Systems Programming

Spring 2023

12주차

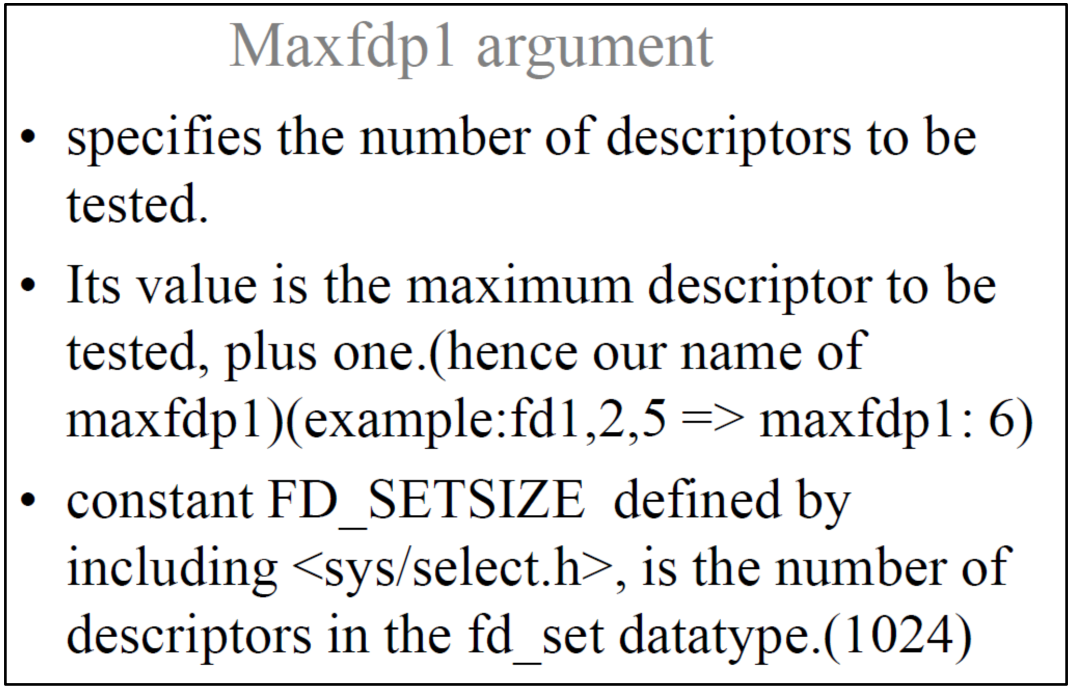
- Q&A
  - recap select()

  - 16-sync-basic
  - 17-sync-advanced
  - 17-sync-supplement

recap select()

Questions

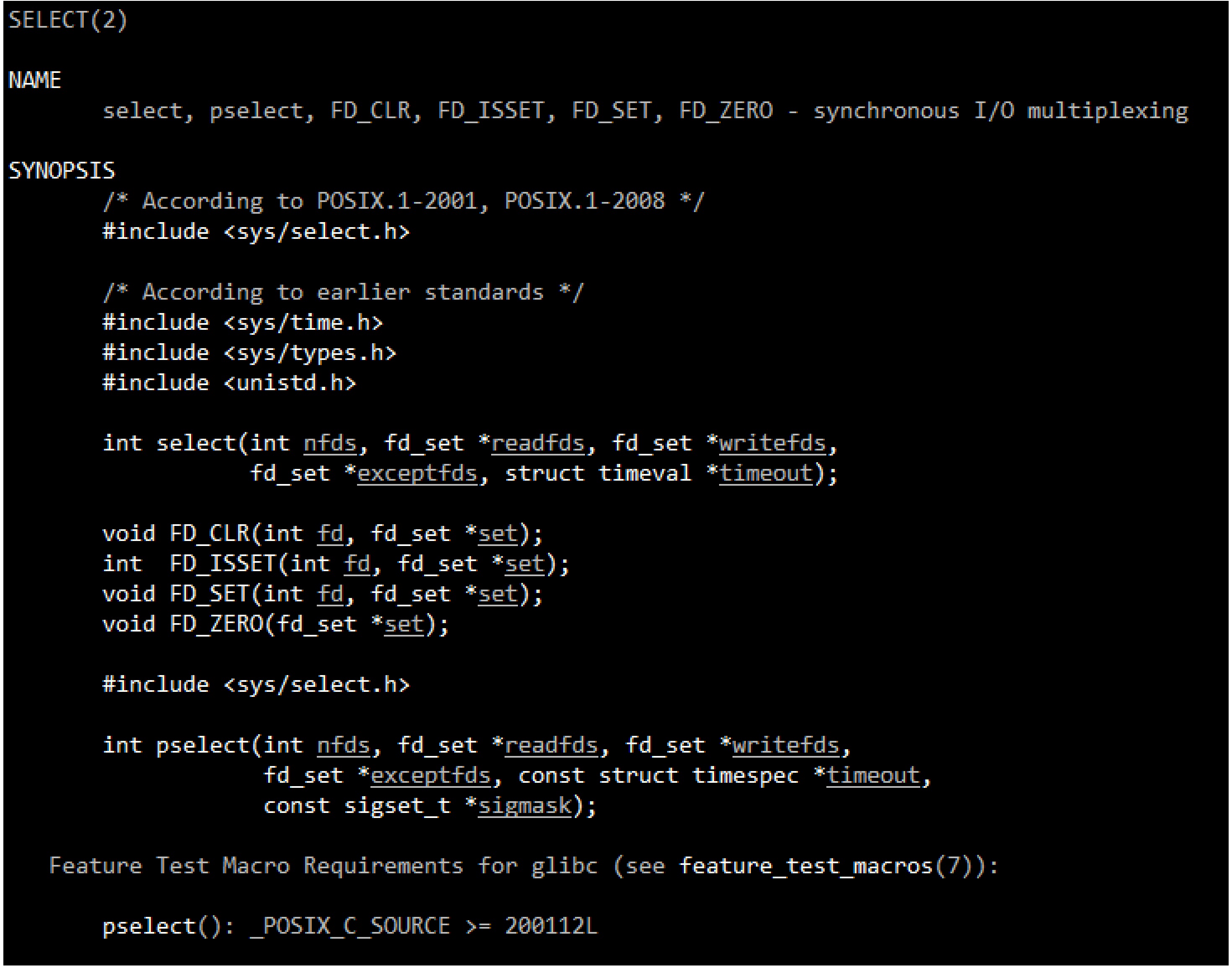
22p에 보면 "• constant FD_SETSIZE defined by including <sys/select.h>, is the number of descriptors in the fd_set datatype.(1024)" 라는 내용이 나오는데 이 상수의 사용처를 모르겠습니다. 그냥 maxfdp1값이 FD_SETSIZE 보다 작아야 하므로 select함수를 호출할때 주의해라 이정도로 받아들이면 될까요?



Maxfdp1 argument

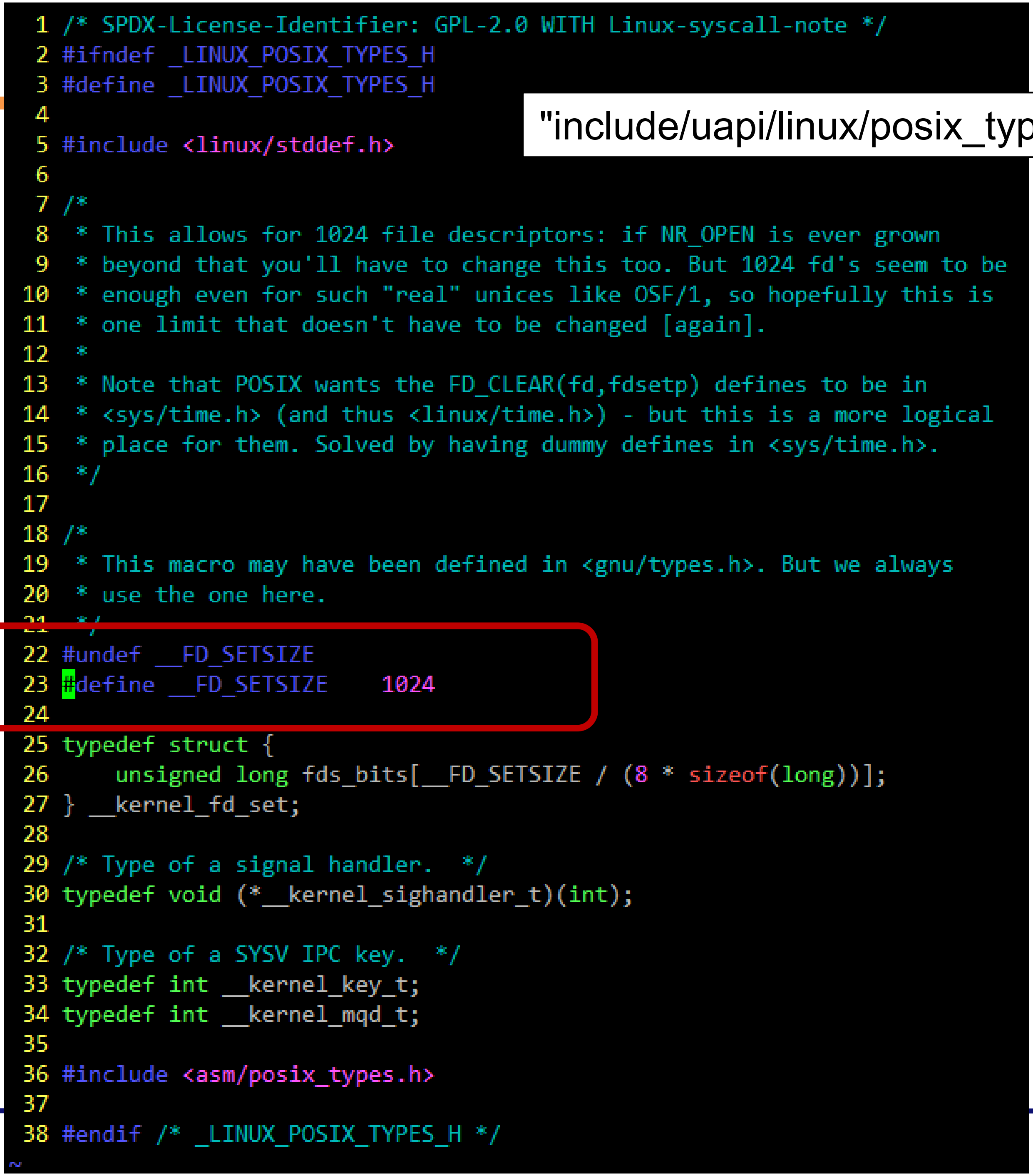
- specifies the number of descriptors to be tested.
- Its value is the maximum descriptor to be tested, plus one.(hence our name of maxfdp1)(example:fd1,2,5 => maxfdp1: 6)
- constant FD_SETSIZE defined by including <sys/select.h>, is the number of descriptors in the fd_set datatype.(1024)

Questions



```
SELECT(2)

NAME
       select, pselect, FD_CLR, FD_ISSET, FD_SET, FD_ZERO - synchronous I/O multiplexing

SYNOPSIS
       /* According to POSIX.1-2001, POSIX.1-2008 */
       #include <sys/select.h>

       /* According to earlier standards */
       #include <sys/time.h>
       #include <sys/types.h>
       #include <unistd.h>

       int select(int nfds, fd_set *readfds, fd_set *writefds,
                  fd_set *exceptfds, struct timeval *timeout);

       void FD_CLR(int fd, fd_set *set);
       int  FD_ISSET(int fd, fd_set *set);
       void FD_SET(int fd, fd_set *set);
       void FD_ZERO(fd_set *set);

       #include <sys/select.h>

       int pselect(int nfds, fd_set *readfds, fd_set *writefds,
                   fd_set *exceptfds, const struct timespec *timeout,
                   const sigset_t *sigmask);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       pselect(): _POSIX_C_SOURCE >= 200112L
```

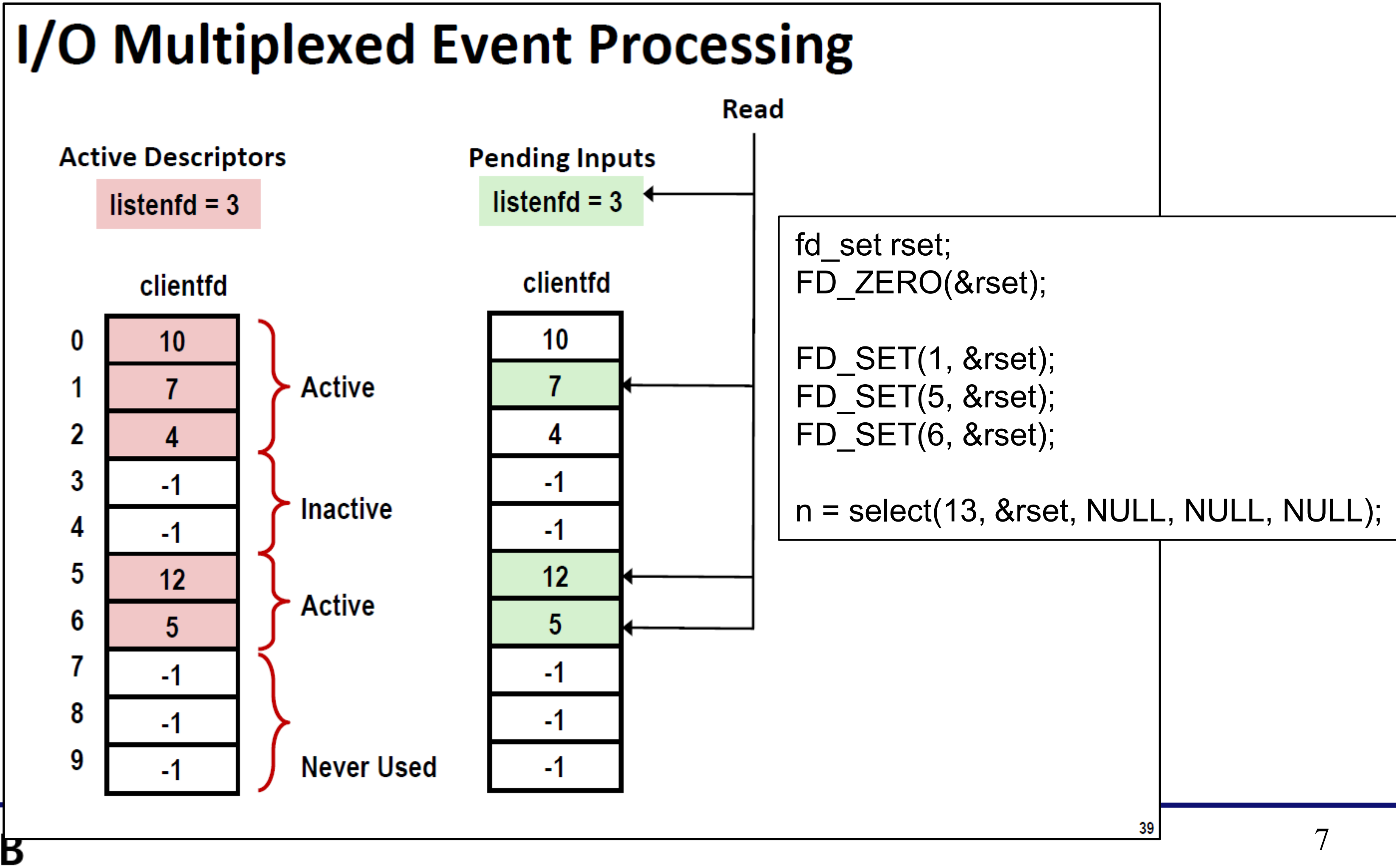
"include/uapi/linux/posix_types.h"



```c
/* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note */
#ifndef _LINUX_POSIX_TYPES_H
#define _LINUX_POSIX_TYPES_H

#include <linux/stddef.h>

/*
 * This allows for 1024 file descriptors: if NR_OPEN is ever grown
 * beyond that you'll have to change this too. But 1024 fd's seem to be
 * enough even for such "real" unices like OSF/1, so hopefully this is
 * one limit that doesn't have to be changed [again].
 *
 * Note that POSIX wants the FD_CLEAR(fd,fdsetp) defines to be in
 * <sys/time.h> (and thus <linux/time.h>) - but this is a more logical
 * place for them. Solved by having dummy defines in <sys/time.h>.
 */

/*
 * This macro may have been defined in <gnu/types.h>. But we always
 * use the one here.
 */
#undef __FD_SETSIZE
#define __FD_SETSIZE    1024

typedef struct {
    unsigned long fds_bits[__FD_SETSIZE / (8 * sizeof(long))];
} __kernel_fd_set;

/* Type of a signal handler.  */
typedef void (*__kernel_sighandler_t)(int);

/* Type of a SYSV IPC key.  */
typedef int __kernel_key_t;
typedef int __kernel_mqd_t;

#include <asm/posix_types.h>

#endif /* _LINUX_POSIX_TYPES_H */
~
```

Questions

39p에 그림을 강의를 들으며 listenfd는 3, 그리고 각 배열에 있는 번호들 (4~12)는 clientfd라고 이해했습니다. 하지만 그림에 나와있는 배열(0, 1, 2 ... 9번까지 indexing된) 이 무엇을 나타내는지 모르겠으며 -1로 비어있는 공간도 정확히 왜 존재하는지 이해하지 못했고 왜 connfd가 아닌 clientfd인지도 잘 모르겠습니다.



I/O Multiplexed Event Processing

	Active Descriptors clientfd	Pending Inputs clientfd
0	10	10
1	7	7
2	4	4
3	-1	-1
4	-1	-1
5	12	12
6	5	5
7	-1	-1
8	-1	-1
9	-1	-1

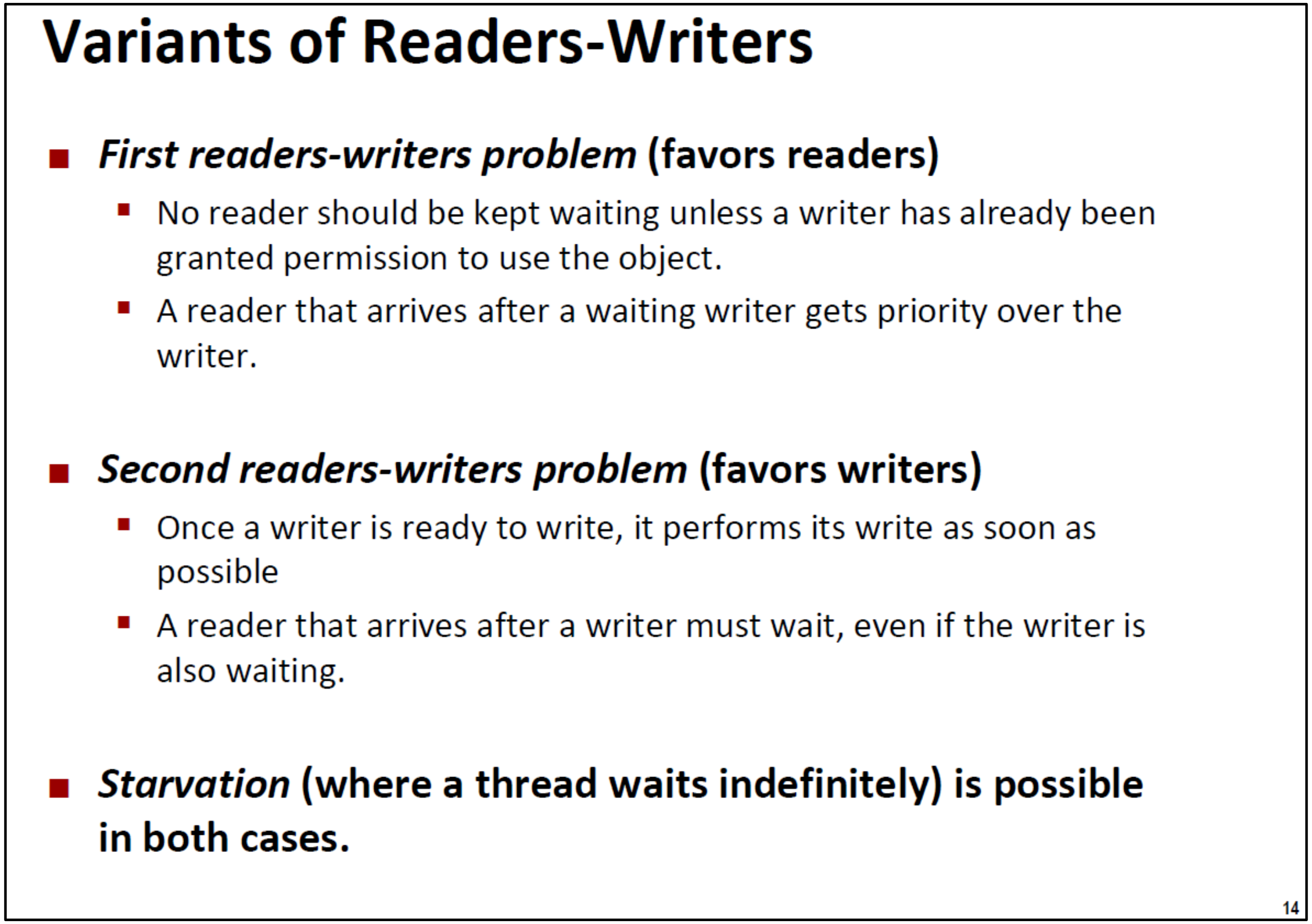
```
fd_set rset;
FD_ZERO(&rset);

FD_SET(1, &rset);
FD_SET(5, &rset);
FD_SET(6, &rset);

n = select(13, &rset, NULL, NULL, NULL);
```

16-sync-basic

17-sync-advanced

Questions

14p Second readers-writers problem (favors writers)부분을 정확하게 이해하지 못한 것 같아 질문드립니다.
예를 들어 (현재 write 진행중) -> (R) -> (W) 이와 같은 순서로 도착했다 했을 때 write를 favor한다는 것은 (W) 실행 이후 (R)이 실행된다는 의미인지, 아니면 (W)보다 (R)이 먼저왔으니 (R)을 먼저 실행시켜주는지 궁금합니다.



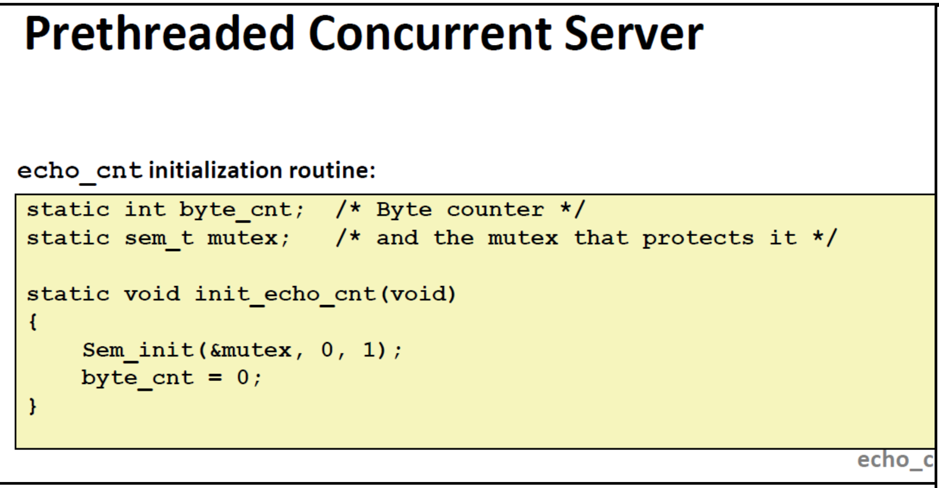
Variants of Readers-Writers

- ***First readers-writers problem*** **(favors readers)**
  - No reader should be kept waiting unless a writer has already been granted permission to use the object.
  - A reader that arrives after a waiting writer gets priority over the writer.
- ***Second readers-writers problem*** **(favors writers)**
  - Once a writer is ready to write, it performs its write as soon as possible
  - A reader that arrives after a writer must wait, even if the writer is also waiting.
- ***Starvation*** **(where a thread waits indefinitely) is possible in both cases.**

14

Questions

19-20p에 나오는 코드를 보면 byte_cnt변수를 static으로 정의했는데, 이러면 모든 thread들이 공유하기 때문에 여러 thread가 있을 때 전체 입력된 byte의 총합을 출력하게 되는 것 아닌가요? 하지만 출력되는 printf문을 보면 "thread %d received %d (%d total) bytes on fd"라고 되어있는 것을 보면 각 쓰레드 별 입력량인 것 같은데 이해가 잘 가지 않아 질문드립니다.

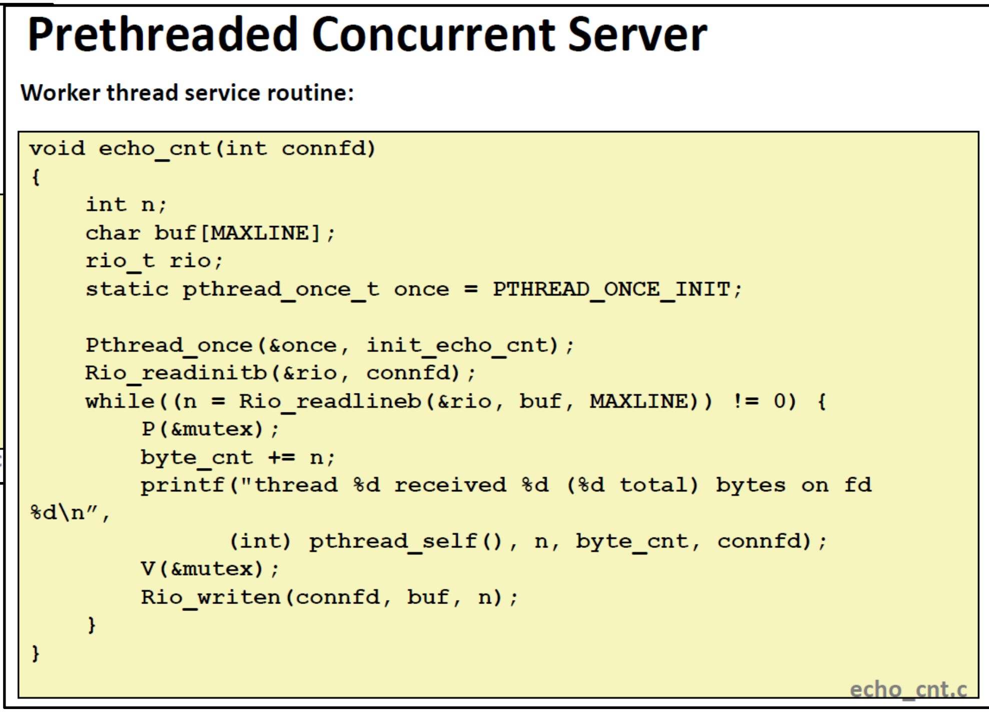


Prethreaded Concurrent Server

echo_cnt initialization routine:

```
static int byte_cnt;  /* Byte counter */
static sem_t mutex;   /* and the mutex that protects it */

static void init_echo_cnt(void)
{
    Sem_init(&mutex, 0, 1);
    byte_cnt = 0;
}
```

echo_c



Prethreaded Concurrent Server

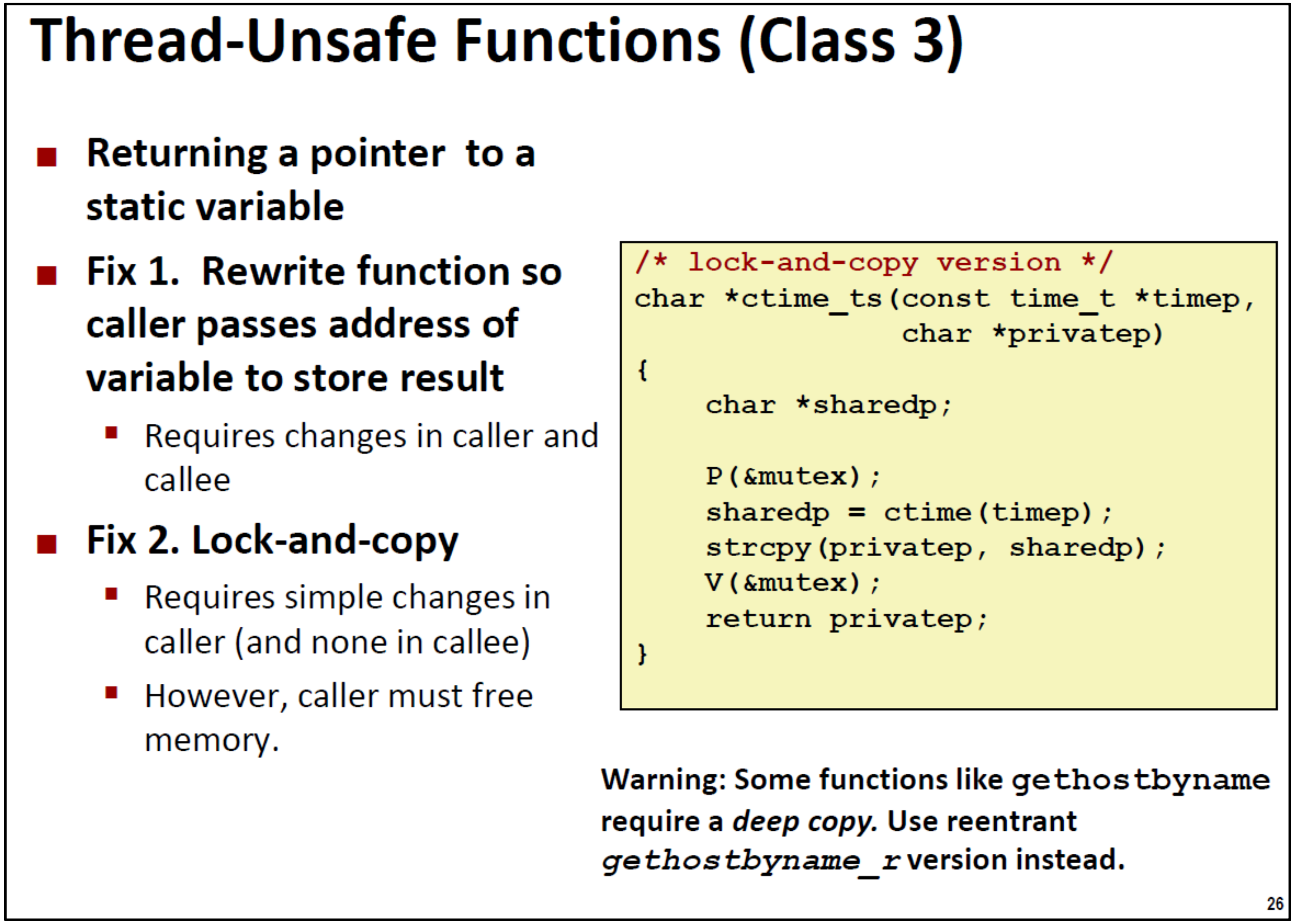
Worker thread service routine:

```
void echo_cnt(int connfd)
{
    int n;
    char buf[MAXLINE];
    rio_t rio;
    static pthread_once_t once = PTHREAD_ONCE_INIT;

    Pthread_once(&once, init_echo_cnt);
    Rio_readinitb(&rio, connfd);
    while((n = Rio_readlineb(&rio, buf, MAXLINE)) != 0) {
        P(&mutex);
        byte_cnt += n;
        printf("thread %d received %d (%d total) bytes on fd
%d\n”,
               (int) pthread_self(), n, byte_cnt, connfd);
        V(&mutex);
        Rio_writen(connfd, buf, n);
    }
}
```

echo_cnt.c

Questions

26p의 우측에 Lock-and-Copy version을 이용한 해결법 예시로 나와있는 코드를 보면 privatep라고 불리는 값을 넣어 줄 주소를 보내는 것처럼 보이는데, 이는 "Fix 1. Rewrite function so caller passes address of variable to store result" 의 예시가 아닌가요? 둘의 차이를 잘 이해하지 못한 것인지, 아니면 코드 예시가 잘못된 것인지 여쭤보고 싶습니다.



Thread-Unsafe Functions (Class 3)

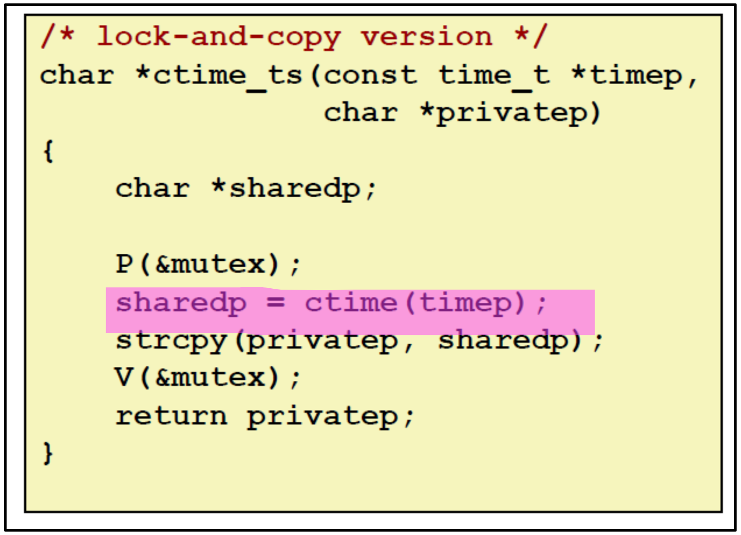
- **Returning a pointer to a static variable**
- **Fix 1. Rewrite function so caller passes address of variable to store result**
  - Requires changes in caller and callee
- **Fix 2. Lock-and-copy**
  - Requires simple changes in caller (and none in callee)
  - However, caller must free memory.

```
/* lock-and-copy version */
char *ctime_ts(const time_t *timep,
               char *privatep)
{
    char *sharedp;

    P(&mutex);
    sharedp = ctime(timep);
    strcpy(privatep, sharedp);
    V(&mutex);
    return privatep;
}
```

Warning: Some functions like `gethostbyname` require a *deep copy*. Use reentrant *`gethostbyname_r`* version instead.

Questions

26p의 우측에 Lock-and-Copy version을 이용한 해결법 예시로 나와있는 코드를 보면 privatep라고 불리는 값을 넣어줄 주소를 보내는 것처럼 보이는데, 이는 "Fix 1. Rewrite function so caller passes address of variable to store result" 의 예시가 아닌가요? 둘의 차이를 잘 이해하지 못한 것인지, 아니면 코드 예시가 잘못된 것인지 여쭤보고 싶습니다.

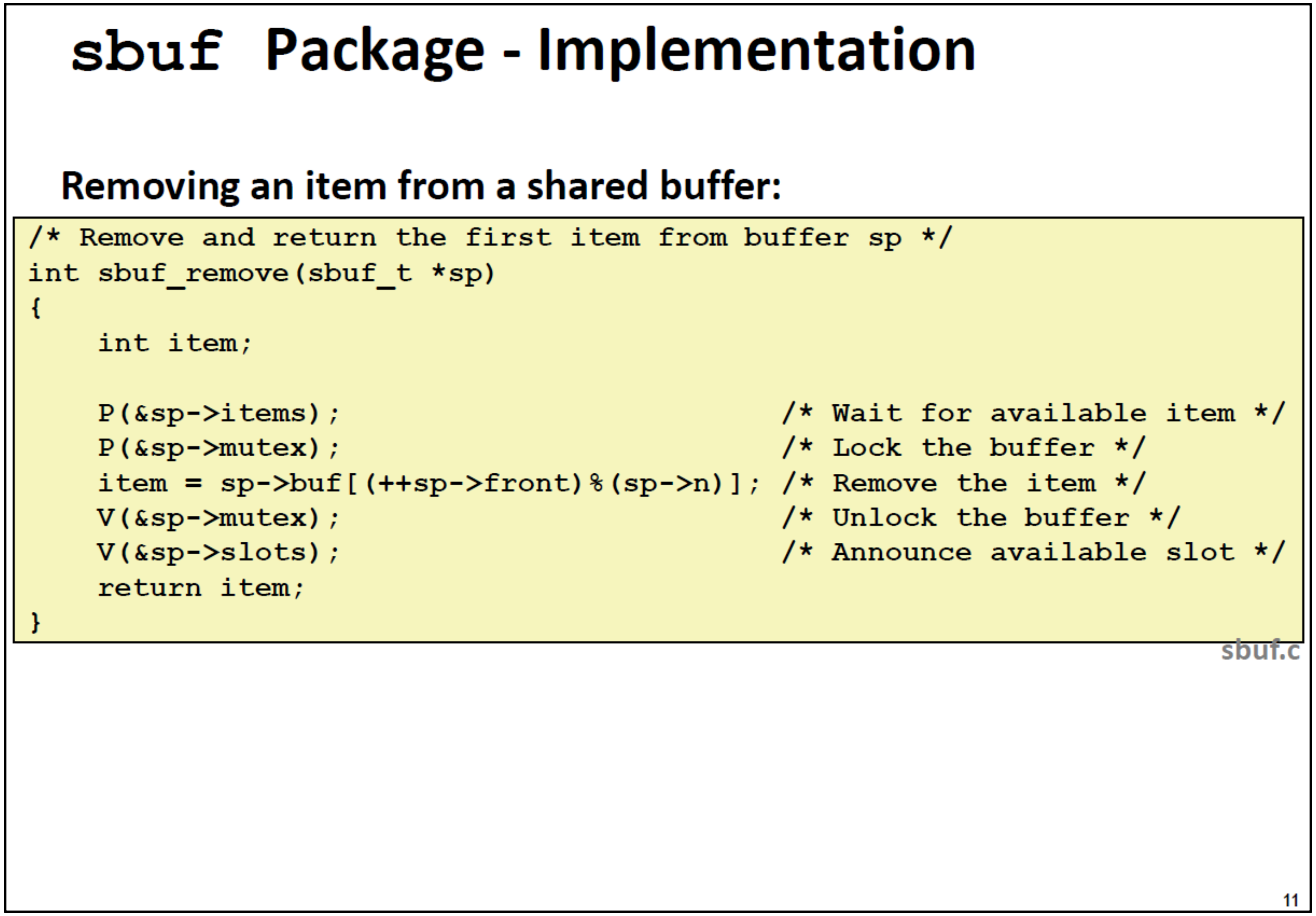


```
/* lock-and-copy version */
char *ctime_ts(const time_t *timep,
               char *privatep)
{
    char *sharedp;

    P(&mutex);
    sharedp = ctime(timep);
    strcpy(privatep, sharedp);
    V(&mutex);
    return privatep;
}
```

- *(textbook)*
- Some functions, such as `ctime` and `gethostbyname`, compute a result in a `static` variable and then return a pointer to that variable. If we call such functions from concurrent threads, then disaster is likely, as results being used by one thread are silently overwritten by another thread.
- Rewrite the function so that the caller passes the address of the variable in which to store the results.
  - Requires programmer to have access to the function source code.
- If the thread-unsafe function is difficult or impossible to modify, then another option is to use `lock-and-copy` technique.

Questions

17강 p11에서 결국 lock을 줄일 수 있는 방법은 mutex를 아예 사용하지 않는 것 같은데, 맞나요?



sbuf Package - Implementation

Removing an item from a shared buffer:

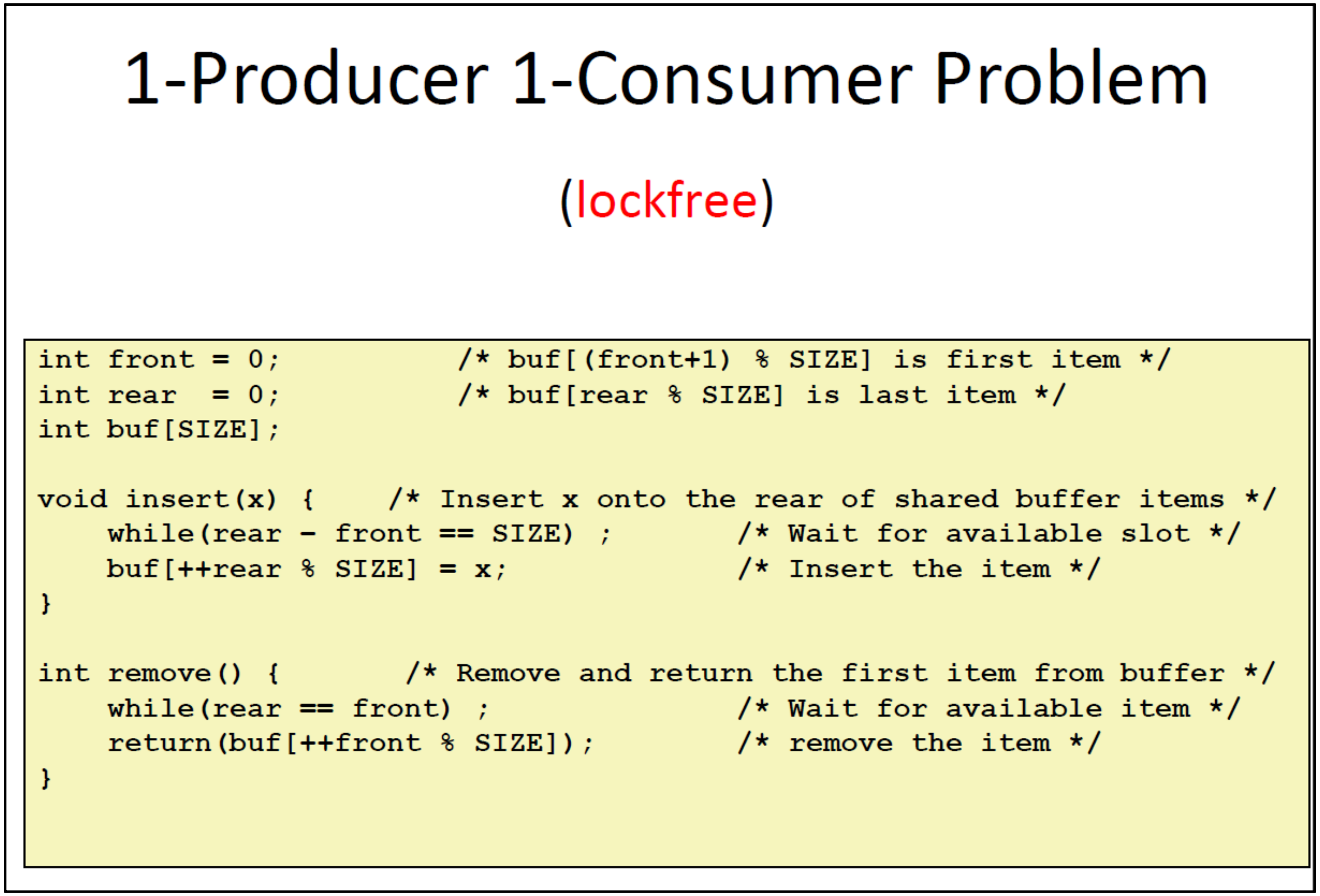
```
/* Remove and return the first item from buffer sp */
int sbuf_remove(sbuf_t *sp)
{
    int item;

    P(&sp->items);                          /* Wait for available item */
    P(&sp->mutex);                          /* Lock the buffer */
    item = sp->buf[(++sp->front)%(sp->n)]; /* Remove the item */
    V(&sp->mutex);                          /* Unlock the buffer */
    V(&sp->slots);                          /* Announce available slot */
    return item;
}
```

sbuf.c

11

Questions

17강 p11에서 결국 lock을 줄일 수 있는 방법은 mutex를 아예 사용하지 않는 것 같은데, 맞나요?



1-Producer 1-Consumer Problem

(lockfree)

```
int front = 0;           /* buf[(front+1) % SIZE] is first item */
int rear  = 0;           /* buf[rear % SIZE] is last item */
int buf[SIZE];

void insert(x) {     /* Insert x onto the rear of shared buffer items */
    while(rear - front == SIZE) ;       /* Wait for available slot */
    buf[++rear % SIZE] = x;             /* Insert the item */
}

int remove() {        /* Remove and return the first item from buffer */
    while(rear == front) ;              /* Wait for available item */
    return(buf[++front % SIZE]);        /* remove the item */
}
```