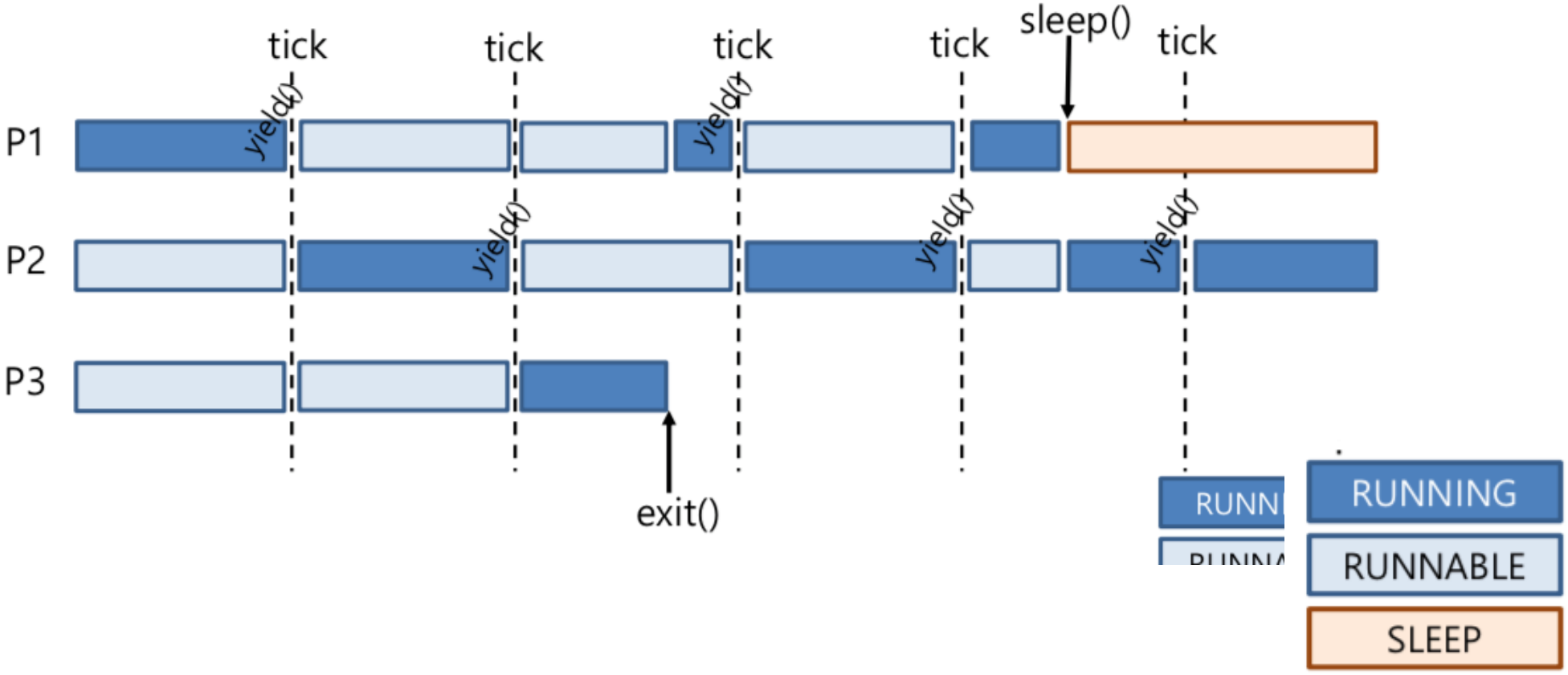
Operating Systems Practice

5: Synchronization

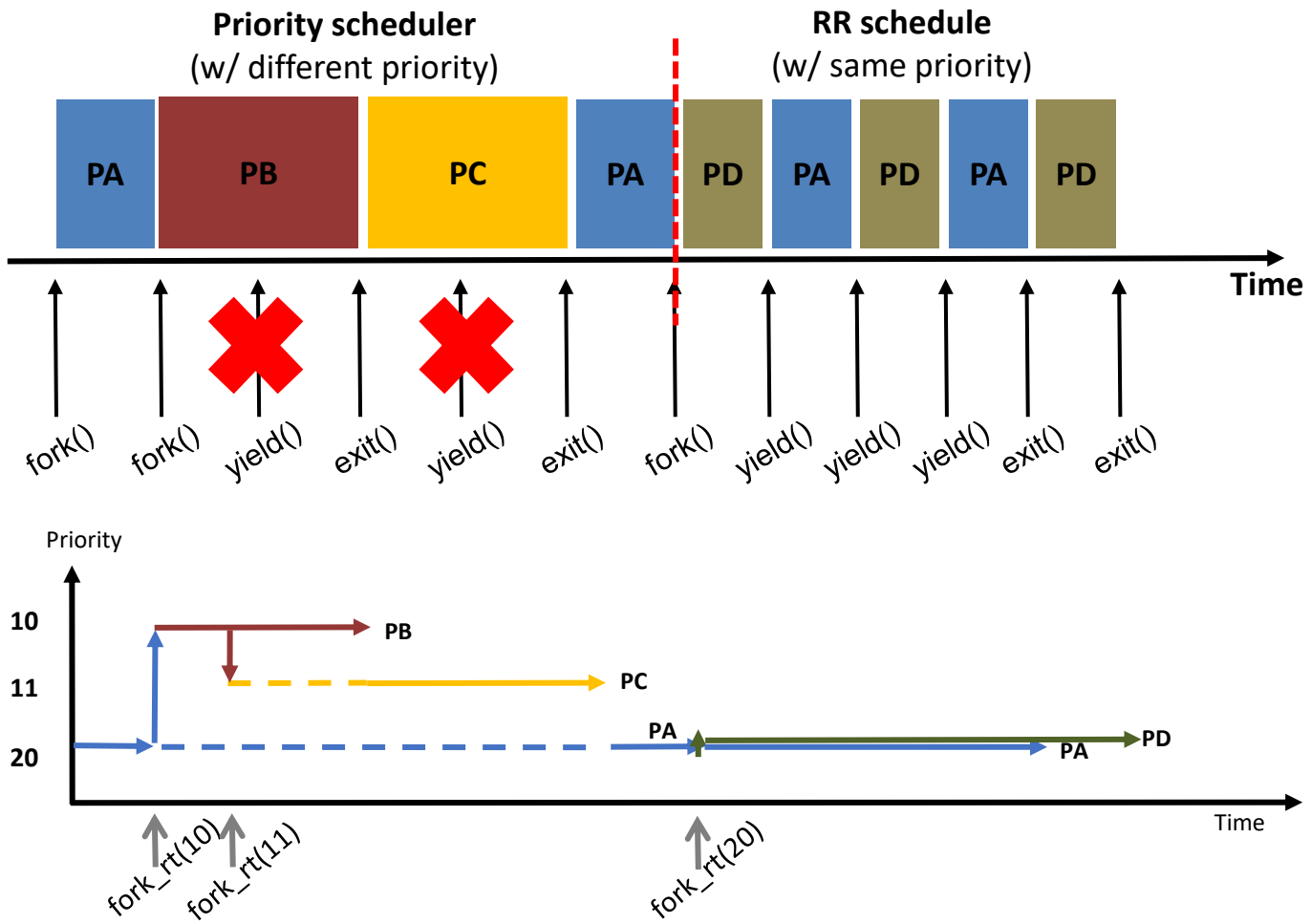
Kilho Lee

Revisit: xv6 Scheduler (RR)

- Timer's interrupt request (IRQ) enforces an yield of a CPU
- A "RUNNABLE" process is chosen to be run in a round-robin manner



Prerequisite: priority-based scheduler



Prerequisite: priority-based scheduler

I. Rules of the priority-based scheduler.

- Each process will be forked with a new system call "fork_rt ()" to designate a given priority.
- Basic rule: the lower the priority value, the higher the priority.
- Thereby, the scheduler should run a process which has the lowest priority value.
- If two or more processes has the identical priority value, then schedule them in a round-robin manner.
  - RR time quanta – one tick, in other words, switch the process every single timer interrupt.

Project 4. Synchronization

Implementation goals:

1. priority-based scheduler
2. semaphore
3. priority inheritance protocol

- New files
  - syslock.c, semaphore.h, semaphore.c

- Note
  - Do not modify test code
  - Based on the priority-based scheduler

Project 4. Synchronization

For the priority-based scheduler:

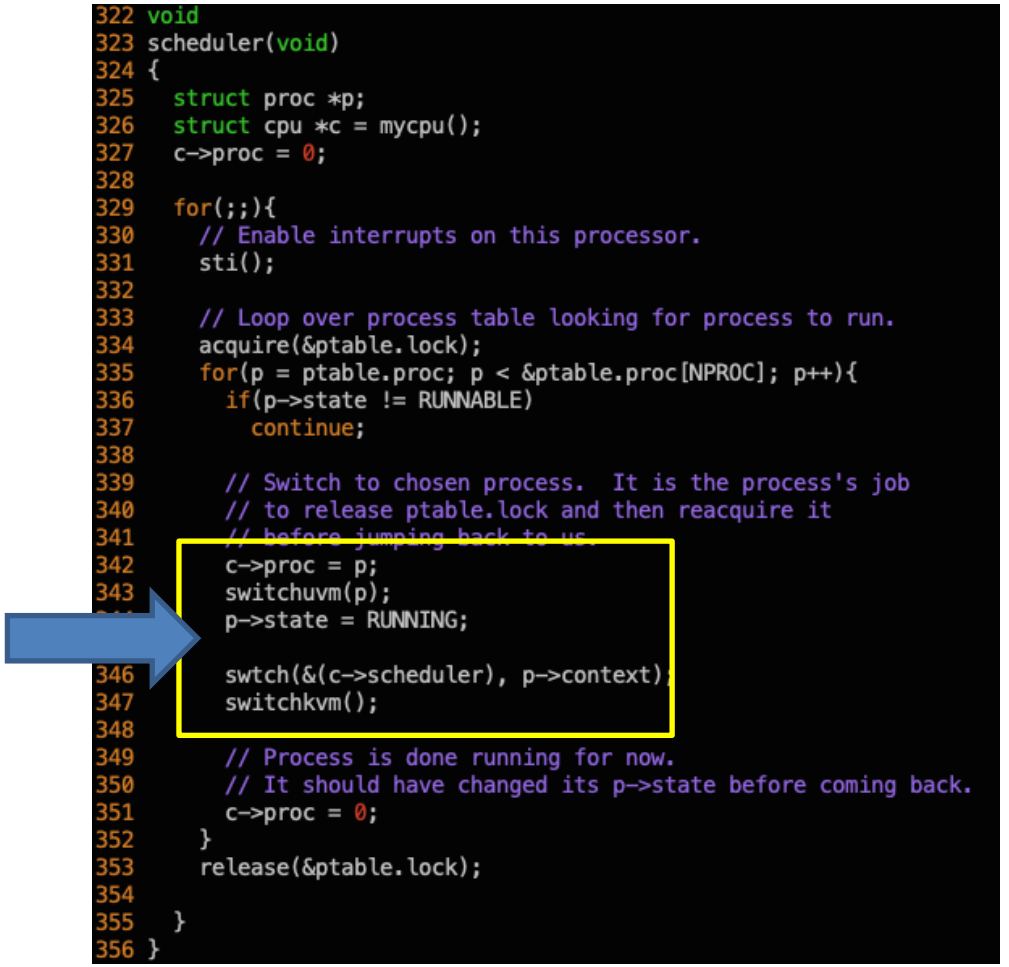
- proc.h
  - An additional variable for the priority value of each process.

- proc.c
  - Implement that: schedules a proper process according to the priority value (choose the process with the lowest priority value).

proc.h

```
35 enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
36
37 // Per-process state
38 struct proc {
39   uint sz;                     // Size of process memory (bytes)
40   pde_t* pgdir;                // Page table
```

proc.c



Project 4. Synchronization

- syslock.c
  - pre-allocate NLOCK(10) of *struct semaphore*.
  - When initializing a semaphore, an unused semaphore from the pool of pre-allocated semaphore is found and assigned for use.
  - Since this code is only as interface, the actual implementation of the semaphore should be done in '*semaphore.c*' and '*semaphore.h*'

```
17 int
18 sys_sem_init(void)
19 {
20   char *lock_ptr;
21   int init_value;
22   if(argptr(0, &lock_ptr, sizeof(int)) < 0 || a
23     return -1;
24   int *lock_value = (int *)lock_ptr;
25   int lock_idx = find_empty_lock();
26   if(lock_idx != -1){
27     dict[lock_idx] = lock_value;
28   }
29   else{
30     cprintf("no lock available\n");
31     return -1;
32   }
33   sem_init(&usema[lock_idx], init_value);
34   return 0;
35 }
36
37 int
38 sys_sem_wait(void)
39 {
40   char *lock_ptr;
41   if(argptr(0, &lock_ptr, sizeof(int)) < 0)
42     return -1;
43   int *lock_value = (int *)lock_ptr;
44   int lock_idx = search_lock(lock_value);
45   if(lock_idx == -1)
46     return -1;
47   sem_wait(&usema[lock_idx]);
48   return 0;
49 }
```

```
51 int
52 sys_sem_post(void)
53 {
54   char *lock_ptr;
55   if(argptr(0, &lock_ptr, sizeof(int)) < 0)
56     return -1;
57   int *lock_value = (int *)lock_ptr;
58   int lock_idx = search_lock(lock_value);
59   if(lock_idx == -1)
60     return -1;
61   sem_post(&usema[lock_idx]);
62   return 0;
63 }
64
65 int
66 sys_sem_destroy(void)
67 {
68   char *lock_ptr;
69   if(argptr(0, &lock_ptr, sizeof(int)) < 0)
70     return -1;
71   int *lock_value = (int *)lock_ptr;
72   int lock_idx = search_lock(lock_value);
73   if(lock_idx == -1)
74     return -1;
75   sem_destroy(&usema[lock_idx]);
76   dict[lock_idx] = 0;
77   return 0;
78 }
```

The files *syscall.c, syscall.h, sylock.c, user.h and usys.S* have been modified to add new system calls.

This interface will be given, so you don't have to modify these files.

Project 4. Synchronization

- semaphore.h (YOUR TARGET)
  - You are free to implement this structure as needed, depending on the required members.
  - According to your own *struct semaphore*, you can proceed to implement the semaphore on *'semaphore.c'*

semaphore.h+

```
1  struct semaphore {
2
3    /* ********************** */
4    /* * WRITE YOUR CODE     */
5    /* ********************** */
6
7  };
8
9  extern struct semaphore usema[NLOCK];
10
```

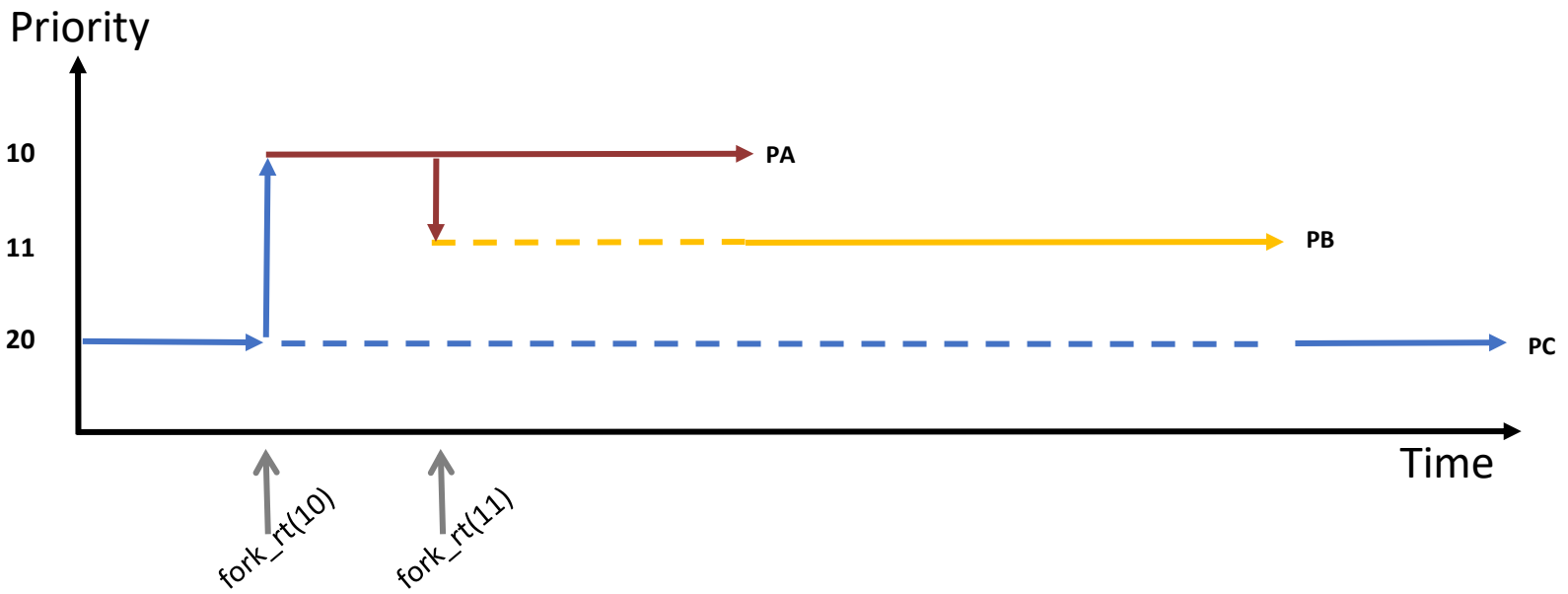
Project 4. Synchronization

- semaphore.c (YOUR TARGET)
  - void sem_init(struct semaphore *s, int init_value)
    - Initialization function
    - Init your own struct semaphore and initialize it to the value *init_value*
    - The semaphore is shared between all processes.
  - void sem_wait(struct semaphore *s)
    - Decrement the value of *semaphore s* by one, wait if value of *semaphore s* is negative
  - void sem_post(struct semaphore *s)
    - Increment the value of *semaphore s* by one, if there are one or more processes waiting, wake one
  - void sem_destroy(struct semaphore *s)
    - Free function
    - Free your own struct semaphore

```
11 void
12 sem_init(struct semaphore *s, int init_value)
13 {
14   /* ********************* */
15   /* * WRITE YOUR CODE    */
16   /* ********************* */
17 }
18
19 void
20 sem_wait(struct semaphore *s)
21 {
22   /* ********************* */
23   /* * WRITE YOUR CODE    */
24   /* ********************* */
25 }
26
27 void
28 sem_post(struct semaphore *s)
29 {
30   /* ********************* */
31   /* * WRITE YOUR CODE    */
32   /* ********************* */
33 }
34
35 void
36 sem_destroy(struct semaphore *s)
37 {
38   /* ********************* */
39   /* * WRITE YOUR CODE    */
40   /* ********************* */
41 }
```

test_rt.c

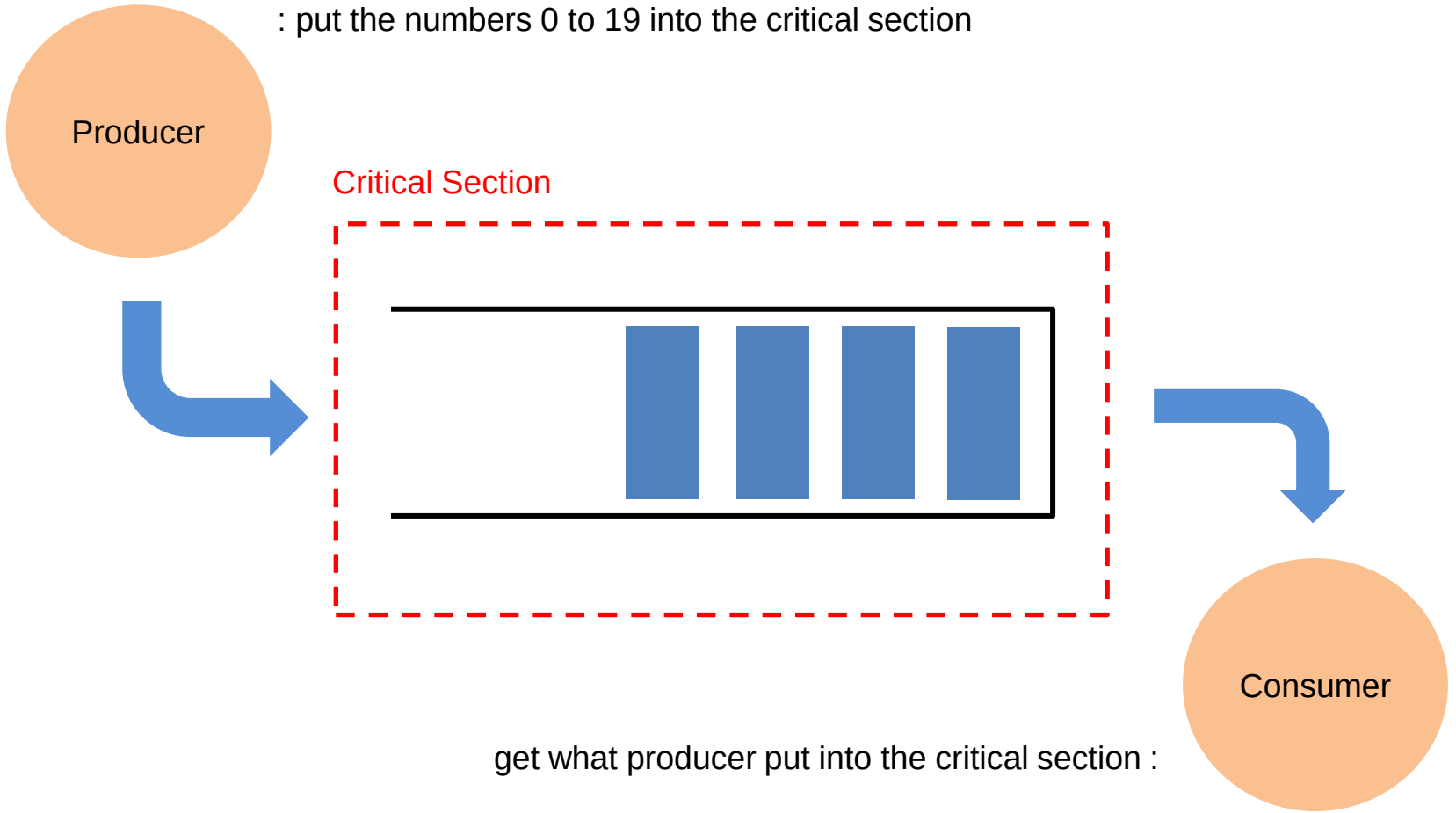
- Use test_rt.c as a user program
- The output is deterministic; it should be identical to the figure
- ./test_rt




```
$ test_rt
=== TEST START ===
PA, i = 0, dummy = 20000000
PA, i = 1, dummy = E0000000
PA, i = 2, dummy = 60000000
PA, i = 3, dummy = E0000000
PA, i = 4, dummy = 60000000
PA, i = 5, dummy = 0
PA, i = 6, dummy = 80000000
PA, i = 7, dummy = 0
PA, i = 8, dummy = 80000000
PA, i = 9, dummy = 0
PA, i = 0, dummy = 20000000
PA, i = 1, dummy = 60000000
PA, i = 2, dummy = A0000000
PA, i = 3, dummy = E0000000
PA, i = 4, dummy = 20000000
PA, i = 5, dummy = 60000000
PA, i = 6, dummy = A0000000
PA, i = 7, dummy = E0000000
PA, i = 8, dummy = 20000000
PA, i = 9, dummy = 60000000
PB, i = 0, dummy = 20000000
PB, i = 1, dummy = 60000000
PB, i = 2, dummy = A0000000
PB, i = 3, dummy = E0000000
PB, i = 4, dummy = 20000000
PB, i = 5, dummy = 60000000
PB, i = 6, dummy = A0000000
PB, i = 7, dummy = E0000000
PB, i = 8, dummy = 20000000
PB, i = 9, dummy = 60000000
PC, i = 0, dummy = 20000000
PC, i = 1, dummy = E0000000
PC, i = 2, dummy = 60000000
PC, i = 3, dummy = E0000000
PC, i = 4, dummy = 60000000
PC, i = 5, dummy = 0
PC, i = 6, dummy = 80000000
PC, i = 7, dummy = 0
PC, i = 8, dummy = 80000000
PC, i = 9, dummy = 0
PC, i = 10, dummy = 20000000
PC, i = 11, dummy = 60000000
PC, i = 12, dummy = A0000000
PC, i = 13, dummy = E0000000
PC, i = 14, dummy = 20000000
PC, i = 15, dummy = 60000000
PC, i = 16, dummy = A0000000
PC, i = 17, dummy = E0000000
PC, i = 18, dummy = 20000000
PC, i = 19, dummy = 60000000
=== TEST DONE ===
```

test_pc.c

: put the numbers 0 to 19 into the critical section



get what producer put into the critical section :

```
$ test_pc
=== TEST START ===
Producer puts 0
Consumer gets 0
Producer puts 1
Producer puts 2
Consumer gets 1
Producer puts 3
Producer puts 4
Consumer gets 2
Producer puts 5
Consumer gets 3
Producer puts 6
Consumer gets 4
Consumer gets 5
Producer puts 7
Consumer gets 6
Producer puts 8
Producer puts 9
Consumer gets 7
Producer puts 10
Consumer gets 8
Producer puts 11
Consumer gets 9
Consumer gets 10
Producer puts 12
Producer puts 13
Consumer gets 11
Producer puts 14
Producer puts 15
Consumer gets 12
Producer puts 16
Consumer gets 13
Producer puts 17
Consumer gets 14
Producer puts 18
Consumer gets 15
Consumer gets 16
Producer puts 19
Consumer gets 17
Consumer gets 18
Consumer gets 19
=== TEST DONE ===
```

Note: The producer and the consumer processes have the same priority, they will run in a RR manner.
Since the execution environment varies depending on each PC, the results may differ.
But, after producer puts into the critical section, consumer can get the number.
And consumer must get the numbers 0 to 19.

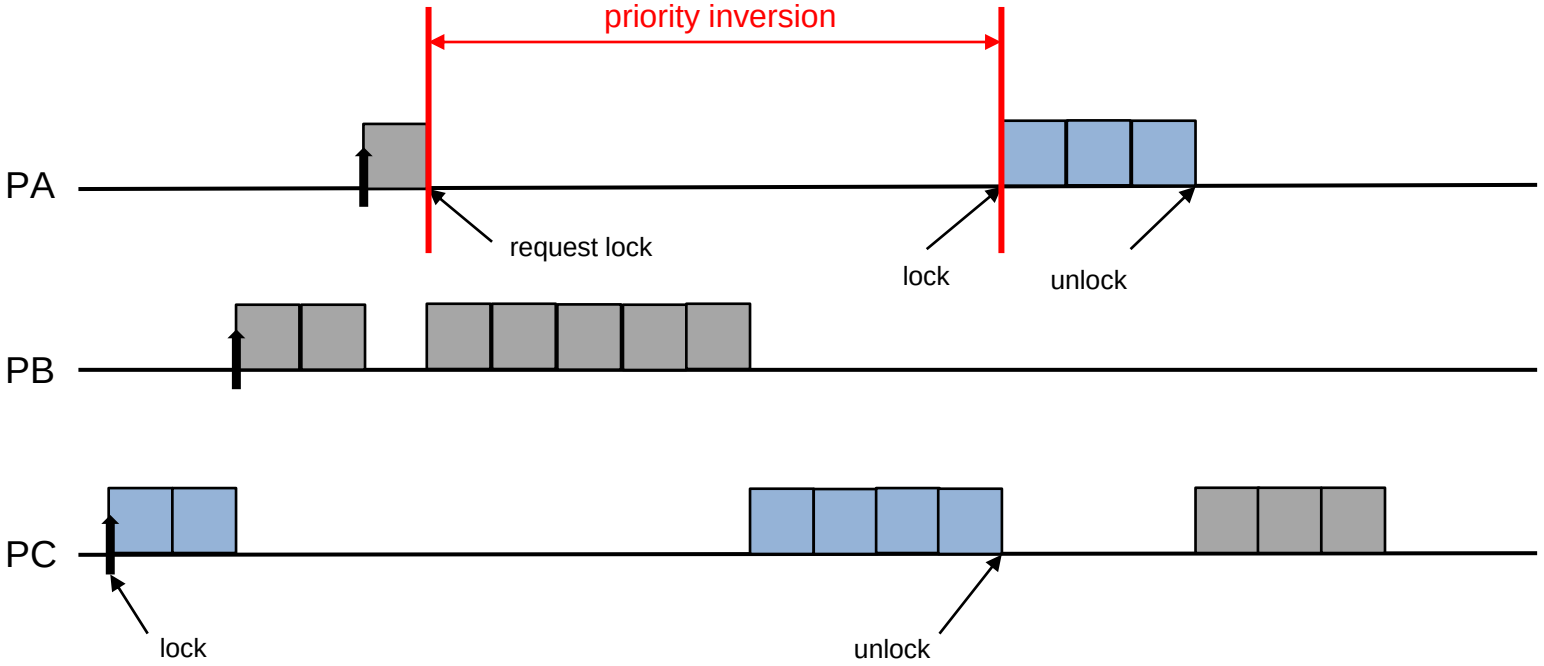
test_pi.c

Before PIP (Priority Inheritance Protocol)

PA, PB, PC : real-time process
Priority : PA > PB > PC

: normal execution

: execution with critical section



NOTE: If you make this result (without PIP), you will get a partial credit.

```
$ test_pi
=== TEST START ===
PC holds the lock
PC, i = 0, dummy = C0000000
PC, i = 1, dummy = A0000000
PC, i = 2, dummy = C0000000
PC, i = 3, dummy = 0
PC, i = 4, dummy = 80000000
PB, i = 0, dummy = C0000000
PB, i = 1, dummy = A0000000
PB, i = 2, dummy = C0000000
PB, i = 3, dummy = 0
PB, i = 4, dummy = 80000000
PA, i = 0, dummy = C0000000
PA, i = 1, dummy = A0000000
PA tries to hold the lock
PB, i = 5, dummy = 0
PB, i = 6, dummy = 80000000
PB, i = 7, dummy = 0
PB, i = 8, dummy = 0
PB, i = 9, dummy = 0
PC, i = 5, dummy = 0
PC, i = 6, dummy = 80000000
PC, i = 7, dummy = 0
PC releases the lock
PA holds the lock
PA, i = 2, dummy = C0000000
PA, i = 3, dummy = 0
PA, i = 4, dummy = 80000000
PA, i = 5, dummy = 0
PA, i = 6, dummy = 80000000
PA, i = 7, dummy = 0
PA, i = 8, dummy = 0
PA, i = 9, dummy = 0
PA releases the lock
PC, i = 8, dummy = 0
PC, i = 9, dummy = 0
```

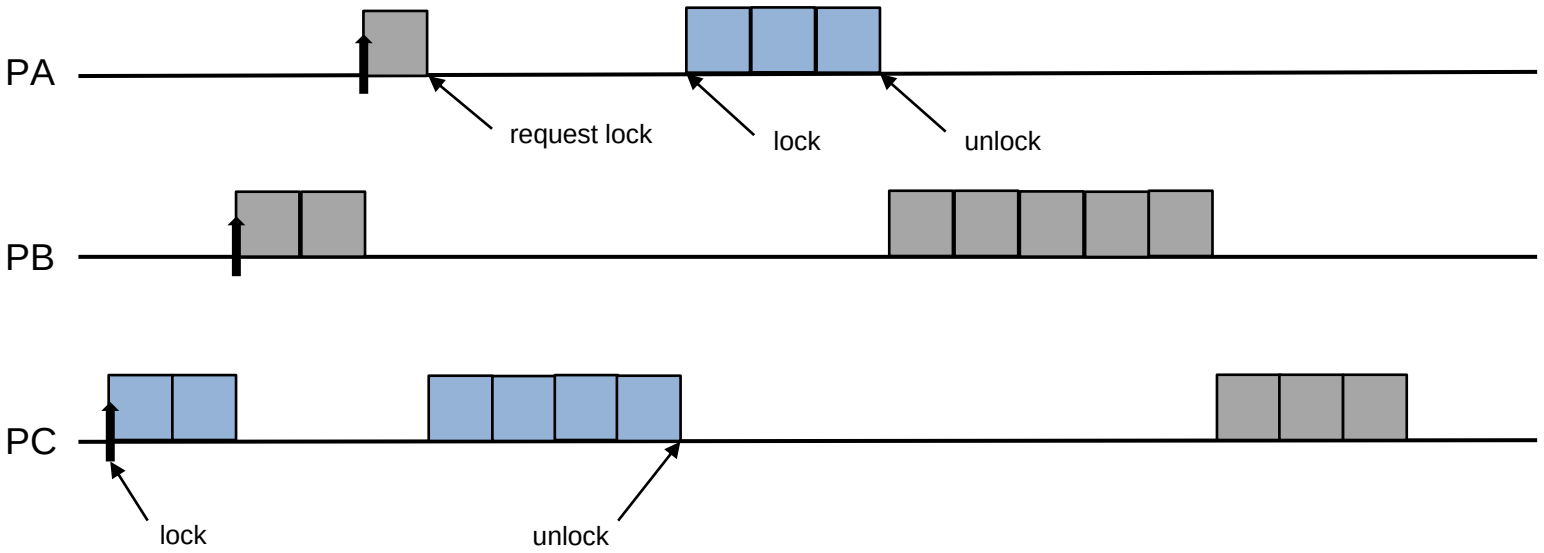
test_pi.c

After PIP (Priority Inheritance Protocol)

PA, PB, PC : real-time process
Priority : PA > PB > PC

: normal execution

: execution with critical section



NOTE: If you make this result (PIP), you will get a full credit.

```
$ test_pi
=== TEST START ===
PC holds the lock
PC, i = 0, dummy = C0000000
PC, i = 1, dummy = A0000000
PC, i = 2, dummy = C0000000
PC, i = 3, dummy = 0
PC, i = 4, dummy = 80000000
PB, i = 0, dummy = C0000000
PB, i = 1, dummy = A0000000
PB, i = 2, dummy = C0000000
PB, i = 3, dummy = 0
PB, i = 4, dummy = 80000000
PA, i = 0, dummy = C0000000
PA, i = 1, dummy = A0000000
PA tries to hold the lock
PC, i = 5, dummy = 0
PC, i = 6, dummy = 80000000
PC, i = 7, dummy = 0
PC releases the lock
PA holds the lock
PA, i = 2, dummy = C0000000
PA, i = 3, dummy = 0
PA, i = 4, dummy = 80000000
PA, i = 5, dummy = 0
PA, i = 6, dummy = 80000000
PA, i = 7, dummy = 0
PA, i = 8, dummy = 0
PA, i = 9, dummy = 0
PA releases the lock
PB, i = 5, dummy = 0
PB, i = 6, dummy = 80000000
PB, i = 7, dummy = 0
PB, i = 8, dummy = 0
PB, i = 9, dummy = 0
PC, i = 8, dummy = 0
PC, i = 9, dummy = 0
```

Hand-in Procedures (1/2)

- Download template
  - https://github.com/KilhoLee/xv6-ssu.git (pull or clone)
  - `tar xvzf xv6_ssu_sync.tar.gz`

- Add test_*.c to your codes and modify Makefile properly
  - `test_pc.c, test_pi.c`

- Build with CPUS=1 flag
  - Makefile

```
ifndef CPUS
CPUS := 1
endif
```

Hand-in Procedures (2/2)

- Compress your code (ID: 20221234)
  - `$tar cvzf xv6_ssu_sync_20221234.tar.gz xv6_ssu_sync`
  - Please command `$make clean` before compressing

- Submit your `tar.gz` file through class.ssu.ac.kr