SER 334 A Session

Exam 2 Review Session

February 8th 2024

7:00 pm - 9:00 pm MST

Agenda

- Practice Exam
- Sample Problems
- Q&A

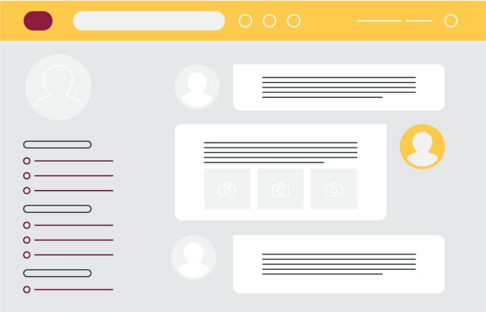
SI Session Expectations

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

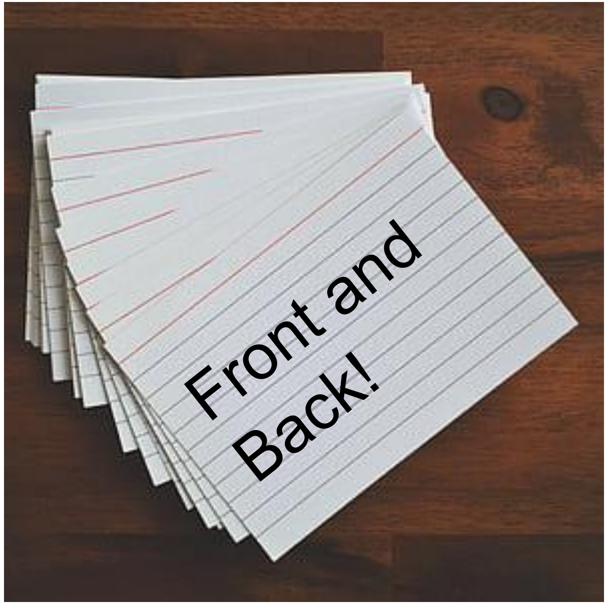
SER 334

Exam 2 Review

[Exam Info Page]

Opens: Saturday February 10th @ 2 AM

Closes: Sunday February 11th @ 11:59 PM



[Practice Exam]

75 minutes

~26 questions

Pro Tip: Take a peek at the programming questions before getting started

SER 334
Exam 2 Review

Processes

1. Imagine that you are trying to write a program that performs a data-intensive computation, and which needs to support some level of recoverability. Would you be better off using shared data or message passing? Explain. [5 points] [Acuña, Pilcher]

   (a) Shared data – because every thread has individual access to the memory

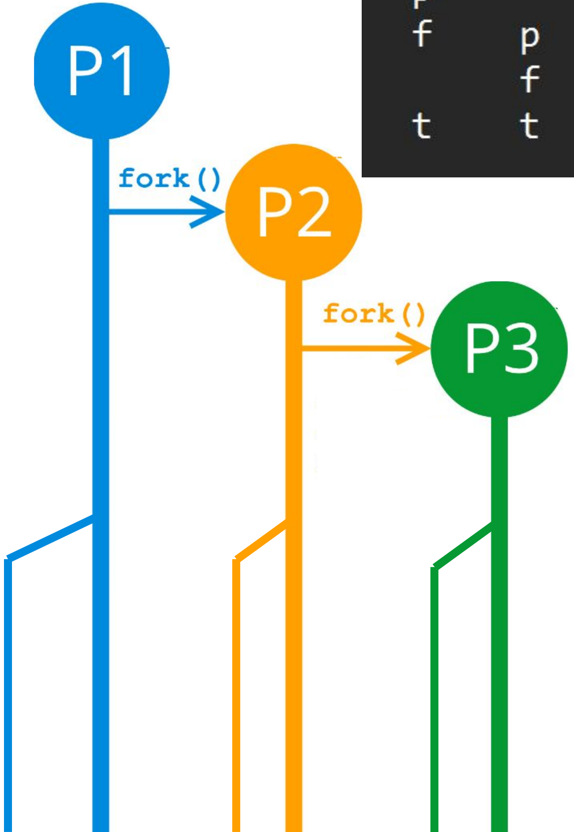
   (b) Message passing – because there will be a record of each message sent which can be saved

   (c) Shared data – because it would require less memory to be used than message passing

   (d) Message passing – because if a message gets corrupted, it only affects one other process

SER 334

Execution Tracing Techniques




```
p
f   p
    f   p
t   t   t
```

Any other suggestions or techniques for tracing during the exam?

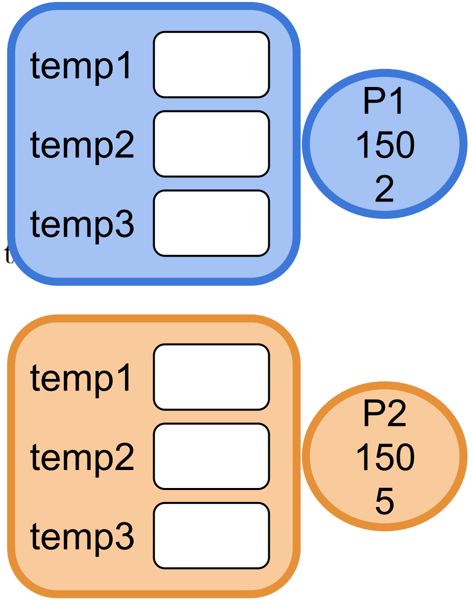
```
  P
  +---->P
  |     |
  |     +----->P
  |     |      |
  |     |      |
 /|    /|     /|
| |   | |    | |
```

ASCII Diagram

p		
fork	p	
	fork	p
thread	thread	thread

SER 334

Exam 2 Review



2. Trace the following program. As pids come into existence, assume they come from t
2600, 2601, 2602, 2603, and so on. [Acuña, Silberschatz]

```
int main() {
    pid_t pid, pid1;

    pid = fork();
    if (pid == 0) {
        pid1 = getpid();
        printf("child: pid = %d", pid); /* A */
        printf("child: pid1 = %d", pid1); /* B */
    }
    else {
        pid1 = getpid();
        printf("parent: pid = %d", pid); /* C */
        printf("parent: pid1 = %d", pid1); /* D */
        wait(NULL);
    }
    return 0;
}
```

From the program above, identify the values of the process ids at lines A, B, C, and D. [10 points]

SER 334

Exam 2 Review

3. Trace the previous program and then list all of the possible outputs (just write the A/B/C/D text; no **need** for pids) that could be generated. (Assume that fork will always succeed.) [10 points] [Acuña]

```
int main() {
    pid_t pid, pid1;

    pid = fork();
    if (pid == 0) {
        pid1 = getpid();
        printf("child: pid = %d", pid);  /* A */
        printf("child: pid1 = %d", pid1);  /* B */
    }
    else {
        pid1 = getpid();
        printf("parent: pid = %d", pid);  /* C */
        printf("parent: pid1 = %d", pid1);  /* D */
        wait(NULL);
    }
    return 0;
}
```

4. Consider the following program.

```
int main() {
    pid_t pid;

    pid = fork();
    if (pid == 0) {
        fork();
        thread_create(...);
    }
    thread_create(...);
    return 0;
}
```

How many unique threads are created in the program above? [5 points] [Acuña, Silberschatz]

(a) 2

(b) 3

(c) 4

(d) 5

SER 334

Exam 2 Review

5. Is it faster to switch between processes or threads within those processes? Explain. [5 points] [Acuña]

   (a) Processes - they are permanently assigned to a specific CPU so switching never occurs.

   (b) Processes - they have kernel level privileges to access the PCB handling functions.

   (c) Threads - they don't require the call stack to be saved/restored since the memory is shared among all threads for a given process.

   (d) Threads - they are light weight and don't require saving/restoring many resources.

SER 334

Exam 2 Review

6. Consider the task of implementing a mutex, specifically the unlock functionality (like is supported by "int pthread_mutex_unlock(pthread_mutex_t *mutex)"). Which data structure would you prefer to use to determine the next waiting process to wake up? [5 points] [Acuña]

   (a) Hash tables - it will mean it's O(1) to find the next process.

   (b) Binary Tree - it mimics the parent-child relationships that are created by calls to fork().

   (c) Stack - it dynamically resizes unlike the other data structures.

   (d) Queue - it would also enable the mutex to support bounded waiting time.

7. In pthreads, mutex support involves three functions and a new type:

```
typedef pthread_mutex_t
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

How does this functionality address (or does not address) the issues of Mutual Exclusion, Progress, and Bounded Waiting Time? [10 points] [Acuña]

SER 334

Exam 2 Review

8. The following code snippet models a user checking out a book from a library:

```
struct Book{
    char* title;
    int user;
    int is_checked_out;
}
void check_out(int userId, int book_num, Book* books){
    Book book = books[book_num];
    while(book.is_checked_out);
    book.is_checked_out = 1;
    book.user = userId;
    books[book_num] = book;
}
```

If this code were to be used in a multithreaded program (where multiple users/threads may try to check out books at the same time) would there be a race condition? If so, describe this race condition. Assume that the book being processed is initally not checked out. (The state is the contents of the books array after use of this function.) [10 points] [Edgar]

SER 334
Exam 2 Review

9. How does a monitor function differently from a semaphore? [5 points] [Pilcher]

   (a) a monitor only allows one process to be in its queue at a time while a semaphore allows multiple processes

   (b) a monitor and a semaphore function the same except a monitor is represented as a class

   (c) a monitor allows a set number of processes to be active at a time while a semaphore only allows one process to be active at a time

   (d) a monitor allows one process to be active at a time while a semaphore can allow multiple processes to be active at the same time

SER 334
Exam 2 Review

10. Consider the semaphore based solution we had for the reader-writer problem:

```
semaphore rw_mutex = 1;
semaphore rc_mutex = 1;
int read_count = 0;

//writer process
do {
    wait(rw_mutex);
    /* writing is performed */
    signal(rw_mutex);
} while (true);

//reader process
do {
    wait(rc_mutex);
    read_count++;
    if (read_count == 1)
        wait(rw_mutex);
    signal(rc_mutex);
    /* reading is performed */
    wait(rc_mutex);
    read_count--;
    if (read_count == 0)
        signal(rw_mutex);
    signal(rc_mutex);
} while (true);
```

What functionality does the logic involving read_count and rc_mutex provide? [5 points] [Acuña, Pilcher]

(a) It guarantees there is never a negative amount of readers

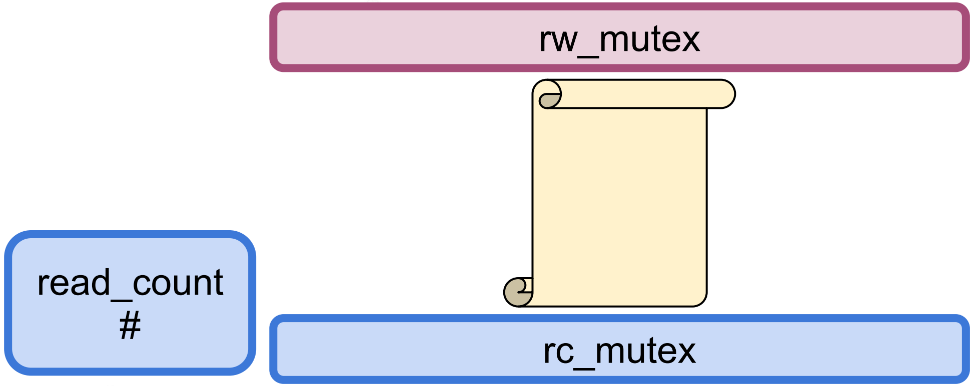
(b) It ensures there is only one writer process at one time

(c) It allows multiple processes to read at the same time

(d) It ensures only one process can read at a time

SER 334
Exam 2 Review

10. Consider the semaphore based solution we had for the reader-writer problem:



What functionality does the logic involving read_count and rc_mutex provide? [5 points] [Acuña, Pilcher]

(a) It guarantees there is never a negative amount of readers

(b) It ensures there is only one writer process at one time

(c) It allows multiple processes to read at the same time

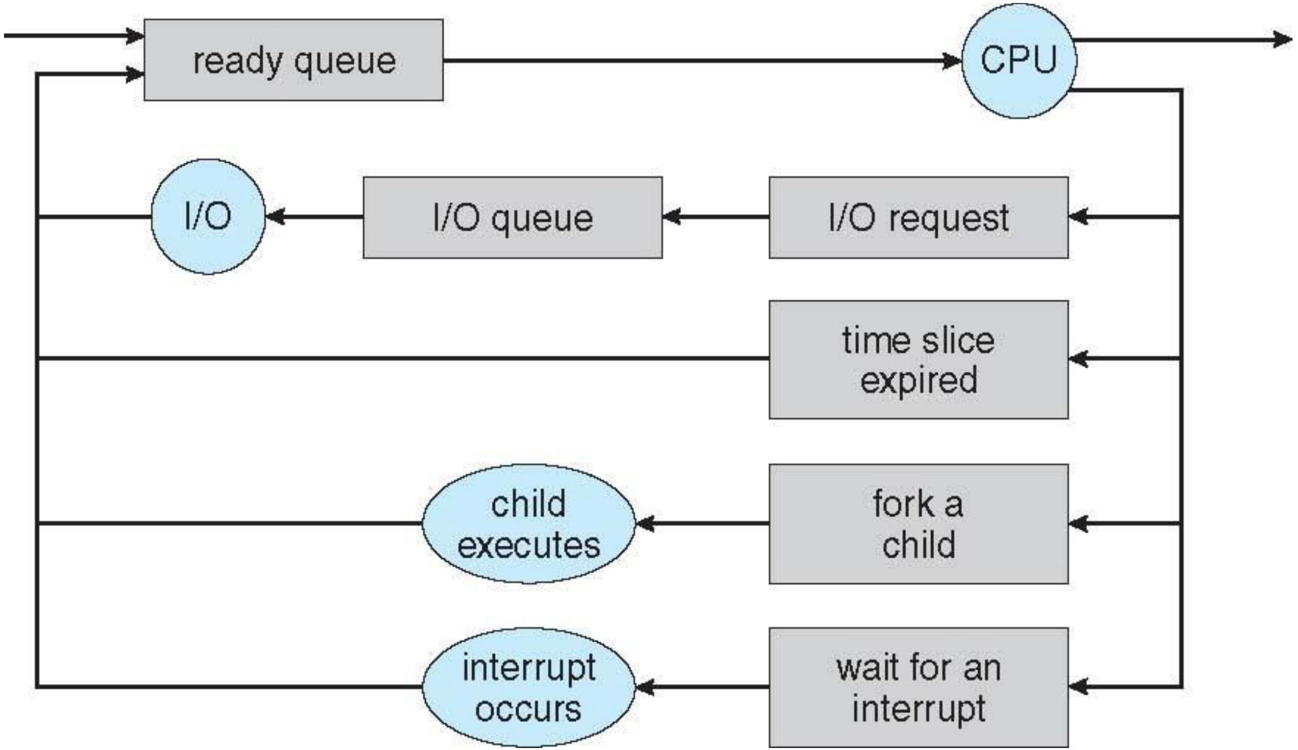
(d) It ensures only one process can read at a time

```
semaphore rw_mutex = 1;
semaphore rc_mutex = 1;
int read_count = 0;

//writer process
do {
    wait(rw_mutex);
    /* writing is performed */
    signal(rw_mutex);
} while (true);

//reader process
do {
    wait(rc_mutex);
    read_count++;
    if (read_count == 1)
        wait(rw_mutex);
    signal(rc_mutex);
    /* reading is performed */
    wait(rc_mutex);
    read_count--;
    if (read_count == 0)
        signal(rw_mutex);
    signal(rc_mutex);
} while (true);
```

Module 5 Sample

SER 334
Exam 2 Review

3. [Bahremand] Explain why the queue diagram in slide 9 has a continuous cycle that flows between the listed queues and resources? Is a cycle necessary? [2 points]



SER 334

Exam 2 Review

5. [Acuña] Consider the following program written with the pthreads library: What is this programming trying to compute and how does it uses threads?

```
#include <pthread.h>
#include <stdio.h>

int t1, t2, input;

void* runner(void *param) {
    int result, upper = (int)param;
    int a = 1, b = 1, c = 1;

    for (int i = 2; i < input-upper; i++) {
        c = a + b;
        a = b;
        b = c;
    }

    if(upper == 1) {
        t1 = c;
    } else {
        t2 = c;
    }
    pthread_exit(0);
}
```

```
int main(int argc, char *argv[]) {
    pthread_t tid1, tid2;
    pthread_attr_t attr;
    input = 10;

    pthread_attr_init(&attr);
    pthread_create(&tid1, &attr, runner, 1);
    pthread_create(&tid2, &attr, runner, 2);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    printf("result = %d\n", t1+t2);
}
```

SER 334

Exam 2 Review

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
Exam 2 Review

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;

    }
}
```

7. [Acuña] Consider the monitor based solution we had for the dining philosophers problem:

```
monitor DiningPhilosophers {
        enum { THINKING; HUNGRY, EATING) state[5];
        cond_t self[5];
        void pickup(int i) {
            state[i] = HUNGRY;
            test(i);
            if (state[i] != EATING)
                self[i].wait();
        }
        void putdown(int i) {
            state[i] = THINKING;
            // test left and right neighbors
            test((i + 4) % 5);
            test((i + 1) % 5);
        }
        void test(int i) {
            if ((state[(i + 4) % 5] != EATING) &&        //AND --> OR
                (state[i] == HUNGRY) &&                  //AND --> OR
                (state[(i + 1) % 5] != EATING)) {
                    state[i] = EATING;
                    self[i].signal();
            }
        }
        void initialization_code() {
            for (int i = 0; i < 5; i++)
                state[i] = THINKING;
        }
}
```

How would the functionality of this change if the conditional in test() used || instead of &&?

SER 334
Exam 2 Review

2. [Acuña] **Implement** the wait() operation for a semaphore using the test_and_set() instruction. Assume that you have two methods available: add_this_process(list) which adds the current process to a list, and block() which pauses the current process until it is signaled.

```
struct semaphore* S {
    static boolean lock;
    struct process_node* list;
    int value;
}
void wait(semaphore* S) {
```

SER 334

Exam 2 Review

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 3 Review: TBD

Good luck on the Exam
You got this!

Questions?

Survey:

http://bit.ly/ASN2324

27

More Questions?

Check out our other resources!

tutoring.asu.edu

ASU Arizona State University

Academic Support Network

Services ⌄ Faculty and Staff Resources About Us ⌄

University College

Academic Support

Academic Support Network (ASN) provides a variety of free services in-person and online to help currently enrolled ASU students succeed academically.

Services

Subject Area Tutoring

Need in-person or online help with math, science, business, or engineering courses? Just hop into our Zoom room or drop into a center for small group tutoring. We'll take it from there.

Need help using Zoom?

View the tutoring schedule

View digital resources

Go to Zoom

Writing Tutoring

Need help with undergraduate or graduate writing assignments? Schedule an in-person or online appointment, access your appointment link, or wait in our drop-in queue.

Access your appointment link

Access the drop-in queue

Schedule Appointment

Online Study Hub

Join our online peer communities to connect with your fellow Sun Devils. Engage with our tools to search our bank of resources, videos, and previously asked questions. Or, ask our Tutorbot questions.

Now supporting courses in Math, Science, Business, Engineering, and Writing.

Online Study Hub

1 – Go to Zoom

Need help using Zoom?

2– View the tutoring schedule

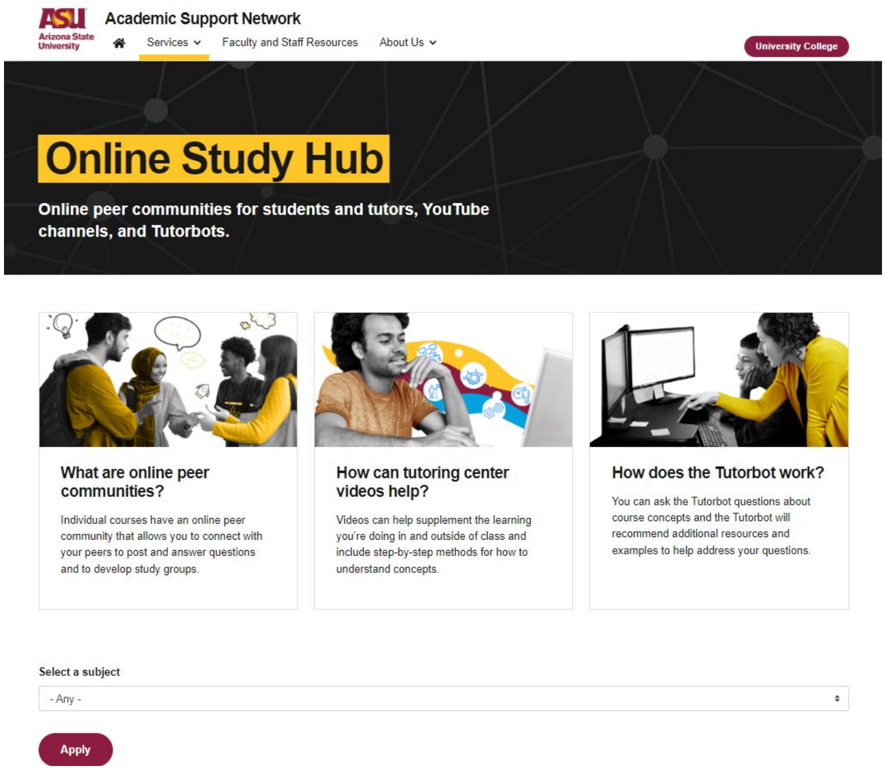
View digital resources

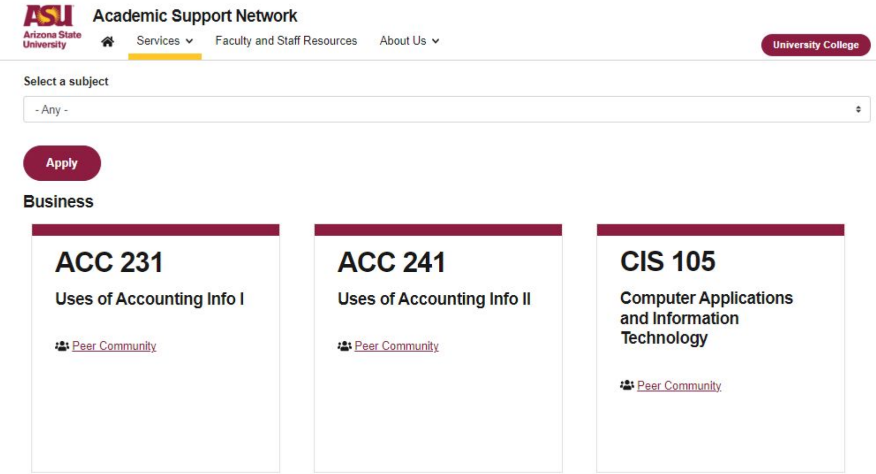
1. Click on ‘Go to Zoom’ to log onto our Online Tutoring Center.
2. Click on ‘View the tutoring schedule’ to see when tutors are available for specific courses.

More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub

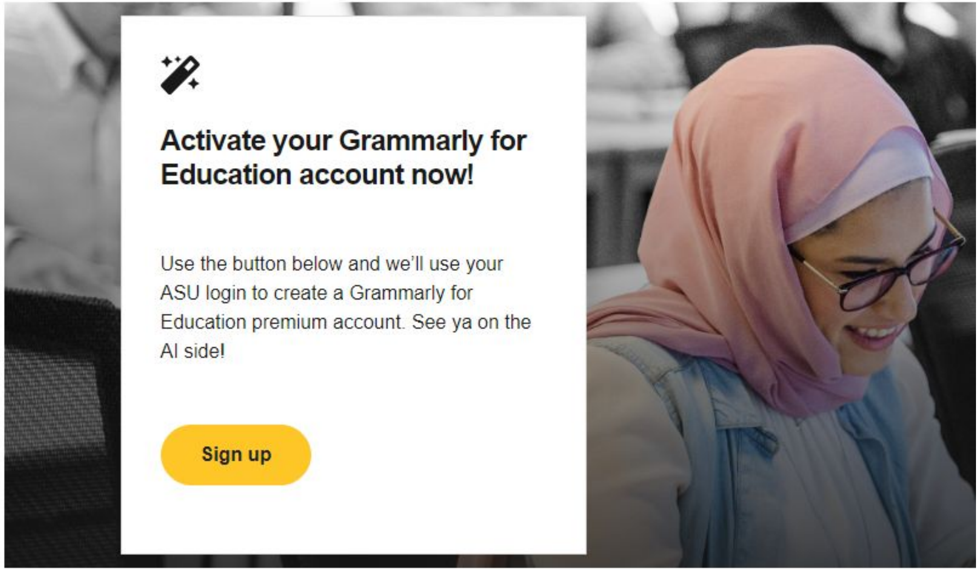




Don’t forget to check out the Online Study Hub for additional resources!

Expanded Writing Support Available

Including Grammarly for Education, at no cost!



tutoring.asu.edu/expanded-writing-support

*Available slots for this pilot are limited

Additional Resources

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