

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

14-513

18-613

Synchronization: Advanced

Assignment Project Exam Help
15-213/18-213/14-513/15-513/18-613:

Introduction to Comp
26th Lecture, Nov. 21, <https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Reminder: Semaphores

- **Semaphore:** non-negative global integer synchronization variable
- Manipulated by *P* and *V* operations:
 - *P(s)*: [**while** (*s* == 0); *s*--;]
 - Dutch for "Pr
 - *V(s)*: [*s*++ ;]
 - Dutch for "Verhogen" (increase)
- OS kernel guarantees that operations between brackets [] are executed atomically
 - Only one *P* or *V* operation at a time can modify *s*.
 - When **while** loop in *P* terminates, only that *P* can decrement *s*
- **Semaphore invariant:** (*s* >= 0)

Review: Using semaphores to protect shared resources via mutual exclusion

■ Basic idea:

- Associate a unique semaphore *mutex*, initially 1, with each shared variable (or related set of shared variables)
- Surround each access to the shared variable(s) with *P(mutex)* and *V(mutex)* operations

Add WeChat edu_assist_pro

```
mutex = 1  
  
P (mutex)  
cnt++  
V (mutex)
```

Review: Using Lock for Mutual Exclusion

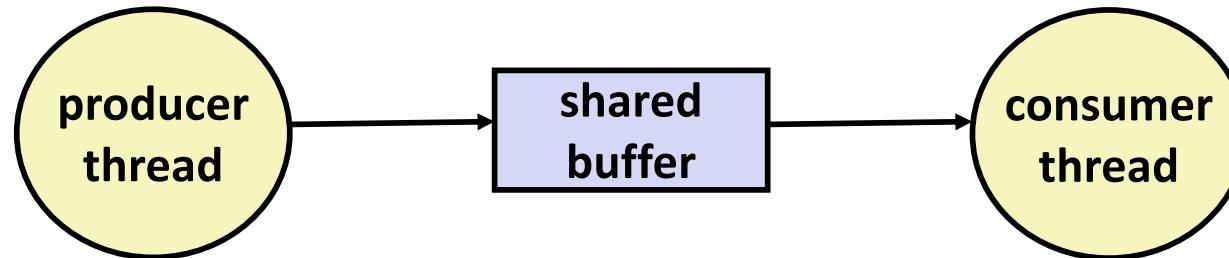
■ Basic idea:

- Mutex is special case of semaphore that only has value 0 (locked) or 1 (unlocked)
- $Lock(m)$: [~~while (m == 0); m=0;~~]
- $Unlock(m)$: [~~m=~~ <https://eduassistpro.github.io/>]
- ~~~2x faster than~~ his purpose
- And, more clearly indicates program

```
mutex = 1

lock(mutex)
cnt++
unlock(mutex)
```

Review: Producer-Consumer Problem



- **Common synchronization pattern:**
 - Producer waits for <https://eduassistpro.github.io/> buffer, and notifies consumer
 - Consumer waits for buffer, and notifies producer
- **Examples**
 - Multimedia processing:
 - Producer creates video frames, consumer renders them
 - Event-driven graphical user interfaces
 - Producer detects mouse clicks, mouse movements, and keyboard hits and inserts corresponding events in buffer
 - Consumer retrieves events from buffer and paints the display

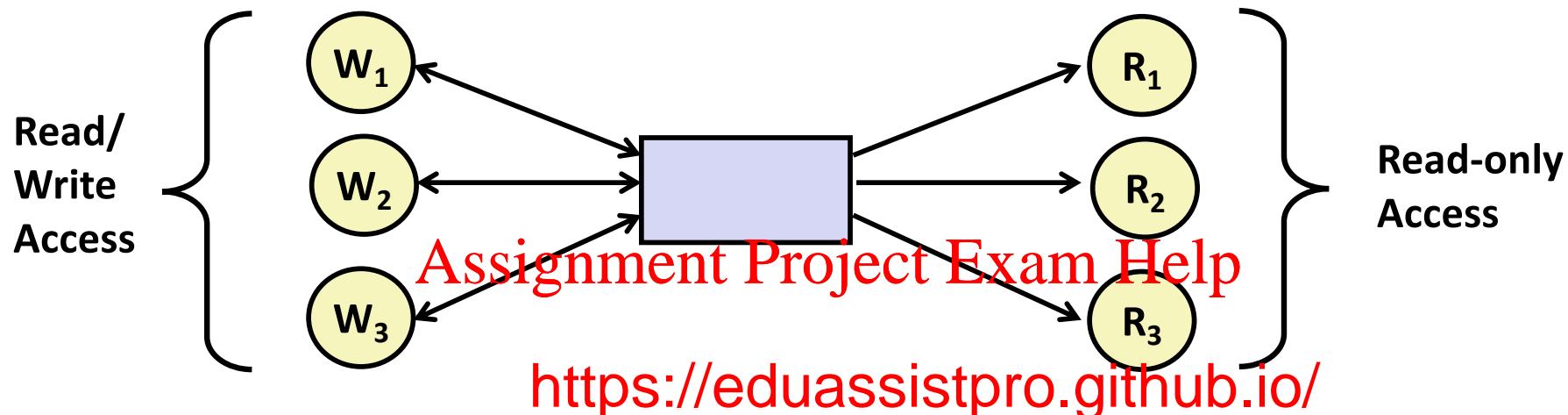
Review: Using Semaphores to Coordinate Access to Shared Resources

- Basic idea: Thread uses a semaphore operation to notify another thread that some condition has become true
 - Use counting semaphores to keep track of resource state.
 - Use binary sema ds.
- The Producer-Consumer Problem
 - Mediating interactions between processes that generate information and that then make use of that information
 - Single entry buffer implemented with two binary semaphores
 - One to control access by producer(s)
 - One to control access by consumer(s)
 - N-entry implemented with semaphores + circular buffer

Today

- **Using semaphores to schedule shared resources**
 - Readers-writers problem
 - **Other concurrency issues**
 - Thread safety
 - Races
 - Deadlocks
 - Interactions between threads and semaphores
- Assignment Project Exam Help
<https://eduassistpro.github.io/>
Add WeChat edu_assist_pro

Readers-Writers Problem



■ Problem statement

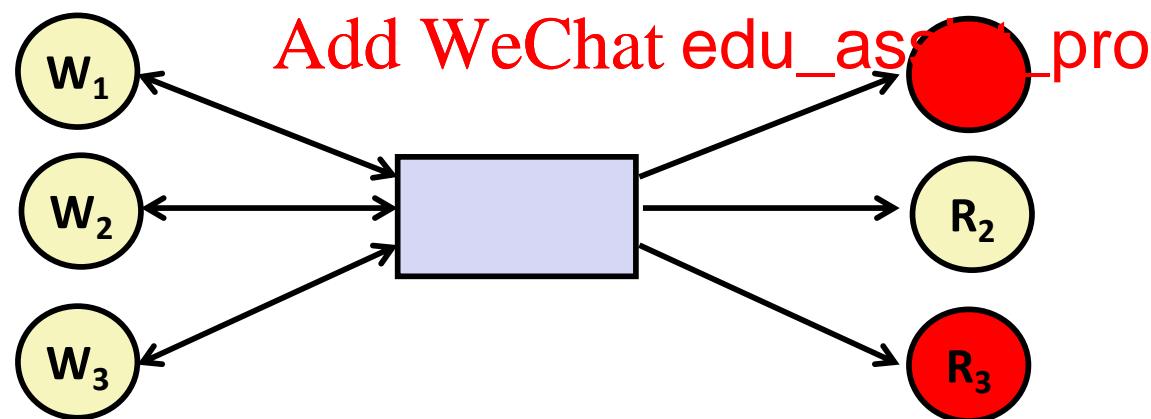
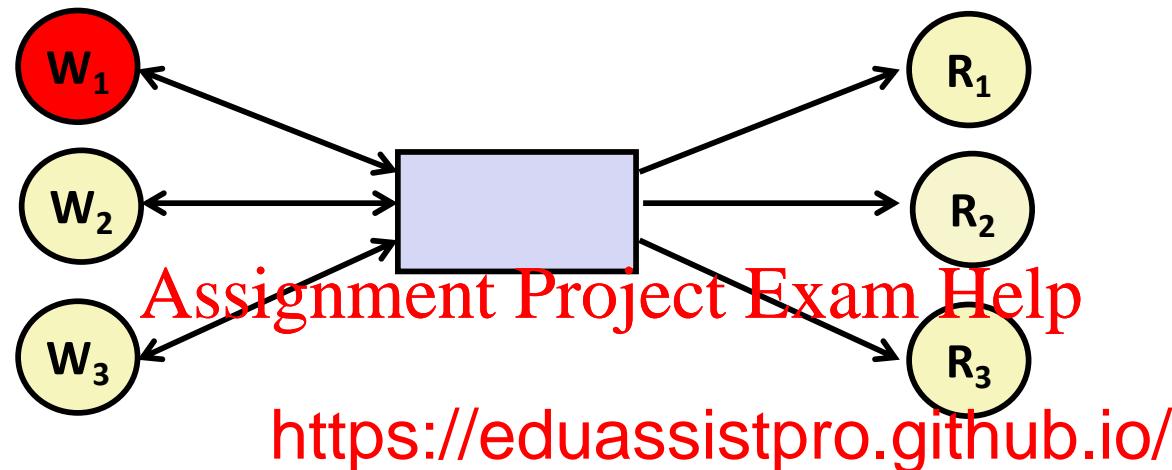
- *Reader* threads only read the object
- *Writer* threads modify the object (read/write access)
- Writers must have exclusive access to the object
- Unlimited number of readers can access the object

Add WeChat edu_assist_pro

■ Occurs frequently in real systems, e.g.,

- Online airline reservation system
- Multithreaded caching Web proxy

Readers/Writers Examples



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.

<https://eduassistpro.github.io/>

■ *Second readers-writers problem (favors writers)*

- Once a writer is ready to write, it writes 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.*

Solution to First Readers-Writers Problem

Readers:

```
int readcnt;      /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
    while (1) {
        P(&mutex);
        readcnt++;
        if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);

        /* Reading happens here */

        P(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            V(&w);
        V(&mutex);
    }
}
```

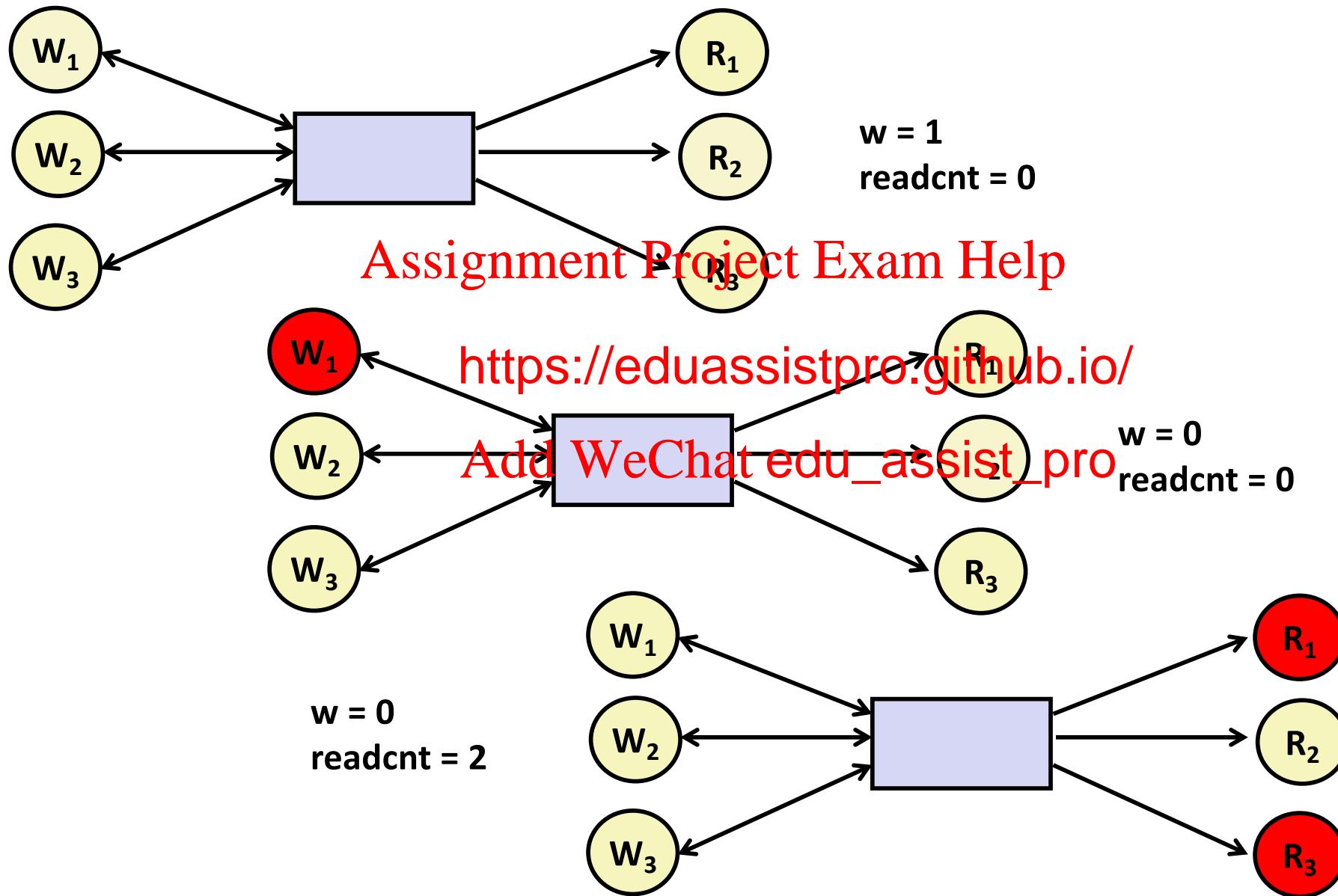
Writers:

```
void writer(void)
{
    while (1) {
        P(&w);
        /* Writing here */
        V(&w),
    }
}
```

Assignment Project Exam Help
<https://eduassistpro.github.io/>
Add WeChat edu_assist_pro

rw1.c

Readers/Writers Examples



Solution to First Readers-Writers Problem

Readers:

```

int readcnt;      /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
    while (1) {
        P(&mutex);
        readcnt++;
        if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);

        /* Reading happens here */

        P(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            V(&w);
        V(&mutex);
    }
}

```

Writers:

```

void writer(void)
{
    while (1) {
        P(&w);
        /* Writing here */
        V(&w),
}

```

Assignment Project Exam Help
<https://eduassistpro.github.io/>
Add WeChat edu_assist_pro

rw1.c

Arrivals: R1 R2 W1 R3

Solution to First Readers-Writers Problem

Readers:

```
int readcnt;      /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
    while (1) {
        P(&mutex);
        readcnt++;
        if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);
    }
}
```

Writers:

```
void writer(void)
{
    while (1) {
        P(&w);
        /* Writing here */
        V(&w);
    }
}
```

Assignment Project Exam Help
<https://eduassistpro.github.io/>
 Add WeChat edu_assist_pro

R1 → /* Reading happens here */

```
P(&mutex);
readcnt--;
if (readcnt == 0) /* Last out */
    V(&w);
V(&mutex);
}
```

rw1.c

Arrivals: R1 R2 W1 R3

Readcnt == 1
 W == 0

Solution to First Readers-Writers Problem

Readers:

```

int readcnt;      /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
    while (1) {
        P(&mutex);
        readcnt++;
        R2 → if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);

        R1 → /* Reading happens here */

        P(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            V(&w);
        V(&mutex);
    }
}

```

Writers:

```

void writer(void)
{
    while (1) {
        P(&w);
        /* Writing here */
        V(&w);
}

```

Assignment Project Exam Help
<https://eduassistpro.github.io/>
 Add WeChat edu_assist_pro

rw1.c

Arrivals: R1 R2 W1 R3

Readcnt == 2
W == 0

Solution to First Readers-Writers Problem

Readers:

```

int readcnt;      /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
    while (1) {
        P(&mutex);
        readcnt++;
        if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);

R2 →
R1 → /* Reading happens here */

        P(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            V(&w);
        V(&mutex);
    }
}

```

Writers:

```

void writer(void)
{
    while (1) {
        P(&w); ← W1
        /* Writing here */
        V(&w);
}

```

Assignment Project Exam Help
<https://eduassistpro.github.io/>
 Add WeChat edu_assist_pro

rw1.c

Arrivals: R1 R2 W1 R3

Readcnt == 2
W == 0

Solution to First Readers-Writers Problem

Readers:

```

int readcnt;      /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
    while (1) {
        P(&mutex);
        readcnt++;
        if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);

        /* Reading happens here */
        P(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            V(&w);
        V(&mutex);
    }
}

```

R2

→ /* Reading happens here */

R1

→ }

Writers:

```

void writer(void)
{
    while (1) {
        P(&w); ← W1
        /* Writing here */
        V(&w),
}

```

Assignment Project Exam Help
<https://eduassistpro.github.io/>
 Add WeChat edu_assist_pro

rw1.c

Arrivals: R1 R2 W1 R3

Readcnt == 1

W == 0

Solution to First Readers-Writers Problem

Readers:

```

int readcnt;      /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
    while (1) {
        P(&mutex);
        readcnt++;
        R3 → if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);

        /* Reading happens here */
        R2 → P(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            V(&w);
        V(&mutex);
    }
}

```

Writers:

```

void writer(void)
{
    while (1) {
        W1 ← P(&w);
        /* Writing here */
        V(&w),
}

```

Assignment Project Exam Help
<https://eduassistpro.github.io/>
 Add WeChat edu_assist_pro

rw1.c

Arrivals: R1 R2 W1 R3

Readcnt == 2
W == 0

Solution to First Readers-Writers Problem

Readers:

```

int readcnt;      /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
    while (1) {
        P(&mutex);
        readcnt++;
        if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);

R3 → /* Reading happens here */

        P(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            V(&w);
    }
}

```

Writers:

```

void writer(void)
{
    while (1) {
        P(&w); ← W1
        /* Writing here */
        V(&w);
}

```

Assignment Project Exam Help
<https://eduassistpro.github.io/>
 Add WeChat edu_assist_pro

rw1.c

Arrivals: R1 R2 W1 R3

Readcnt == 1
W == 0

Solution to First Readers-Writers Problem

Readers:

```

int readcnt;      /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
    while (1) {
        P(&mutex);
        readcnt++;
        if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);

        /* Reading happens here */

        P(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            V(&w);
        V(&mutex);
    }
}

```

Writers:

```

void writer(void)
{
    while (1) {
        P(&w); ← W1
        /* Writing here */
        V(&w),
}

```

Assignment Project Exam Help
<https://eduassistpro.github.io/>
Add WeChat edu_assist_pro

rw1.c

Arrivals: R1 R2 W1 R3

Readcnt == 0
W == 1

R3 →

Other Versions of Readers-Writers

■ Shortcoming of first solution

- Continuous stream of readers will block writers indefinitely

■ Second version

- Once writer comes along, blocks access to later readers
- Series of writes

■ FIFO implementation

- See rwqueue code in code directory
- Service requests in order received
- Threads kept in FIFO
- Each has semaphore that enables its access to critical section

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Solution to Second Readers-Writers Problem

```
int readcnt, writecnt;           // Initially 0
sem_t rmutex, wmutex, r, w; // Initially 1
void reader(void)
{
    while (1) {
        P(&r);
        P(&rmutex);
        readcnt++;
        /* Reading happens here */
        V(&w);
        V(&rmutex);
    }
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Solution to Second Readers-Writers Problem

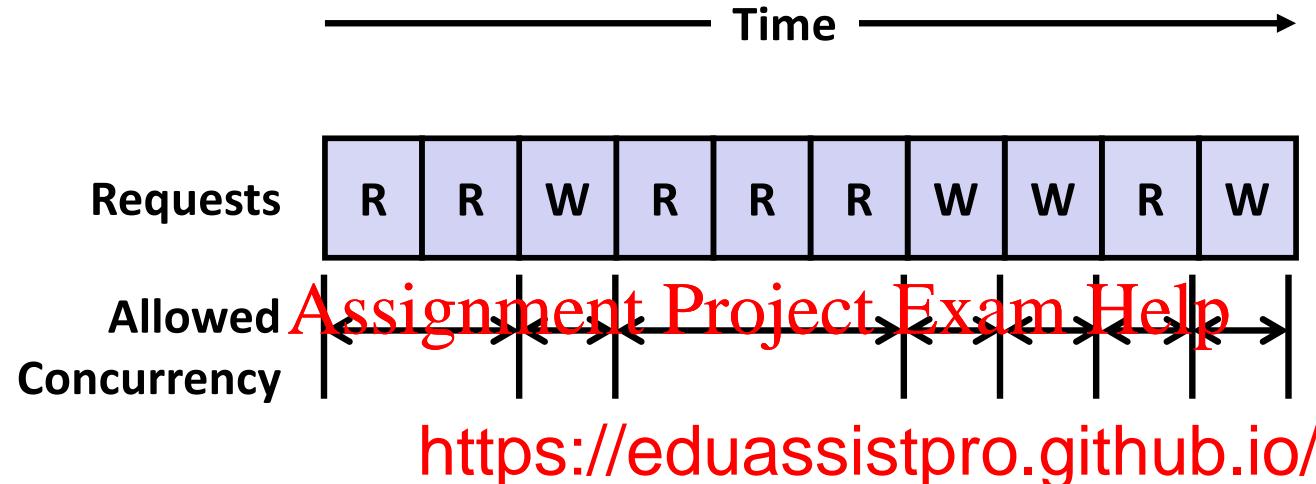
```
void writer(void)
{
    while (1) {
        P(&wmutex);
        writecnt++;
        if (writecnt == 1)
            P(&r);
        V(&wmutex);
    }
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Managing Readers/Writers with FIFO



■ Idea

Add WeChat edu_assist_pro

- Read & Write requests are inserted
- Requests handled as remove from FIFO
 - Read allowed to proceed if currently idle or processing read
 - Write allowed to proceed only when idle
- Requests inform controller when they have completed

■ Fairness

- Guarantee every request is eventually handled

Readers Writers FIFO Implementation

■ Full code in rwqueue.{h,c}

```
/* Queue data structure */
typedef struct {
    sem_t mutex;           /* Mutual exclusion */
    int reading_count;    // Number of active readers
    int writers;           /* Writers in queue */
    rw_token_t *head;      /* Head of queue */
    rw_token_t *tail;      /* Tail of queue */
} rw_queue_t;
```

Assignment Project Exam Help
with tail// FIFO queue in <https://eduassistpro.github.io/>
Add WeChat edu_assist_pro

```
/* Represents individual thread's position in queue */
typedef struct TOK {
    bool is_reader;
    sem_t enable;          // Enables access
    struct TOK *next;     // Allows chaining as linked list
} rw_token_t;
```

Readers Writers FIFO Use

■ In rwqueue-test.c

```
/* Get write access to data and write */
void iwriter(int *buf, int v)
{
    rw_token_t tok;
    rw_queue_request_write(&q, &tok);
    /* Critical section */
    *buf = v;
    /* End of Critical section */
    rw_queue_release(&q);
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

```
Add WeChat edu_assist_pro and read */

int iread()
{
    rw_token_t tok;
    rw_queue_request_read(&q, &tok);
    /* Critical section */
    int v = *buf;
    /* End of Critical section */
    rw_queue_release(&q);
    return v;
}
```

Library Reader/Writer Lock

- Data type `pthread_rwlock_t`

- Operations

- Acquire read lock

~~Pthread_rwlock_rdlock(pthread_rwlock_t *rwlock)~~

- Acquire write loc

~~Pthread_rwlock_wrlock(pthread_rwlock_t *rwlock)~~

- Release (either) lock

~~Add WeChat edu_assist_pro~~

~~Pthread_rwlock_unlock(pthread_rwlock_t *rwlock)~~

- Observation

- Library must be used correctly!

- Up to programmer to decide what requires read access and what requires write access

Today

- Using semaphores to schedule shared resources

- Readers-writers problem

- Other concurrency issues

- Races [Assignment](#) [Project](#) [Exam](#) [Help](#)
 - Deadlocks
 - Thread safety
 - Interactions between threads and sockets

<https://eduassistpro.github.io/>

[Add WeChat edu_assist_pro](#)

One Worry: Races

- A *race* occurs when correctness of the program depends on one thread reaching point x before another thread reaches point y

```
/* a threaded program with a race */
int main(int argc, char** argv) {
    pthread_t tid[N];
    int i;
    for (i = 0; i < N; i++) {
        Pthread_create(tid[i], NULL, &i);
    }
    for (i = 0; i < N; i++)
        Pthread_join(tid[i], NULL);
    return 0;
}

/* thread routine */
void *thread(void *vargp) {
    int myid = *((int *)vargp);
    printf("Hello from thread %d\n", myid);
    return NULL;
}
```

Data Race

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro



Race Elimination

- **Don't share state**
 - E.g., use malloc to generate separate copy of argument for each thread
- **Use synchronization primitives to control access to shared state**
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Today

- Using semaphores to schedule shared resources

- Producer-consumer problem

- Other concurrency issues

- Races **Assignment Project Exam Help**

- Deadlocks

- Thread safety **<https://eduassistpro.github.io/>**

- Interactions between threads and semaphores

- **Add WeChat edu_assist_pro**

A Worry: Deadlock

- Def: A process is *deadlocked* iff it is waiting for a condition that will never be true.

- Typical Scenario
 - Processes 1 and 2
 - Process 1 acquire B) to proceed
 - Process 2 acquires B waits for A
 - Both will wait forever!

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Deadlocking With Semaphores

```

int main(int argc, char** argv)
{
    pthread_t tid[2];
    Sem_init(&mutex[0], 0, 1); /* mutex[0] = 1 */
    Sem_init(&mutex[1], 0, 1); /* mutex[1] = 1 */
    Pthread_create(&tid[0], NULL, count, (void*) 0);
    Pthread_create(&tid[1], NULL, count, (void*) 1);
    Pthread_join(tid[0], NULL);
    Pthread_join(tid[1], NULL);
    printf("cnt=%d\n");
    return 0;
}

```

Assignment Project Exam Help

Add WeChat edu_assist_pro

```

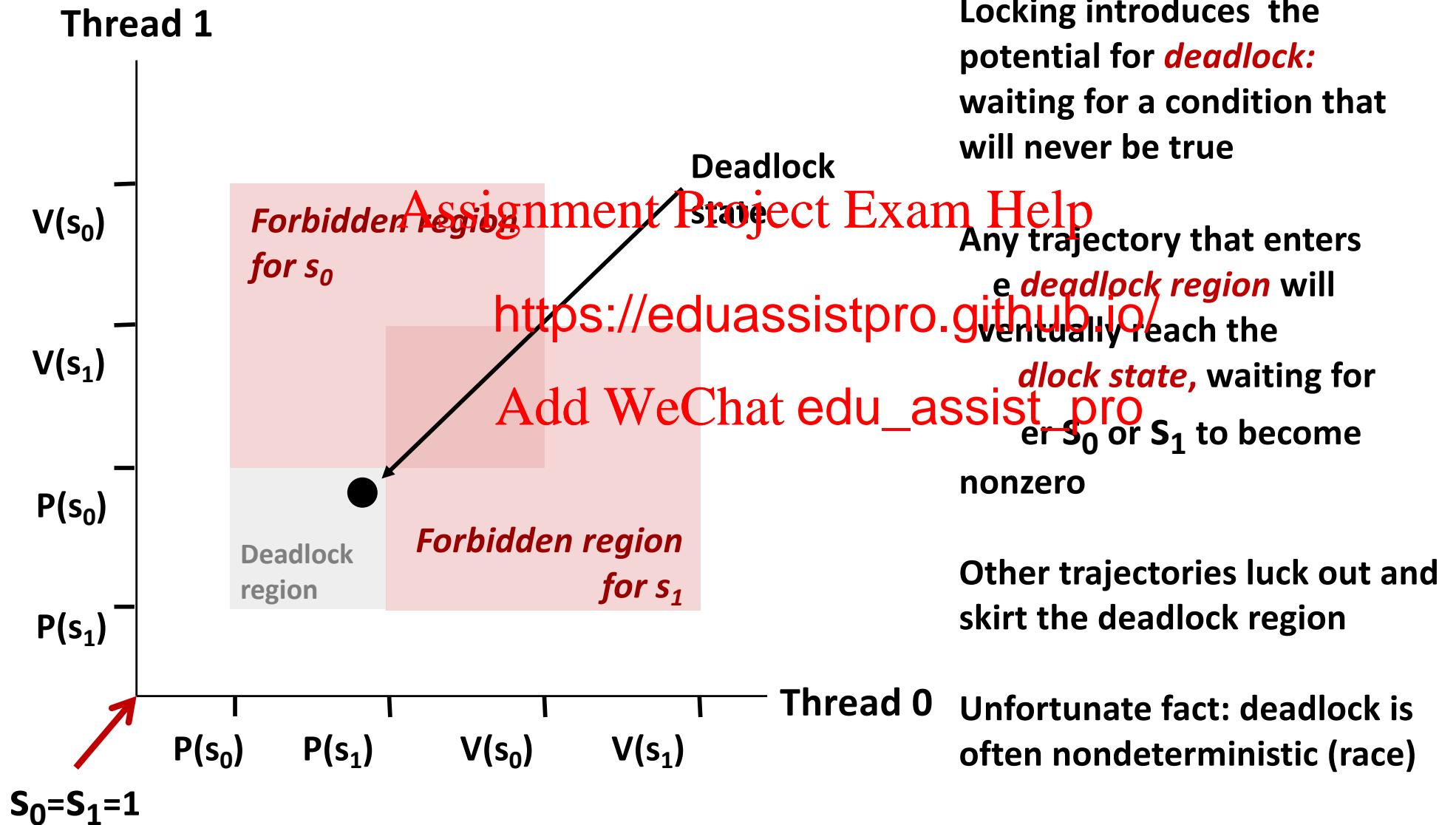
void *count(void *vargp)
{
    int i;
    int id = (int) vargp;
    for (i = 0; i < NITERS; i++) {
        P(&mutex[id]); P(&mutex[1-id]);
        cnt++;
        V(&mutex[id]); V(&mutex[1-id]);
    }
    return NULL;
}

```

Tid[0] :
 P(s_0);
 P(s_1);
 cnt++;
 V(s_0);
 V(s_1);

Tid[1] :
 P(s_1);
 P(s_0);
 cnt++;
 V(s_1);
 V(s_0);

Deadlock Visualized in Progress Graph



Deadlock

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Avoiding Deadlock

Acquire shared resources in same order

```

int main(int argc, char** argv)
{
    pthread_t tid[2];
    Sem_init(&mutex[0], 0, 1); /* mutex[0] = 1 */
    Sem_init(&mutex[1], 0, 1); /* mutex[1] = 1 */
    Pthread_create(&tid[0], NULL, count, (void*) 0);
    Pthread_create(&tid[1], NULL, count, (void*) 1);
    Pthread_join(tid[0], NULL);
    Pthread_join(tid[1], NULL);
    printf("cnt=%d\n"https://eduassistpro.github.io/
    return 0;
}

```

Assignment Project Exam Help

Add WeChat edu_assist_pro

```

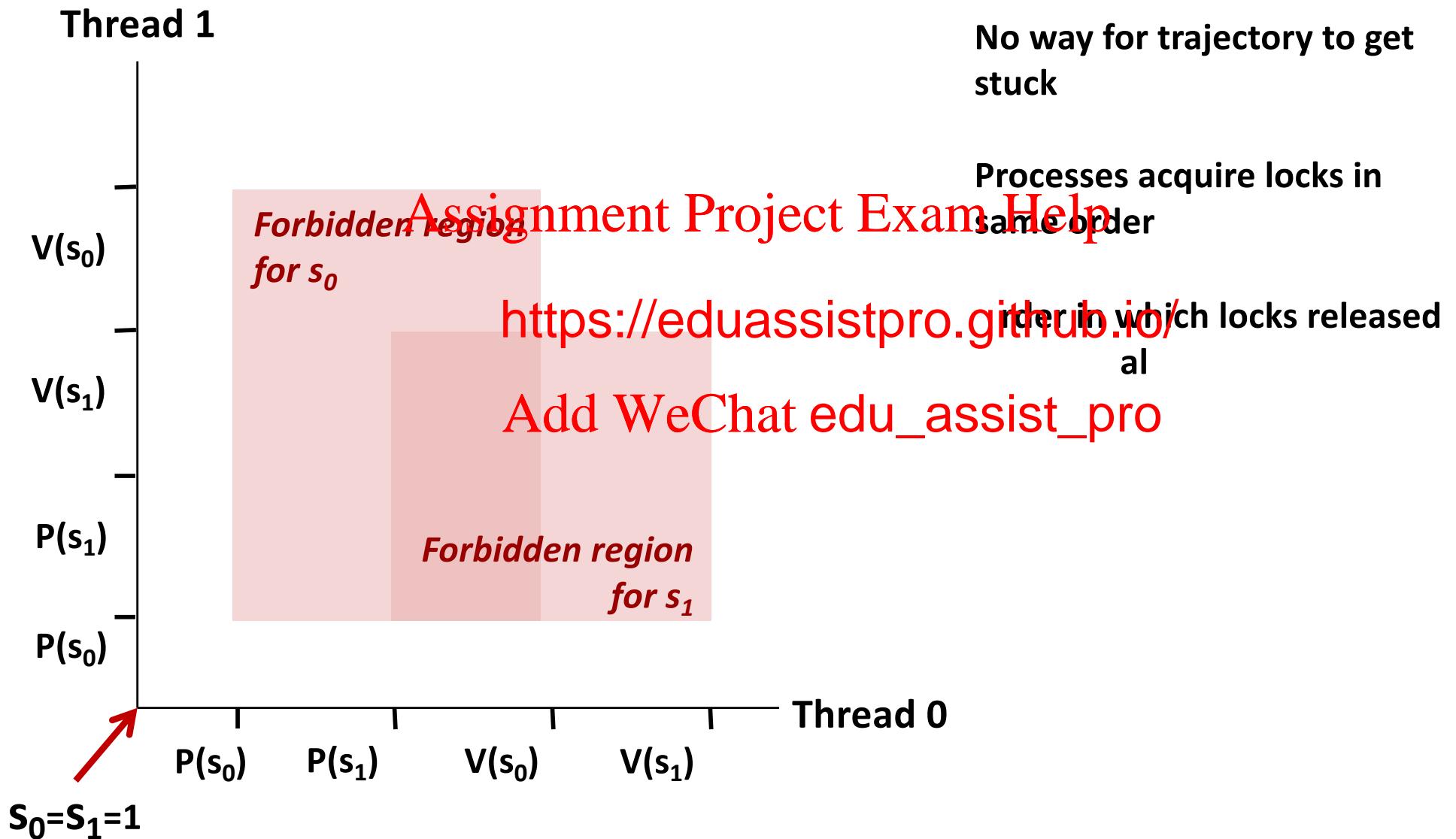
void *count(void *vargp)
{
    int i;
    int id = (int) vargp;
    for (i = 0; i < NITERS; i++) {
        P(&mutex[0]); P(&mutex[1]);
        cnt++;
        V(&mutex[id]); V(&mutex[1-id]);
    }
    return NULL;
}

```

Tid[0] :
 P(s_0);
 P(s_1);
 cnt++;
 V(s_0);
 V(s_1);

Tid[1] :
 P(s_0);
 P(s_1);
 cnt++;
 V(s_1);
 V(s_0);

Avoided Deadlock in Progress Graph



Demonstration

- See program deadlock.c
- 100 threads, each acquiring same two locks
- Risky mode
 - Even numbered threads request locks in opposite order of odd-numbered ones
<https://eduassistpro.github.io/>
- Safe mode
 - All threads acquire locks in same order

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Quiz Time!

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Check out:

<https://canvas.cmu.edu/courses/10968>

Today

- Using semaphores to schedule shared resources

- Readers-writers problem

- Other concurrency issues

- Races **Assignment Project Exam Help**

- Deadlocks

- Thread safety

- Interactions between threads and semaphores

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Crucial concept: Thread Safety

- Functions called from a thread must be *thread-safe*
- *Def:* A function is *thread-safe* iff it will always produce correct results when called repeatedly from multiple concurrent threads.
<https://eduassistpro.github.io/>
- Classes of thread-unsafe functions
 - Class 1: Functions that do not protect shared variables
 - Class 2: Functions that keep state across multiple invocations
 - Class 3: Functions that return a pointer to a static variable
 - Class 4: Functions that call thread-unsafe functions

Thread-Unsafe Functions (Class 1)

■ Failing to protect shared variables

- Fix: Use *P* and *V* semaphore operations (or mutex)
- Example: `goodcnt.c`
- Issue: Synchronisation operations will slow down code

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Thread-Unsafe Functions (Class 2)

- Relying on persistent state across multiple function invocations
 - Example: Random number generator that relies on static state

```
static unsigned int next = 1;  
/* rand: 32-bit random number */  
int rand(void)    https://eduassistpro.github.io/  
{  
    next = next * 1103515245 + 123  
    return (unsigned int)(next / 68);  
}  
  
/* srand: set seed for rand() */  
void srand(unsigned int seed)  
{  
    next = seed;  
}
```

Thread-Safe Random Number Generator

- Pass state as part of argument

- and, thereby, eliminate static state

Assignment Project Exam Help

```
/* rand_r - return pseudo-random integer on 0..32767 */

int rand_r(int *nhttps://eduassistpro.github.io/
{
    *nextp = *nextp*1103515245+
    return (unsigned int) (*nextp
    }  
Add WeChat edu_assist_pro  
2768;
```

- Consequence: programmer using `rand_r` must maintain seed

Thread-Unsafe Functions (Class 3)

- Returning a pointer to a static variable
- Fix 1. Rewrite function so caller passes ~~Assignment of~~ Project Exam Help
 - Requires change callee
- Fix 2. Lock-and-copy
 - Requires simple changes in caller (and none in callee)
 - However, caller must free memory.

```
/* Convert integer to string */  
char *itoa(int x)  
{  
    static char buf[11];  
    sprintf(buf, "%d", x);  
    return buf;  
}
```

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```
oa (int x, char *dest)  
{  
    strcpy(dest, itoa(x));  
    V(&mutex);  
    return dest;  
}
```

Thread-Unsafe Functions (Class 4)

■ Calling thread-unsafe functions

- Calling one thread-unsafe function makes the entire function that calls it thread-unsafe
- Fix: Modify the function so it calls only thread-safe functions ☺

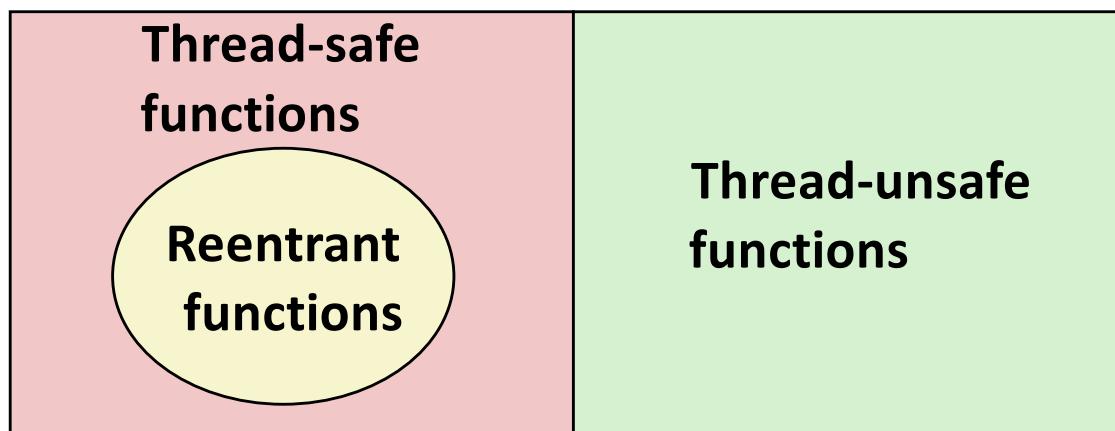
<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Reentrant Functions

- Def: A function is *reentrant* iff it accesses no shared variables when called by multiple threads.
 - Important subset of thread-safe functions
 - Require synchronization operations
 - Only way to make it ad-safe is to make it reentrant (e. <https://eduassistpro.github.io/>)

All functions Add WeChat edu_assist_pro



Thread-Safe Library Functions

- All functions in the Standard C Library (at the back of your K&R text) are thread-safe
 - Examples: `malloc`, `free`, `printf`, `scanf`
- Most Unix system calls are thread-safe, with a few exceptions:

<https://eduassistpro.github.io/>

Thread-unsafe function	Class	ion
<code>asctime</code>	3	
<code>ctime</code>	3	<code>ctime_r</code>
<code>gethostbyaddr</code>	3	<code>gethostbyaddr_r</code>
<code>gethostbyname</code>	3	<code>gethostbyname_r</code>
<code>inet_ntoa</code>	3	(none)
<code>localtime</code>	3	<code>localtime_r</code>
<code>rand</code>	2	<code>rand_r</code>

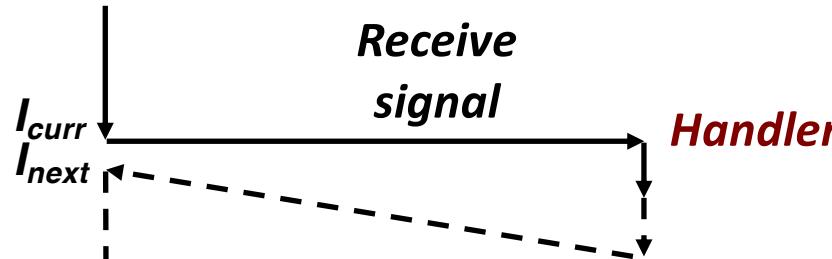
Today

- Using semaphores to schedule shared resources
 - Readers-writers problem
- Other concurrency issues
 - Races **Assignment Project Exam Help**
 - Deadlocks
 - Thread safety
 - Interactions between threads and programs

<https://eduassistpro.github.io/>

[Add WeChat edu_assist_pro](#)

Signal Handling Review



Assignment Project Exam Help

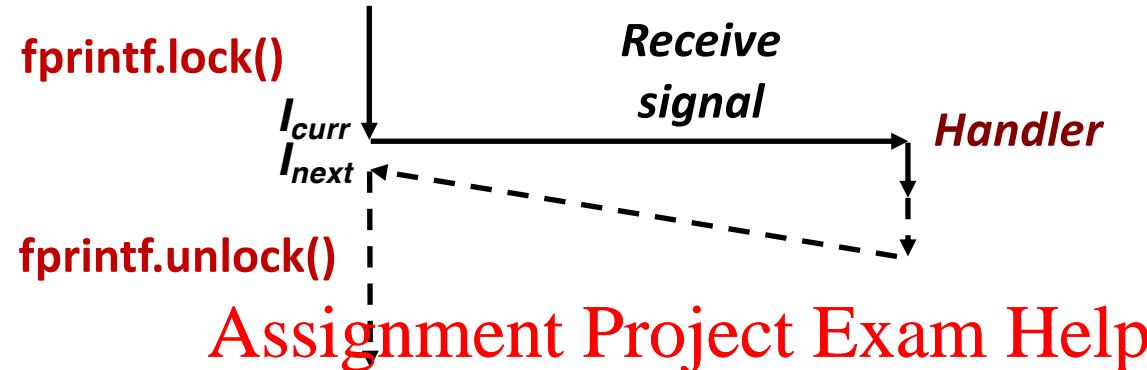
■ Action

<https://eduassistpro.github.io/>

- Signal can occur at any point in program
 - Unless signal is blocked
- Signal handler runs within same thread
- Must run to completion and then return to regular program execution

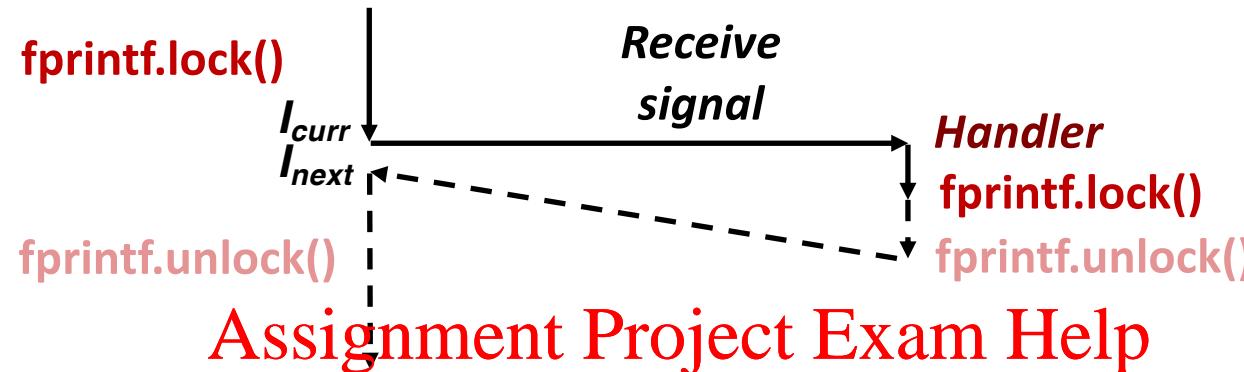
Add WeChat  on

Threads / Signals Interactions



- Many library functions for thread safety
 - Because they have hidden state
 - malloc
 - Free lists
 - fprintf, printf, puts
 - So that outputs from multiple threads don't interleave
 - sprintf
 - Not officially asynch-signal-safe, but seems to be OK
- OK for handler that doesn't use these library functions

Bad Thread / Signal Interactions



■ What if: <https://eduassistpro.github.io/>

- Signal received while library function `Add WeChat edu_assist_pro` is locked
- Handler calls same (or related) function

■ Deadlock!

- Signal handler cannot proceed until it gets lock
- Main program cannot proceed until handler completes

■ Key Point

- Threads employ symmetric concurrency
- Signal handling is asymmetric

Threads Summary

- Threads provide another mechanism for writing concurrent programs
- Threads are growing in popularity
 - Somewhat cheaper than processes
 - Easy to share data
- However, the ease of use can lead to bugs:
 - Easy to introduce subtle synchronization errors
 - Tread carefully with threads!
- For more info:
 - D. Butenhof, “Programming with Posix Threads”, Addison-Wesley, 1997