Homework Assignment #3
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

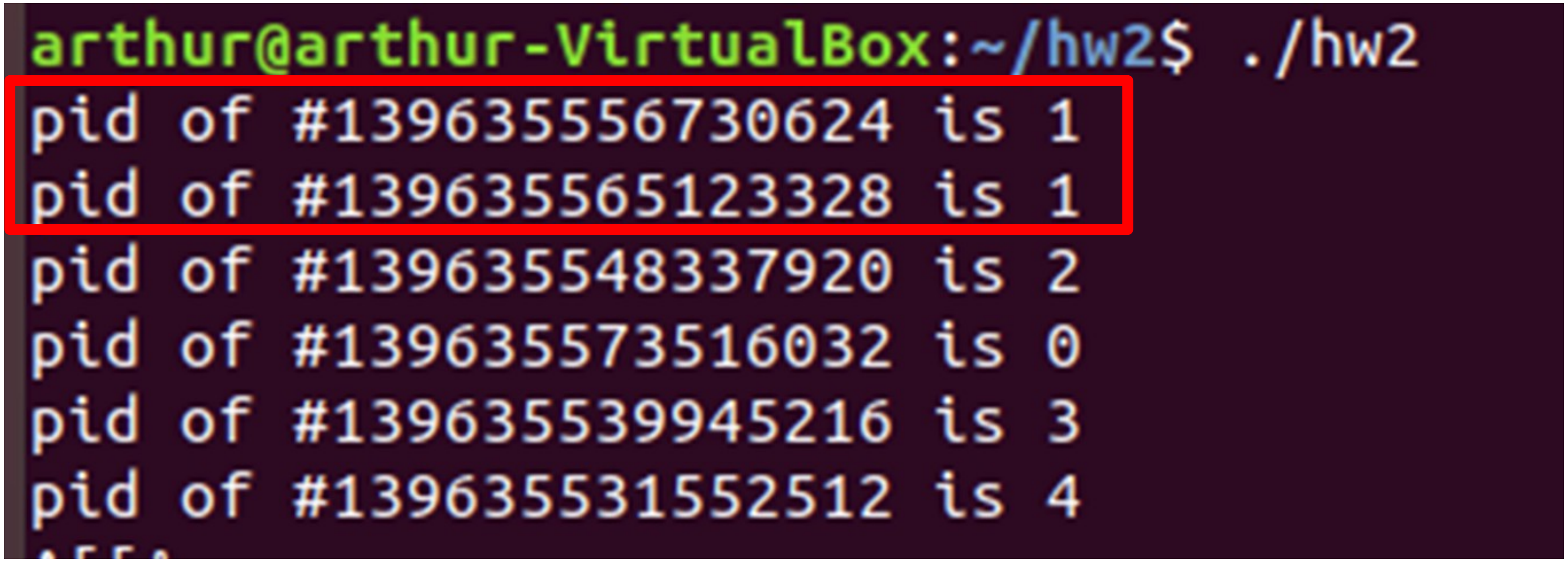
Race Condition

- 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)		0	NULL	NULL
	if(bitmap/300/ == 0)	0	NULL	NULL
bitmap/300/ = 1 →		1	300	NULL
	bitmap/300/ = 1 →	1	300	300

Race Condition

Race Condition



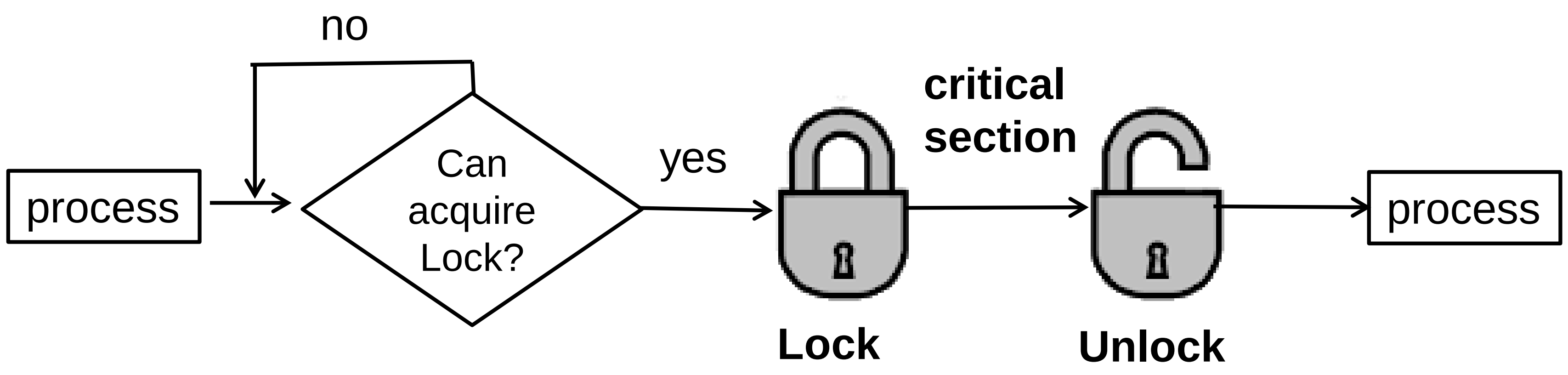
```
arthur@arthur-VirtualBox:~/hw2$ ./hw2
pid of #139635556730624 is 1
pid of #139635565123328 is 1
pid of #139635548337920 is 2
pid of #139635573516032 is 0
pid of #139635539945216 is 3
pid of #139635531552512 is 4
```

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 created 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 created three Pthreads APIs in homework#2.
  - ***#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);***

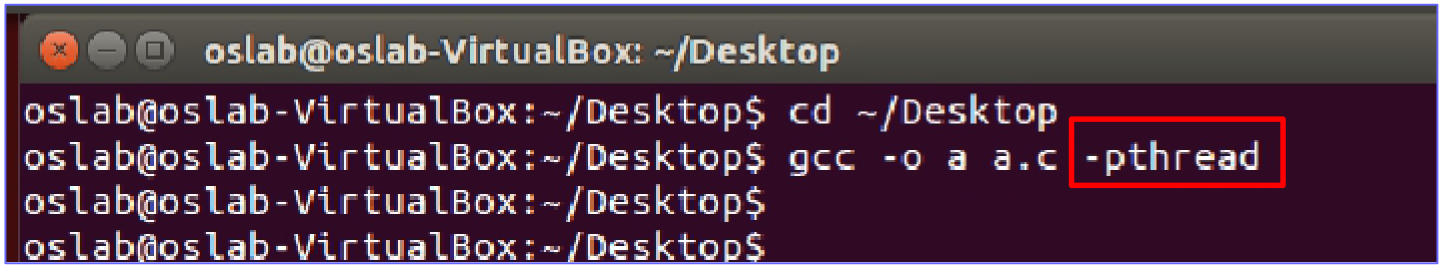
    Wait for a thread

    Causes the caller to wait for the specified thread to exit
  - ***int pthread_exit(void *value_ptr);***

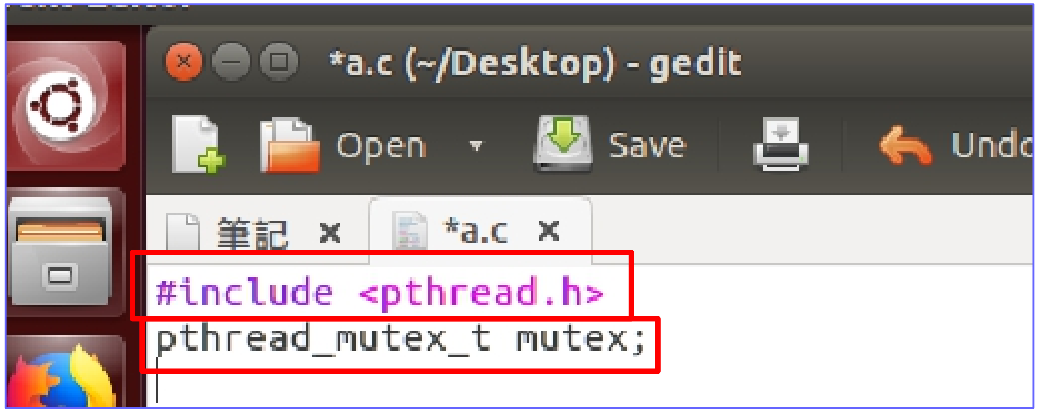
    Exit a thread without exiting process

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`**



```
oslab@oslab-VirtualBox:~/Desktop$ cd ~/Desktop
oslab@oslab-VirtualBox:~/Desktop$ gcc -o a a.c -pthread
oslab@oslab-VirtualBox:~/Desktop$
oslab@oslab-VirtualBox:~/Desktop$
```



```
#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

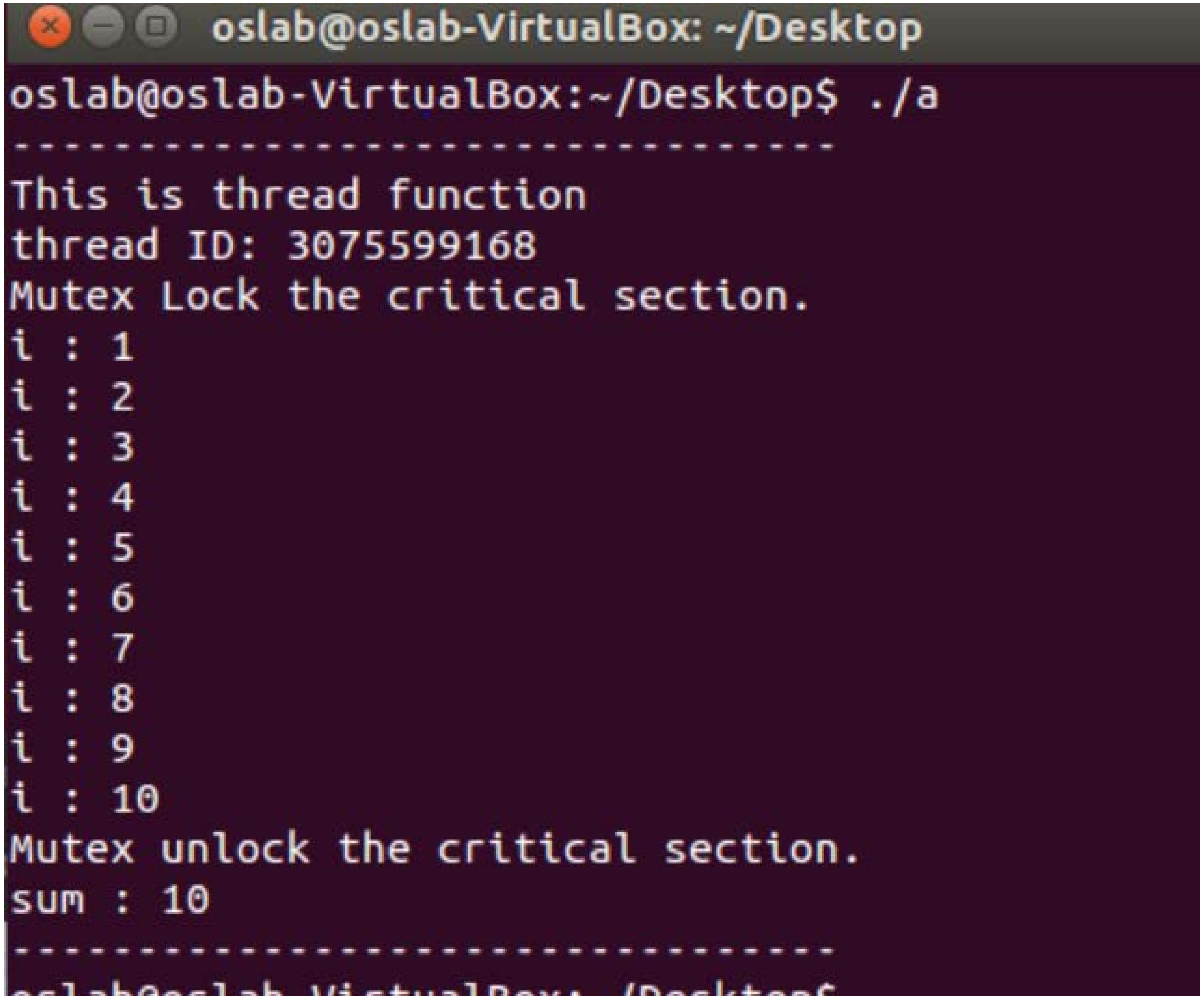
Pthread Mutex Locks_Example

```
#include <stdio.h>            #include <stdlib.h>           #include <pthread.h>          #define MAXPID 10
pthread_mutex_t mutex;    // Declare the name of pthread_mutex_t .

//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++;
                    printf("i : %d\n", i);
    }
    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;
        pthread_mutex_init(& mutex  , NULL );        //Initializes the mutex.
        int rc, t=100;
        void *reBuf;
        rc = pthread_create(&thread, NULL, threadFunc, NULL);
        if(rc)
        {
                printf("ERROR; return code from pthread_create() is %d\n", rc);
                exit(-1);
        }
        pthread_join(thread, &reBuf);

        //Release the resource  and destroy a mutex.
        pthread_mutex_destroy (& mutex );
  return 0;
}
```

Pthread Mutex Locks_Example(cont.)



```
oslab@oslab-VirtualBox:~/Desktop$ ./a
---------------------------------
This is thread function
thread ID: 3075599168
Mutex Lock the critical section.
i : 1
i : 2
i : 3
i : 4
i : 5
i : 6
i : 7
i : 8
i : 9
i : 10
Mutex unlock the critical section.
sum : 10
---------------------------------
oslab@oslab-VirtualBox:~/Desktop$
```

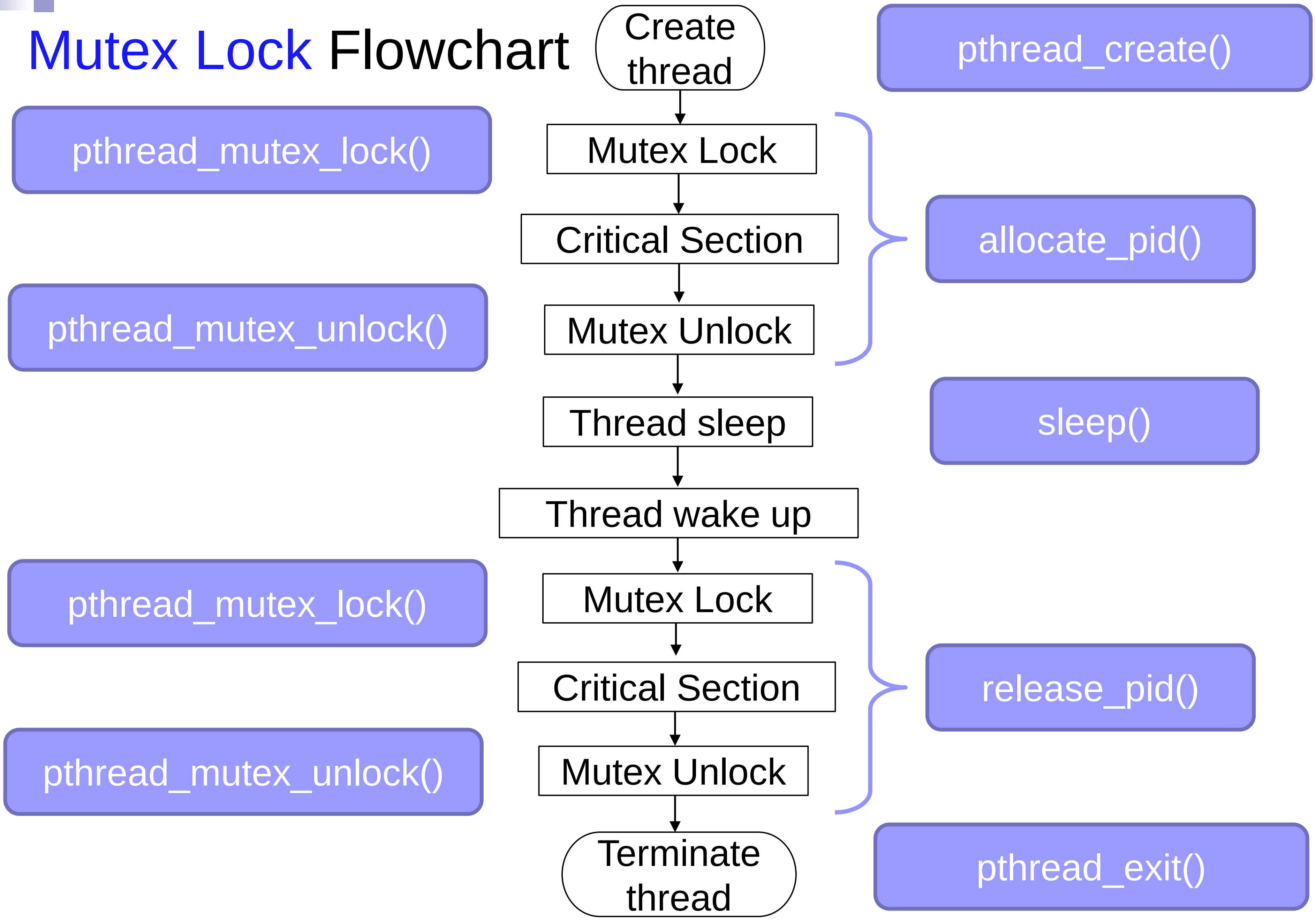
Outline

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

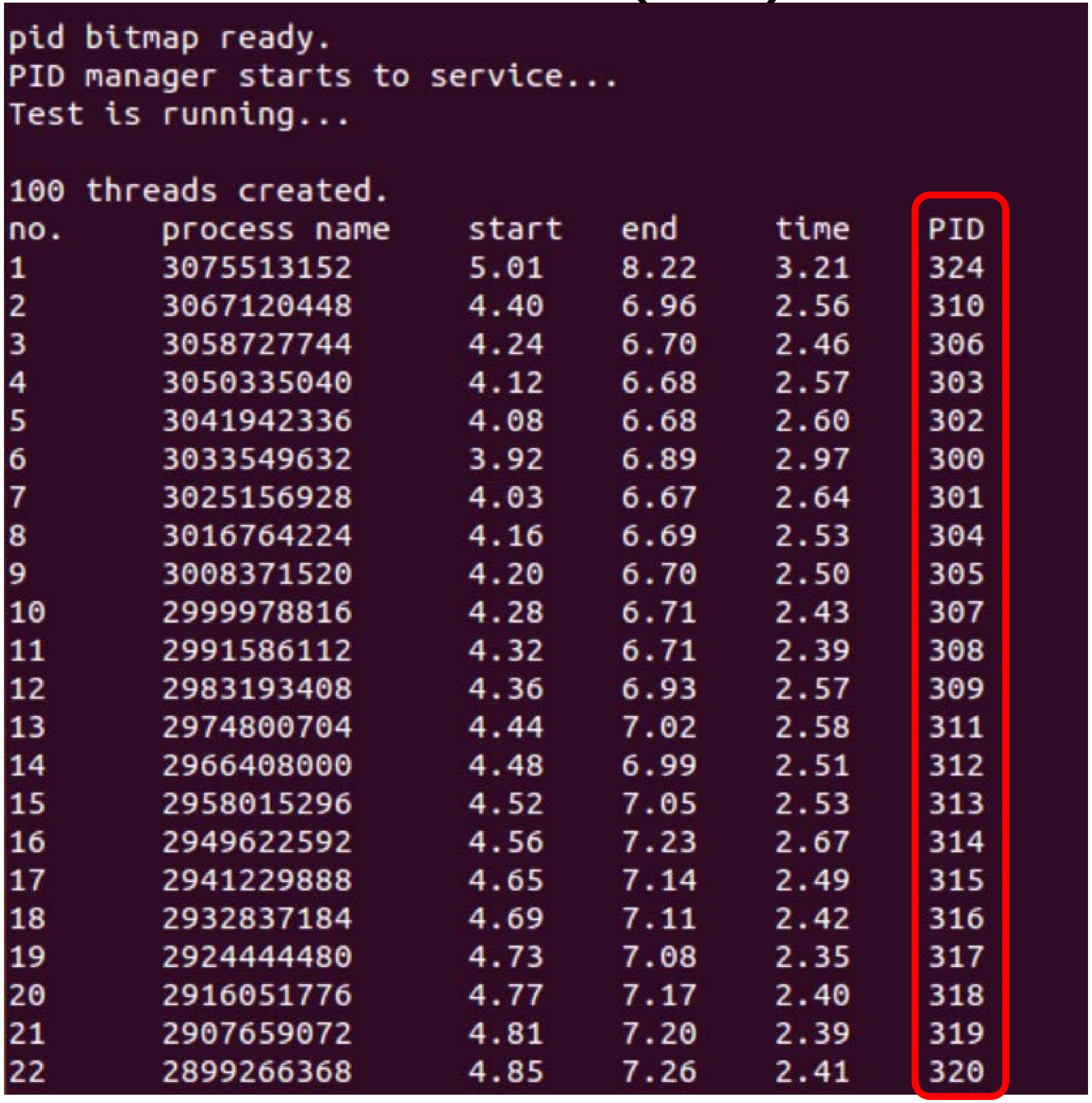
Homework Assignments #3

- Step1. Use Pthreads API to Create 100 threads.
- Step2. Use **Mutex Lock** to protect PID manager, which can allocate PID for each thread. (PID range : 300~399)
- Step3. Use **Mutex Unlock** after allocated PID.
- Step4. Let thread sleep for 1~3 seconds.
- Step5. When the thread wake up, using **Mutex Lock** to protect PID manager, which can release PID for each thread.
- Step6. Use **Mutex Unlock** after released PID.
- Step7. Terminate the thread and Destroy the mutex.
- Step8. print out the100 threads and the thread’s PID.

Mutex Lock Flowchart

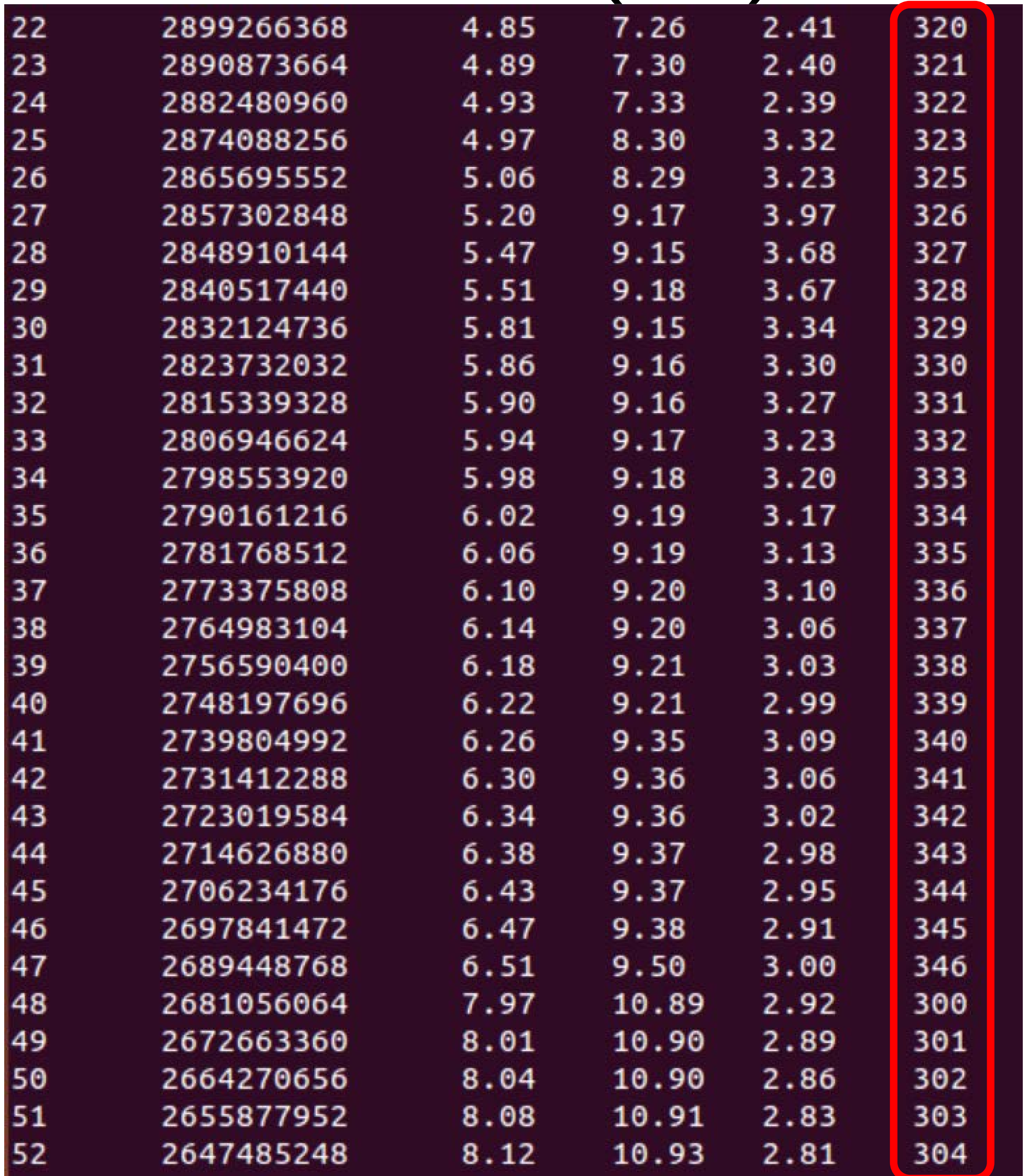


Result(1/4)



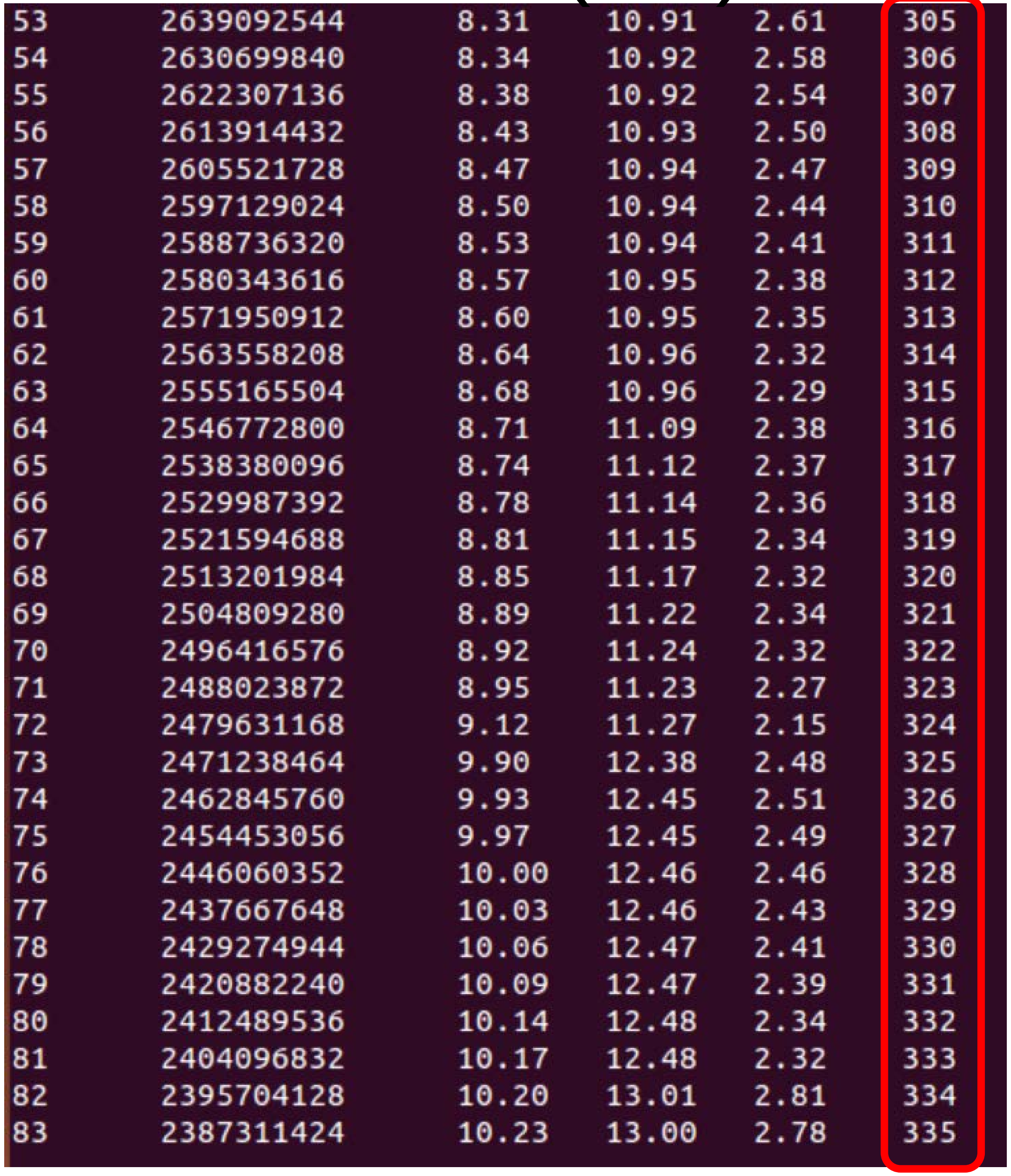
```
pid bitmap ready.
PID manager starts to service...
Test is running...

100 threads created.
no.     process name    start   end     time    PID
1       3075513152      5.01    8.22    3.21    324
2       3067120448      4.40    6.96    2.56    310
3       3058727744      4.24    6.70    2.46    306
4       3050335040      4.12    6.68    2.57    303
5       3041942336      4.08    6.68    2.60    302
6       3033549632      3.92    6.89    2.97    300
7       3025156928      4.03    6.67    2.64    301
8       3016764224      4.16    6.69    2.53    304
9       3008371520      4.20    6.70    2.50    305
10      2999978816      4.28    6.71    2.43    307
11      2991586112      4.32    6.71    2.39    308
12      2983193408      4.36    6.93    2.57    309
13      2974800704      4.44    7.02    2.58    311
14      2966408000      4.48    6.99    2.51    312
15      2958015296      4.52    7.05    2.53    313
16      2949622592      4.56    7.23    2.67    314
17      2941229888      4.65    7.14    2.49    315
18      2932837184      4.69    7.11    2.42    316
19      2924444480      4.73    7.08    2.35    317
20      2916051776      4.77    7.17    2.40    318
21      2907659072      4.81    7.20    2.39    319
22      2899266368      4.85    7.26    2.41    320
```

Result(2/4)



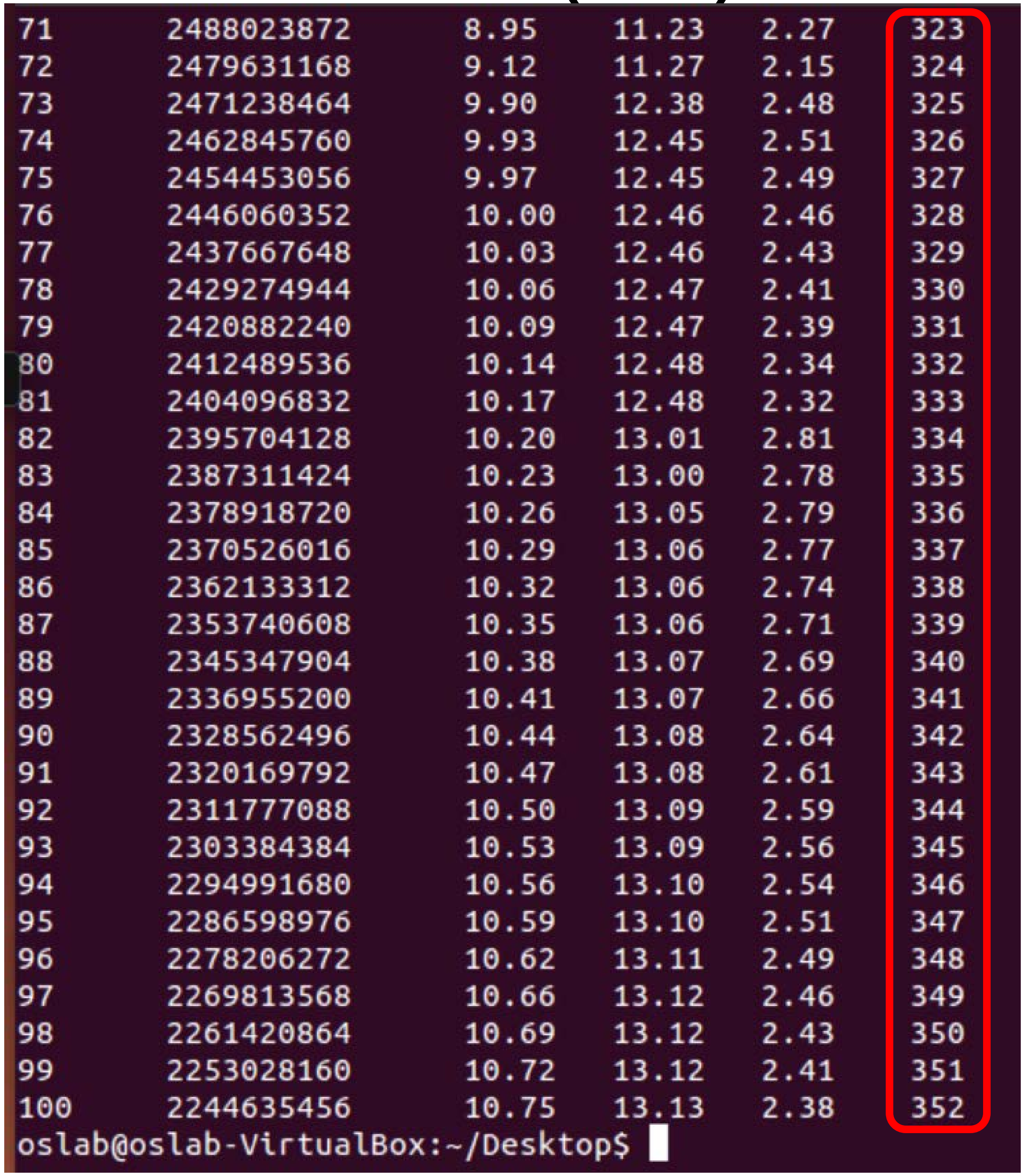
```
22        2899266368    4.85    7.26    2.41    320
23        2890873664    4.89    7.30    2.40    321
24        2882480960    4.93    7.33    2.39    322
25        2874088256    4.97    8.30    3.32    323
26        2865695552    5.06    8.29    3.23    325
27        2857302848    5.20    9.17    3.97    326
28        2848910144    5.47    9.15    3.68    327
29        2840517440    5.51    9.18    3.67    328
30        2832124736    5.81    9.15    3.34    329
31        2823732032    5.86    9.16    3.30    330
32        2815339328    5.90    9.16    3.27    331
33        2806946624    5.94    9.17    3.23    332
34        2798553920    5.98    9.18    3.20    333
35        2790161216    6.02    9.19    3.17    334
36        2781768512    6.06    9.19    3.13    335
37        2773375808    6.10    9.20    3.10    336
38        2764983104    6.14    9.20    3.06    337
39        2756590400    6.18    9.21    3.03    338
40        2748197696    6.22    9.21    2.99    339
41        2739804992    6.26    9.35    3.09    340
42        2731412288    6.30    9.36    3.06    341
43        2723019584    6.34    9.36    3.02    342
44        2714626880    6.38    9.37    2.98    343
45        2706234176    6.43    9.37    2.95    344
46        2697841472    6.47    9.38    2.91    345
47        2689448768    6.51    9.50    3.00    346
48        2681056064    7.97    10.89   2.92    300
49        2672663360    8.01    10.90   2.89    301
50        2664270656    8.04    10.90   2.86    302
51        2655877952    8.08    10.91   2.83    303
52        2647485248    8.12    10.93   2.81    304
```

Result(3/4)



```
53       2639092544      8.31     10.91    2.61     305
54       2630699840      8.34     10.92    2.58     306
55       2622307136      8.38     10.92    2.54     307
56       2613914432      8.43     10.93    2.50     308
57       2605521728      8.47     10.94    2.47     309
58       2597129024      8.50     10.94    2.44     310
59       2588736320      8.53     10.94    2.41     311
60       2580343616      8.57     10.95    2.38     312
61       2571950912      8.60     10.95    2.35     313
62       2563558208      8.64     10.96    2.32     314
63       2555165504      8.68     10.96    2.29     315
64       2546772800      8.71     11.09    2.38     316
65       2538380096      8.74     11.12    2.37     317
66       2529987392      8.78     11.14    2.36     318
67       2521594688      8.81     11.15    2.34     319
68       2513201984      8.85     11.17    2.32     320
69       2504809280      8.89     11.22    2.34     321
70       2496416576      8.92     11.24    2.32     322
71       2488023872      8.95     11.23    2.27     323
72       2479631168      9.12     11.27    2.15     324
73       2471238464      9.90     12.38    2.48     325
74       2462845760      9.93     12.45    2.51     326
75       2454453056      9.97     12.45    2.49     327
76       2446060352      10.00    12.46    2.46     328
77       2437667648      10.03    12.46    2.43     329
78       2429274944      10.06    12.47    2.41     330
79       2420882240      10.09    12.47    2.39     331
80       2412489536      10.14    12.48    2.34     332
81       2404096832      10.17    12.48    2.32     333
82       2395704128      10.20    13.01    2.81     334
83       2387311424      10.23    13.00    2.78     335
```

Result(4/4)



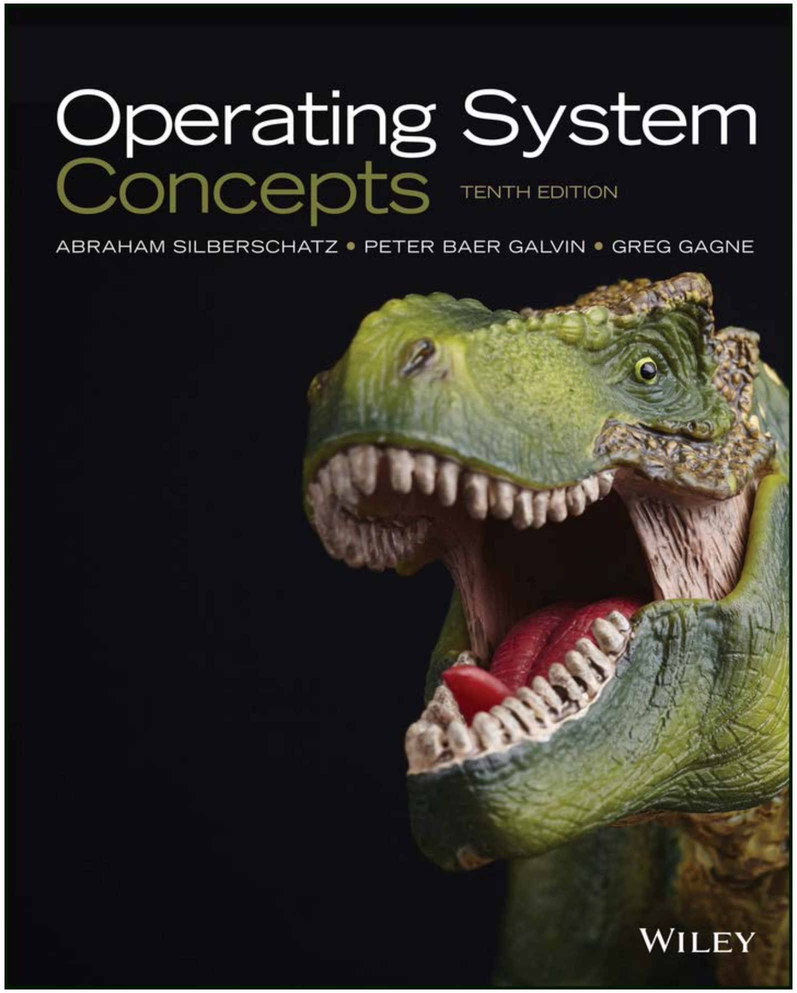
```
71          2488023872        8.95      11.23     2.27      323
72          2479631168        9.12      11.27     2.15      324
73          2471238464        9.90      12.38     2.48      325
74          2462845760        9.93      12.45     2.51      326
75          2454453056        9.97      12.45     2.49      327
76          2446060352        10.00     12.46     2.46      328
77          2437667648        10.03     12.46     2.43      329
78          2429274944        10.06     12.47     2.41      330
79          2420882240        10.09     12.47     2.39      331
80          2412489536        10.14     12.48     2.34      332
81          2404096832        10.17     12.48     2.32      333
82          2395704128        10.20     13.01     2.81      334
83          2387311424        10.23     13.00     2.78      335
84          2378918720        10.26     13.05     2.79      336
85          2370526016        10.29     13.06     2.77      337
86          2362133312        10.32     13.06     2.74      338
87          2353740608        10.35     13.06     2.71      339
88          2345347904        10.38     13.07     2.69      340
89          2336955200        10.41     13.07     2.66      341
90          2328562496        10.44     13.08     2.64      342
91          2320169792        10.47     13.08     2.61      343
92          2311777088        10.50     13.09     2.59      344
93          2303384384        10.53     13.09     2.56      345
94          2294991680        10.56     13.10     2.54      346
95          2286598976        10.59     13.10     2.51      347
96          2278206272        10.62     13.11     2.49      348
97          2269813568        10.66     13.12     2.46      349
98          2261420864        10.69     13.12     2.43      350
99          2253028160        10.72     13.12     2.41      351
100         2244635456        10.75     13.13     2.38      352
oslab@oslab-VirtualBox:~/Desktop$ 
```

Outline

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

Reference

- Operating System Concepts, 10th Edition



Turn in

- Deadline

  2019/12/26 PM.11:59
- Upload to iLearning
- File name
  - HW3_ID.zip (e.g. HW3_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：Yun-Jen, Lee
- Email：yunjen.lee@gmail.com
  - Title format : OS HW#3 - [your name]
- Lab: OSNET(1001)