer to the mutex to be destroyed.

The mutex mustn't be used after it has been destroyed.

On success, returns 0. On error, one of the following values is returned : EAGAIN , EINVAL , EFAULT , ENOMEM

# Pthread Mutex Locks\_Example

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define MAXPID 100000
pthread_mutex_t mutex; // Declare the name of pthread_mutex_t .
int glob=0;
//This is thread function
void *threadFunc() {
    int i=0;
    printf("-----\n");
    printf("This is thread function\n");
    printf("thread ID: %lu\n", pthread_self());
    printf("Mutex Lock the critical section.\n");
    //critical section start
    pthread_mutex_lock( &mutex ); //Lock the critical section.

    while(i<MAXPID){
        i++;
        glob++;
    }
    pthread_mutex_unlock ( &mutex); //Unlock the critical section.
    //critical section end
    printf("Mutex unlock the critical section.\n");

    printf("sum : %d\n", i);
    printf("-----\n");
    pthread_exit(NULL);
}
```

```
void *threadFunc2() {
    int i=0;
    printf("-----\n");
    printf("This is thread2 function\n");
    printf("thread ID: %lu\n", pthread_self());
    printf("Mutex Lock the critical section.\n");
    //critical section start
    pthread_mutex_lock( &mutex );
        //Lock the critical section.
    while(i<MAXPID){
        i++;
        glob++;
    }
    pthread_mutex_unlock ( &mutex);
        //Unlock the critical section.
    //critical section end
    printf("Mutex unlock the critical section.\n");

    printf("sum : %d\n", i);
    printf("-----\n");
    pthread_exit(NULL);
}
```



# Pthread Mutex Locks\_Example(cont.)

```
int main(int argc, char** argv)
{
    pthread_t thread,thread2;
    pthread_mutex_init(& mutex , NULL );    //Initializes the mutex.
    int rc, t=100;
    void *reBuf;
    rc = pthread_create(&thread, NULL, threadFunc, NULL);
    rc = pthread_create(&thread2, NULL, threadFunc2, NULL);
    if(rc)
    {
        printf("ERROR; return code from pthread_create() is %d\n", rc);
        exit(-1);
    }
    pthread_join(thread, &reBuf);
    pthread_join(thread2, &reBuf);

    //Release the resource and destroy a mutex.
    pthread_mutex_destroy (& mutex );
    printf("%d\n",glob);
    return 0;
}
```

# Pthread Mutex Locks\_Example(cont.)

```
-----  
This is thread function  
thread ID: 140558076548864  
Mutex Lock the critical section.  
-----  
This is thread2 function  
thread ID: 140558068156160  
Mutex Lock the critical section.  
Mutex unlock the critical section.  
sum : 100000  
-----  
Mutex unlock the critical section.  
sum : 100000  
-----  
200000
```



# Outline

- **Race Condition**
- **Mutex Locks**
- **Application Programming Interface**
  - Exercise 3.20 API
  - Exercise 4.28 API
  - Pthread Mutex
- **Homework Assignment #4**
- **Reference**

# Homework Assignments #3

- Use Pthreads API to create 100 threads.
- For each thread
  - Step 1. Use **Mutex Lock** to protect PID manager, which can allocate PID for each thread. (PID range : 0~99)
  - Step 2. Use **Mutex Unlock** after allocating PID.
  - Step 3. Let thread sleep for 1~3 seconds.
  - Step 4. When the thread wake up, using **Mutex Lock** to protect PID manager, which can release its PID.
  - Step 5. Use **Mutex Unlock** after releasing PID.
  - Step 6. Terminate the thread and Destroy the mutex.

# Mutex Lock Flowchart

Create thread

pthread\_create()

pthread\_mutex\_lock()

Mutex Lock

Critical Section

allocate\_pid()

pthread\_mutex\_unlock()

Mutex Unlock

Thread sleep

sleep()

Thread wake up

pthread\_mutex\_lock()

Mutex Lock

Critical Section

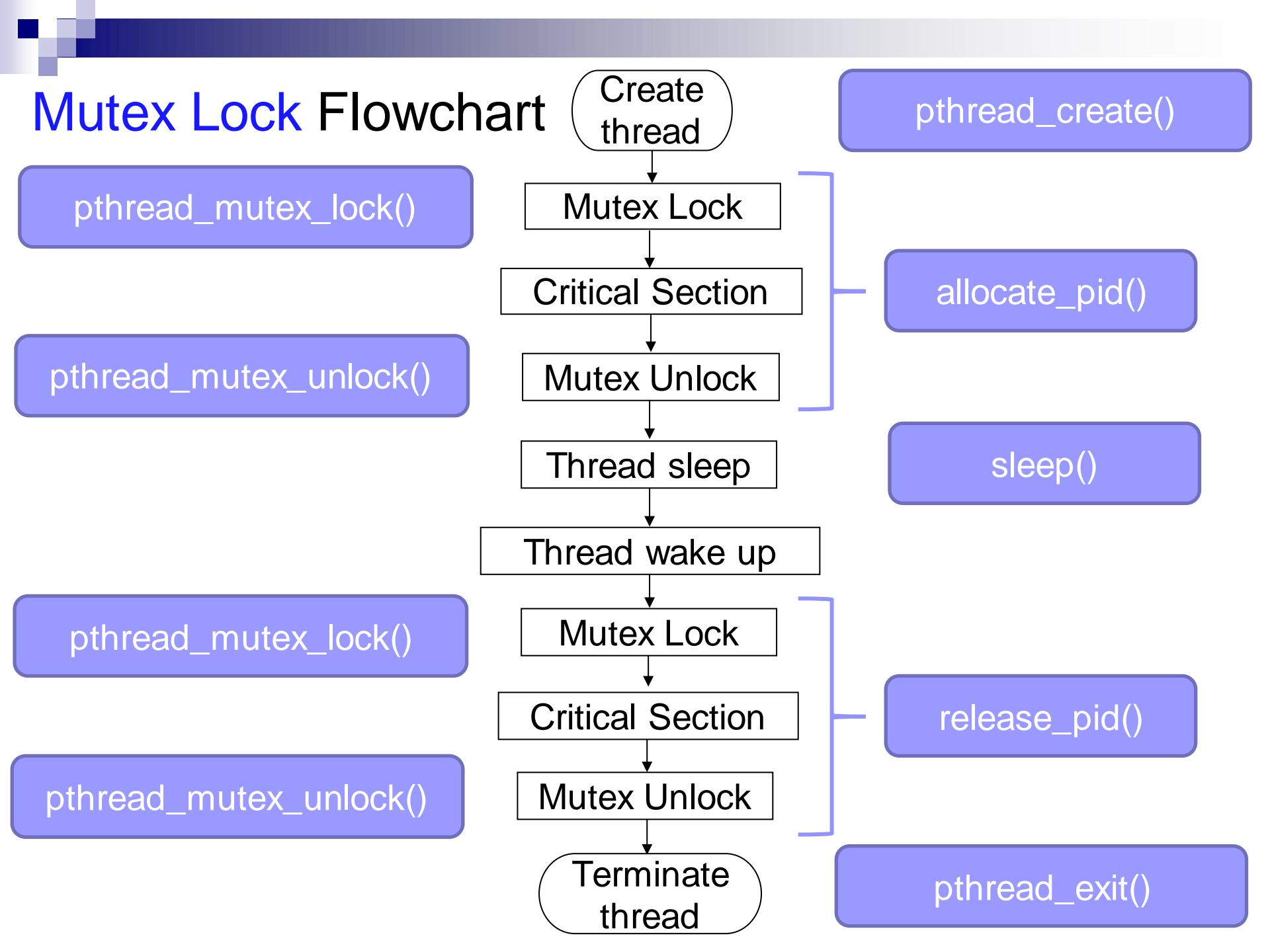
release\_pid()

pthread\_mutex\_unlock()

Mutex Unlock

Terminate thread

pthread\_exit()





# Result

```
pid is 0, will sleep 8 seconds
tid is 139957604431424
pid is 1, will sleep 7 seconds
tid is 139957596038720
pid is 2, will sleep 3 seconds
tid is 139957570860608
pid is 3, will sleep 10 seconds
tid is 139957562467904
pid is 4, will sleep 2 seconds
tid is 139957545682496
pid is 5, will sleep 6 seconds
tid is 139957537289792
pid is 6, will sleep 7 seconds
tid is 139957612824128
pid is 7, will sleep 4 seconds
tid is 139957587646016
pid is 8, will sleep 10 seconds
tid is 139957579253312
pid is 9, will sleep 6 seconds
tid is 139957554075200
pid is 10, will sleep 8 seconds
tid is 139957528897088
pid is 11, will sleep 6 seconds
tid is 139957520504384
pid is 12, will sleep 5 seconds
tid is 139957512111680
pid is 13, will sleep 3 seconds
tid is 139957495326272
pid is 14, will sleep 4 seconds
tid is 139957503718976
pid is 15, will sleep 6 seconds
tid is 139957486933568
pid is 16, will sleep 2 seconds
tid is 139957478540864
pid is 17, will sleep 3 seconds
tid is 139957461755456
pid is 18, will sleep 5 seconds
tid is 139957470148160
pid is 19, will sleep 7 seconds
tid is 139957453362752
pid is 20, will sleep 6 seconds
tid is 139957419791936
pid is 24, will sleep 8 seconds
tid is 139957428184640
pid is 23, will sleep 3 seconds
tid is 139957436577344
pid is 21, will sleep 5 seconds
tid is 139957444970048
pid is 22, will sleep 5 seconds
tid is 139957411399232
```

```
pid is 25, will sleep 2 seconds
tid is 139957463006528
pid is 26, will sleep 6 seconds
tid is 139957394613824
pid is 27, will sleep 6 seconds
tid is 139957386221120
pid is 28, will sleep 10 seconds
tid is 139957377828416
pid is 29, will sleep 3 seconds
tid is 139957369435712
pid is 30, will sleep 5 seconds
tid is 139957361043008
pid is 31, will sleep 8 seconds
tid is 139957352650304
pid is 32, will sleep 2 seconds
tid is 139957344257600
pid is 33, will sleep 9 seconds
tid is 139957335864896
pid is 34, will sleep 9 seconds
tid is 139957319079488
pid is 35, will sleep 5 seconds
tid is 139957327472192
pid is 36, will sleep 7 seconds
tid is 139957310686784
pid is 37, will sleep 6 seconds
tid is 139957302294080
pid is 38, will sleep 8 seconds
tid is 139957293901376
pid is 39, will sleep 8 seconds
tid is 139957285508672
pid is 40, will sleep 1 seconds
tid is 139957277115968
pid is 41, will sleep 7 seconds
tid is 139957268723264
pid is 42, will sleep 3 seconds
tid is 139957226759744
pid is 43, will sleep 7 seconds
tid is 139957251937856
pid is 44, will sleep 10 seconds
tid is 139957260330560
pid is 45, will sleep 7 seconds
tid is 139957235152448
pid is 46, will sleep 3 seconds
tid is 139957243545152
pid is 47, will sleep 1 seconds
tid is 139957218367040
pid is 48, will sleep 1 seconds
tid is 139957209974336
pid is 49, will sleep 7 seconds
tid is 139957201581632
```

```
pid is 50, will sleep 7 seconds
tid is 139957151225408
pid is 56, will sleep 9 seconds
tid is 139957142832704
pid is 57, will sleep 4 seconds
tid is 139956949800512
tid is 139957117654592
pid is 63, will sleep 1 seconds
tid is 139957092476480
pid is 64, will sleep 3 seconds
pid is 80, will sleep 9 seconds
tid is 139957016942144
pid is 72, will sleep 3 seconds
tid is 139957042120256
pid is 69, will sleep 8 seconds
tid is 139957075691072
pid is 62, will sleep 6 seconds
tid is 139957100869184
tid is 139957109261888
pid is 60, will sleep 2 seconds
pid is 65, will sleep 3 seconds
tid is 139957067298368
tid is 139957159618112
tid is 139957033727552
pid is 70, will sleep 1 seconds
pid is 55, will sleep 4 seconds
tid is 139957168010816
pid is 54, will sleep 3 seconds
tid is 139957176403520
pid is 53, will sleep 10 seconds
tid is 139957025334848
tid is 139957193188928
pid is 51, will sleep 8 seconds
tid is 139957008549440
pid is 73, will sleep 5 seconds
tid is 139957084083776
pid is 61, will sleep 7 seconds
tid is 139956991764032
pid is 74, will sleep 8 seconds
tid is 139957184796224
pid is 52, will sleep 7 seconds
tid is 139957134440000
pid is 58, will sleep 4 seconds
tid is 139956983371328
pid is 76, will sleep 1 seconds
tid is 139956974978624
pid is 77, will sleep 4 seconds
tid is 139956874266176
pid is 71, will sleep 9 seconds
tid is 139956807124544
```

```
pid is 66, will sleep 2 seconds
tid is 139956933015104
pid is 82, will sleep 8 seconds
tid is 139956916229696
pid is 84, will sleep 7 seconds
tid is 139956907836992
pid is 85, will sleep 1 seconds
tid is 139956891051584
tid is 139956790339136
pid is 99, will sleep 5 seconds
tid is 139956857480768
pid is 91, will sleep 9 seconds
tid is 139957126047296
pid is 59, will sleep 9 seconds
tid is 139957058905664
pid is 67, will sleep 6 seconds
tid is 139956849088064
pid is 92, will sleep 4 seconds
tid is 139956865873472
pid is 90, will sleep 9 seconds
tid is 139956966585920
pid is 78, will sleep 8 seconds
tid is 139957000156736
pid is 75, will sleep 1 seconds
pid is 89, will sleep 5 seconds
tid is 139956958193216
pid is 79, will sleep 4 seconds
pid is 96, will sleep 9 seconds
tid is 139956815517248
pid is 97, will sleep 1 seconds
tid is 139956798731840
pid is 98, will sleep 5 seconds
tid is 139956941407808
pid is 81, will sleep 6 seconds
tid is 139956924622400
pid is 83, will sleep 2 seconds
tid is 139956899444288
pid is 86, will sleep 10 seconds
tid is 139956823909952
pid is 95, will sleep 10 seconds
pid is 87, will sleep 4 seconds
tid is 139956882658880
pid is 88, will sleep 3 seconds
tid is 139957050512960
pid is 68, will sleep 7 seconds
tid is 139956840695360
pid is 93, will sleep 10 seconds
tid is 139956832302656
pid is 94, will sleep 9 seconds
```

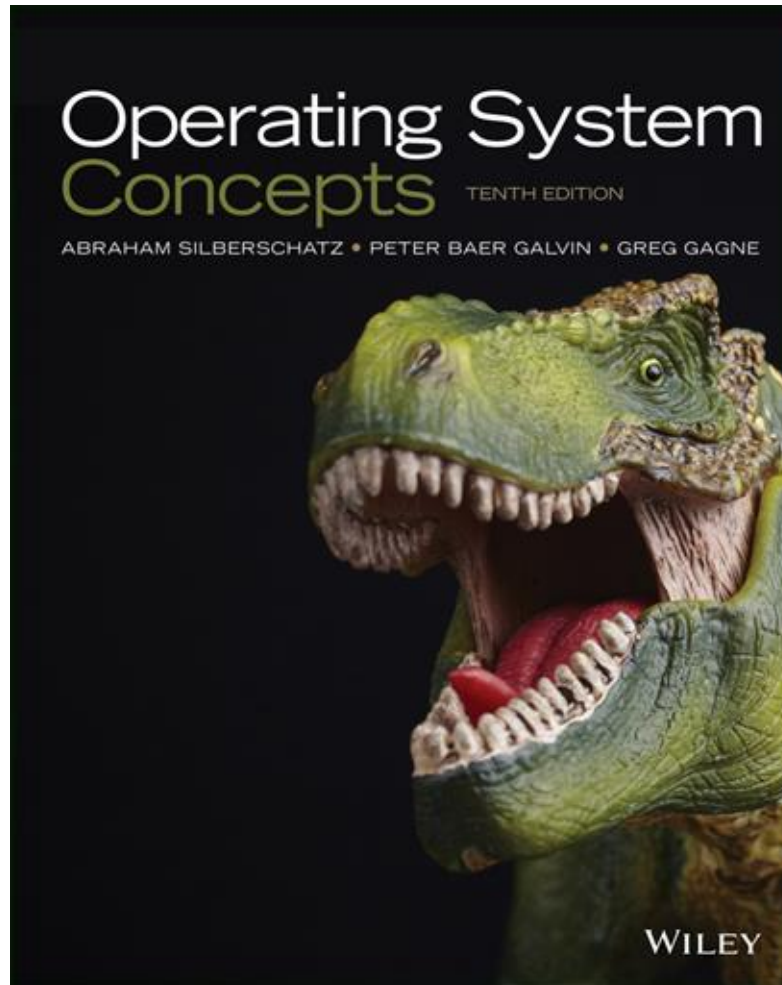


# Outline

- **Race Condition**
- **Mutex Locks**
- **Application Programming Interface**
  - Exercise 3.20 API
  - Exercise 4.28 API
  - Pthread Mutex
- **Homework Assignment #4**
- **Reference**

# Reference

- Operating System Concepts, 10th Edition





# Turn in

- Deadline  
2023/12/26 PM.11:59:59
- Upload to iLearning
- File name
  - HW4\_ID.zip (e.g. HW4\_4106056000.zip)
    - Source code
      - .c file
    - Word
- If you don't hand in your homework on time, your score will be deducted 10 points every day.

# TA

- Name : Gang, Yang
- Email : pupss96162@gmail.com
  - Title format : OS HW#4 - [your name]
- Lab: OSNET(1001)