SER 334 A Session

SI Session

Monday, February 5th 2024

7:00 pm - 8:00 pm MST

Agenda

- Sample Problems!
- Module 7
- Module 8
- Sample Requests

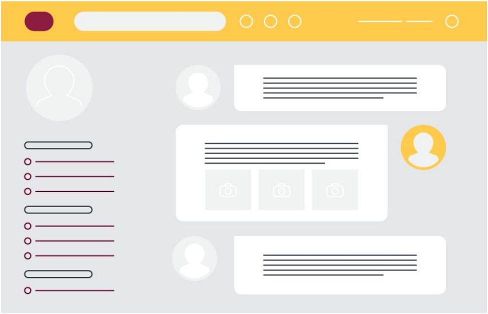
SI Session Expectations

Thanks for coming to the **SER 334** SI session. We have a packed agenda and we are going to try to get through as many of our planned example problems as possible. This session will be recorded and shared with others.

- If after this you want to see additional examples, please visit the drop-in tutoring center.
- We will post the link in the chat now and at the end of the session.
  - tutoring.asu.edu
- Please keep in mind we are recording this session and it will be made available for you to review 24-48 hours after this session concludes.
- Finally, please be respectful to each other during the session.

Interact with us:

Zoom Features



Zoom Chat

- Use the chat feature to interact with the presenter and respond to presenter's questions.
- Annotations are encouraged

SER 334
Threading Issues

Check out the recording for the solution!

- Test and Debugging
- Data Splitting
- Identifying Tasks
- Data Dependency
- Balance

A. Identifying independent functionality

B. Ensuring comparable amounts of work

C. Partitioning and minimizing memory use

D. Some tasks must be performed sequentially

E. Multiple “executors” for each line makes it harder to identify the culprit

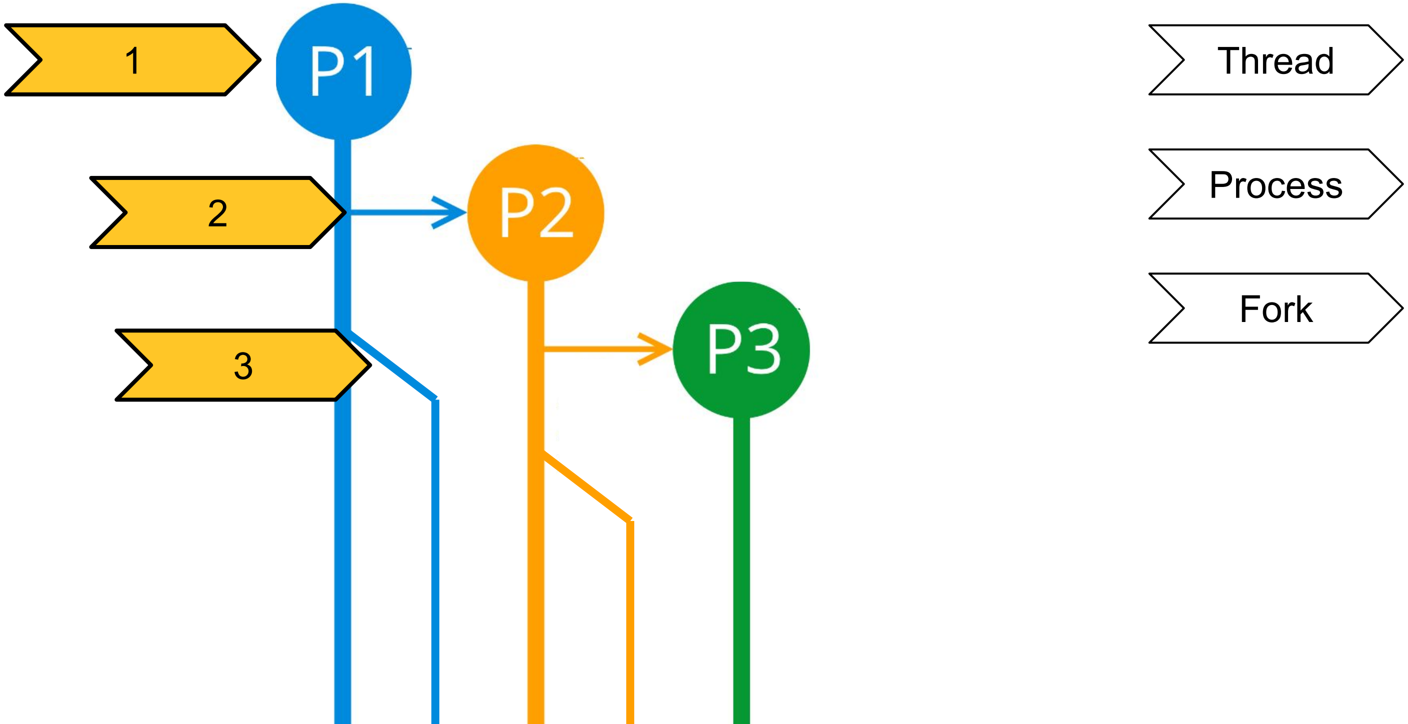
SER 334

Execution Tracing

(a) Using "lifeline notation" (sequence diagrams with threads), draw the creation of processes and threads during execution.

(b) How many unique processes are created? (Do not include the initial process.)

(c) How many unique threads are created? (Hint: processes don't count!)



SER 334
Execution Tracing

(a) Using “lifeline notation” (sequence diagrams with threads), draw the creation of processes and threads during execution.

(b) How many unique processes are created? (Do not include the initial process.)

(c) How many unique threads are created? (Hint: processes don't count!)

```
pid_t pid;
pid = fork();

thread_create(...);

if (pid == 0) {
    pid = fork();
    if(pid == 0)
        pid = fork();
}

thread_create(...);
```

SER 334
Execution Tracing

(a) Using “lifeline notation” (sequence diagrams with threads), draw the creation of processes and threads during execution.

(b) How many unique processes are created? (Do not include the initial process.)

(c) How many unique threads are created? (Hint: processes don't count!)

```
pid_t pid;

thread_create(...);

pid = fork();

thread_create(...);

if (pid == 0) {
    pid = fork();
    if(pid == 0)
        pid = fork();
}

thread_create(...);
```

SER 334
Samples

5. [Acuña] Consider the code for Peterson's Solution. Notice that part of the algorithm has been commented out. Explain how this changes it's functionality. Will it still solve the critical section problem? Explain.

```
//shared memory
int turn = 0;
bool flag[2] = { false, false };


//for some process i
do {
    flag[i] = true;
    turn = j;
    while (flag[j] /* && turn == j */);
    //critical section
    flag[i] = false;
    //remainder section
} while (true);
```

SER 334
Samples

9. [Lisonbee] Given the following partially implemented code, implement the calls to mutex lock and unlock in the appropriate spots in the runner function to ensure that two threads don't write to the same index in the memory array.

```
int main() {
	pthread_t threads[NUM_THREADS];

	pthread_mutex_init(&lock, NULL);
	for (int i = 0; i < NUM_THREADS; i++) {
		pthread_create(threads[i], NULL, runner, (i * 7));
	}
	for (int i = 0; i < NUM_THREADS; i++) {
		pthread_join(threads[i], NULL);
	}

	pthread_mutex_destroy(&lock);
	return 0;
}
```

```
void* runner(void* arg) {

	memory[counter] = *((int*)arg);

	printf("Wrote %d at index %d\n", *((int*)arg), counter);

	counter++;

	pthread_exit(0);

}
```

SER 334
Producer/Consumer

Producer/Consumer Visual

```
monitor class BoundedBuffer {
    //shared data
    int buffer[MAX];
    int fill, use;
    int fullEntries = 0;
    cond_t empty;
    cond_t full;

    void produce(int element) {
        if (fullEntries == MAX)
            wait(&empty);
        buffer[fill] = element;
        fill = (fill + 1) % MAX;
        fullEntries++;
        signal(&full);
    }

    int consume() {
        if (fullEntries == 0)
            wait(&full);
        int tmp = buffer[use];
        use = (use + 1) % MAX;
        fullEntries--;
        signal(&empty);
        return tmp;
     }
}
```

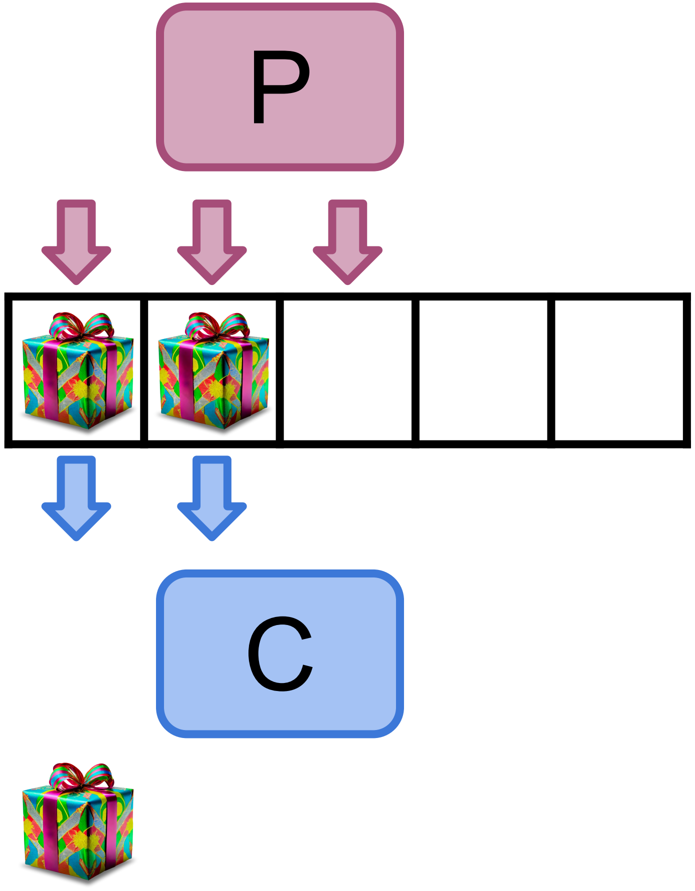
SER 334
Producer/Consumer

```
monitor class BoundedBuffer {
    //shared data
    int buffer[MAX];
    int fill, use;
    int fullEntries = 0;
    cond_t empty;
    cond_t full;

    void produce(int element) {
        if (fullEntries == MAX)
            wait(&empty);
        buffer[fill] = element;
        fill = (fill + 1) % MAX;
        fullEntries++;
        signal(&full);
    }

    int consume() {
        if (fullEntries == 0)
            wait(&full);
        int tmp = buffer[use];
        use = (use + 1) % MAX;
        fullEntries--;
        signal(&empty);
        return tmp;
     }
}
```

Producer/Consumer Visual



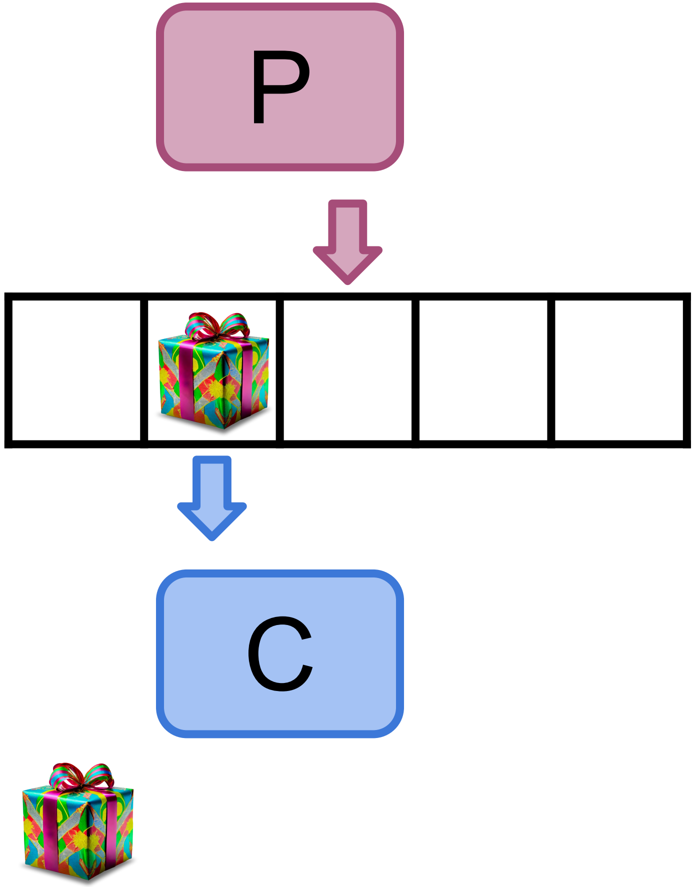
SER 334
Producer/Consumer

```
monitor class BoundedBuffer {
    //shared data
    int buffer[MAX];
    int fill, use;
    int fullEntries = 0;
    cond_t empty;
    cond_t full;

    void produce(int element) {
        if (fullEntries == MAX)
            wait(&empty);
        buffer[fill] = element;
        fill = (fill + 1) % MAX;
        fullEntries++;
        signal(&full);
    }

    int consume() {
        if (fullEntries == 0)
            wait(&full);
        int tmp = buffer[use];
        use = (use + 1) % MAX;
        fullEntries--;
        signal(&empty);
        return tmp;
     }
}
```

Producer/Consumer Visual



SER 334

Samples

4. [Acuña] Consider the following solution to the producer consumer problem from Silberschatz.
This solution to the producer consumer problem with a bounded buffer requires three semaphores - can the problem be solved with less? Explain.

```
//shared data. [Operating Systems Concepts by Silberschatz.]
int n;
//also a buffer data structure
semaphore buf_mutex = 1;
semaphore empty = n;
semaphore full = 0;
```

```
//producer
do {
	// produce next_produced
	wait(empty);
	wait(buf_mutex);
	// add next_produced to the buffer
	signal(buf_mutex);
	signal(full);
} while (true);
```

```
//consumer
do {
	wait(full);
	wait(buf_mutex);
	// move from buffer to next_consumed
	signal(buf_mutex);
	signal(empty);
	// consume next_consumed
}while (true);
```

SER 334
Samples

6. [Lisonbee] The bounded buffer problem is a result of producing and consuming work asynchronously at different and/or variable rates. Provided below is are the producer and consumer functions used by a program. Assume that the producer and consumer are running in parallel. In order to solve the bounded buffer problem demonstrated here, **the appropriate calls to wait and signal need to be added. Rewrite the above code using wait and signal and initialize the 3 semaphores to the appropriate values** (Note: you must use all three semaphores at least once in your calls to wait and signal).

```
int data[15], i0 = 0, i1 = 0;

semaphore mutex = ... ;
semaphore empty = ... ;
semaphore full = ... ;



void producer() {
    while (1) {

        data[i0] = i0 * i0;

        i0 = ++i0 % 15;

    }
}
```

```
void consumer() {
    while (1) {

        printf("%d\n", data[i1]);

        i1 = ++i1 % 15;

    }
}
```

When should you use process synchronization?

- Always
- For shared resources
- With Threads
- Never

SER 334

Concept Check

When using a lock (of any type), where should you place the *lock* and *unlock* calls?

Around the critical section

Before creating a thread

Within the runner

Before the fork

SER 334

Concept Check

Which of the following are most likely to be an atomic action?

int w = 5;	int x = 3 + 9;
int y = w + 7;	int z = x + y;

SER 334
Scratch Space

Upcoming Events

SI Sessions:

- ~~Sunday, February 11th at 7:00 pm MST~~ **Cancelled - Good luck on Exam 2!**
- Monday, February 12th at 7:00 pm MST

Review Sessions:

- Exam 2 Review: Thursday, February 8th 7:00 pm - 9:00 pm MST
- Exam 3 Review: TBD

Questions?

Survey:

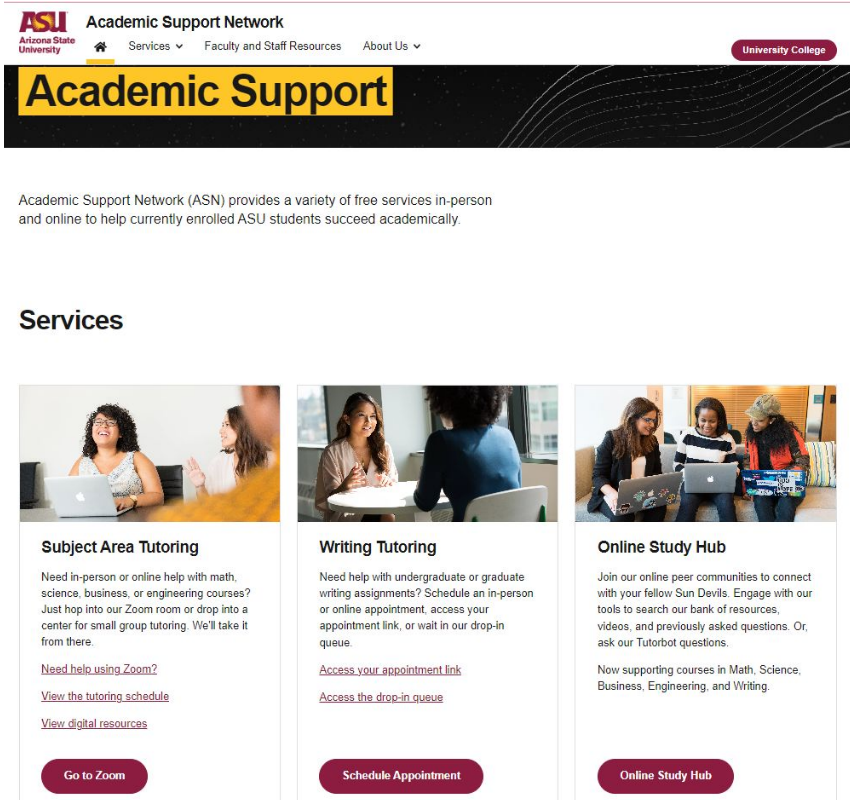
http://bit.ly/ASN2324

21

More Questions?

Check out our other resources!

tutoring.asu.edu

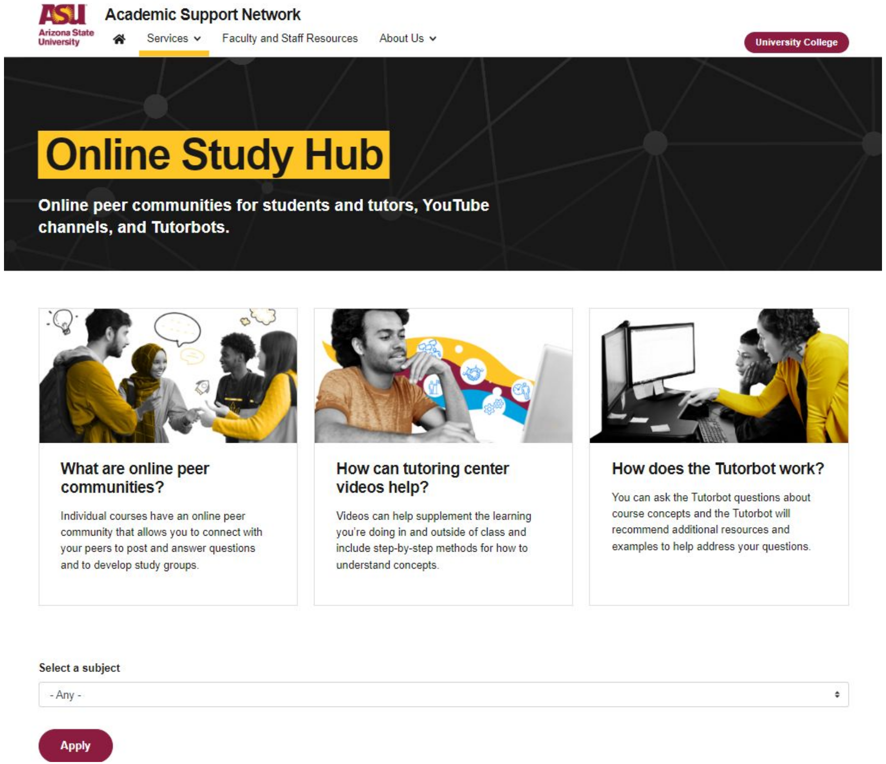


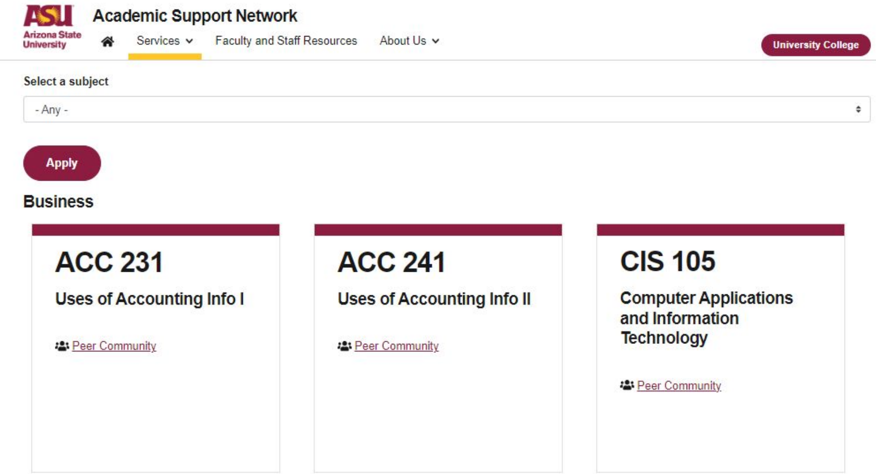
1. **Click on ‘Go to Zoom’ to log onto our Online Tutoring Center.**
2. **Click on ‘View the tutoring schedule’ to see when tutors are available for specific courses.**

More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub





Don't forget to check out the Online Study Hub for additional resources!

Additional Resources

- Course Repo
- Course Discord
- BMP File Format (Wiki)
- Linux Kernel API
- Bootlin - Linux Cross Referencer
- Dining Philosophers Interactive
- Producer/Consumer Visual