[redacted]

Question 1

1 / 1 pts

Concurrency means multiple jobs running at once, while parallelism means multiple jobs can run simultaneously in parallel.

- (●) True
- ( ) False

Question 2

3 / 3 pts

Which of the following statements is **false**?

- ( ) Process scheduling is an OS activity that schedules processes in different states
- (●) OS scheduler stops running when there is no runnable process in the ready or waiting queue
- ( ) OS implements various policies for scheduling multiple processes
- ( ) Context switching can happen in two cases: a process yields a CPU / OS receives an interrupt (an external event)

Context switching is an OS's process of storing the current process and restoring another process

Question 3

1 / 1 pts

In Linux, processes in the ready/waiting states are stored in a queue data structure

- (selected) True
- False

Question 4

1 / 1 pts

In Linux, when a CPU is available, the kernel chooses the next running process from the ready queue in a FIFO manner.

- True
- (selected) False

Question 5

3 / 3 pts

Consider the following code snippet:

```
static int tot_balance = 400;

void *deposit_cash(void *args)
{
```

```
    int balance = tot_balance;
    int *money  = (int *) args;
    printf("[Thread] load: %d, deposit: %d\n", balance, *money);

    sleep(0.1);

    balance += *(money);
    tot_balance = balance;
    printf("[Thread] final: %d\n", tot_balance);
}

int main()
{
    int i;
    int cash[2] = { 100, 200 };
    pthread_t tid[2];

    for (i = 0; i < 2; i++)
        pthread_create(&tid[i], NULL, deposit_cash, (void *)&cash
[i]);

    for (i = 0; i < 2; i++)
        pthread_join(tid[i], NULL);

    printf("[Main] total: %d\n", tot_balance);
    return 0;
}
```

What are the possible balance values of this account in the end (choose all that apply)?

- [ ] 400
- [ ] 800
- [x] 700
- [x] 500
- [x] 600

Question 6

2 / 2 pts

What is **true** about the synchronization in OS?

- [ ] Race condition always occurs when there is a shared resource

To protect a critical section, we need at least two mutex variable

The critical section is a sequence of instructions that can cause race conditions

Atomic operation means a sequence of instructions should be run at once

Any process waiting for the "mutex unlock" can run as soon as the mutex is unlocked

Question 7

4 / 4 pts

Consider the following code snippet:

```
#define MACHINE_CAPACITY    64

sem_t mutex, slots_filled, slots_empty;

struct machine{ int nitems; };

double _sleep_time() { return 2 * ((double) rand() / (double) RAN
D_MAX); }

void *producer_fn(void *args) {
    int cokes;
    struct machine *cur_machine = (struct machine *) args;

    while (1) {
        sem_wait(&slots_empty);
        sem_wait(&mutex);

        cokes = cur_machine->nitems;
        cur_machine->nitems += 1;
        printf("[Producer] enqueue cokes %d -> %d\n", cokes, cur_
machine->nitems);

        sem_post(&mutex);
        sem_post(&slots_filled);
        sleep(_sleep_time());
    }
}

void *consumer_fn(void *args) {
    int cokes;
    struct machine *cur_machine = (struct machine *) args;
```

```
    while (1) {
        sem_wait(&slots_filled);
        sem_wait(&mutex);

        cokes = cur_machine->nitems;
        cur_machine->nitems -= 1;
        printf("[Consumer] dequeue cokes %d -> %d\n", cokes, cur_
machine->nitems);

        sem_post(&mutex);
        sem_post(&slots_empty);
        sleep(_sleep_time());
    }
}

int main(void) {
    struct machine coke_machine = { .nitems = 0 };
    srand (time(NULL));

    sem_init(&mutex, 0, 1);
    sem_init(&slots_empty, 0, MACHINE_CAPACITY);
    sem_init(&slots_filled, 0, 0);

    pthread_t producer, consumer;
    pthread_create(&producer, NULL, producer_fn, (void *)&coke_ma
chine);
    pthread_create(&consumer, NULL, consumer_fn, (void *)&coke_ma
chine);

    pthread_join(producer, NULL);
    pthread_join(consumer, NULL);

    return 0; // code only reaches here if the machine is broken
}
```

Which of the following statements is **false**?

- In the producer_fn, swapping sem_wait(&mutex) and sem_wait(&slots_empty) code lines causes the deadlock problem.
- This code runs without data-race or deadlock problems
- **In the producer_fn, swapping sem_post(&slots_filled) and sem_post(&mutex) code lines causes deadlock.**
- Removing slots_filled and slots_empty semaphores causes the data-race problem

- Removing mutex semaphore causes the data-race problem

Question 8

1 / 1 pts

Deadlock is a situation in which no thread can continue execution because of the locks

- (selected) True
- False

Question 9

2 / 2 pts

Consider a program using a counting semaphore and more than one thread is blocked on sem_wait(). Which of the following statements is **true**?

- The thread blocked on sem_wait() for the shortest time will be executed.
- All the threads on sem_wait() will be unblocked and start execution.
- The thread blocked on sem_wait() for the longest time will be executed.
- (selected) Any arbitrary thread blocked on sem_wait() will be executed.
- The thread that has the shortest code, blocked on sem_wait(), will be executed.

Question 10

1 / 1 pts

The monitor is an object that OS natively supports as a synchronization primitive.

- ○ True
- ◉ False

Question 11

1 / 1 pts

Rust checks memory safety and data-race freedom in a compilation time

- ◉ True
- ○ False

Question 12

2 / 2 pts

Which of the following is **false** in Rust?

- ○ A Rust array “arr” can be accessed by bash-style indexing, i.e., arr[1 .. 10]
- ○ If we call a function using borrowing, the variable is immutable inside the function

If we call a function with a variable, the ownership of the var. transfers to the function

Once initialized, we cannot modify “immutable” variable values

Once initialized, we can assign different types to “mutable” variable

Question 13

1 / 1 pts

In Rust, there is no way to compile source code that breaks its safety guarantees

True

False

Question 14

3 / 3 pts

Consider the following code snippet.

```
fn take(vec: Vec<String>) {
    println!("{:?}", vec);
}

fn main() {
    let vec = Vec::new();
    vec.push(String::from("Hello "));
    vec.push(String::from("World "));

    take(vec);
    vec.push(String::from("from the other side!"))
}
```

Which of the following statements is true (choose all that apply)?

The code prints out [“Hello ”, “World ”]

- [ ] The code compiles correctly if we make “vec” mutable.
- [x] In the current code, the ownership of “vec” transfers after calling the “fn take”
- [ ] The code compiles correctly if we fix it by making “fn take(vec: &Vec<string>) {...”
- [x] The code returns a compilation error.

Question 15

4 / 4 pts

Consider the following code snippet:

```
use std::thread;
use std::sync::{Arc,Mutex};

fn main() {
    let balance = Arc::new(Mutex::new(200));
    let mut threads = vec![];

    let balance4deposit = Arc::clone(&balance);
    threads.push(thread::spawn(move || {
        let mut new_balance = balance4deposit.lock().unwrap();
        *new_balance += 100;
        println!("Increase the balance {}", new_balance);
    }));

    let balance4withdrawal = Arc::clone(&balance);
    threads.push(thread::spawn(move || {
        let mut new_balance = balance4withdrawal.lock().unwrap();
        *new_balance -= 300;
        println!("Decrease the balance {}", new_balance);
    }));

    for thread in threads {
        let _ = thread.join();
    }
    println!("Final balance {}", *balance.lock().unwrap());
}
```

Which of the following statements is **false**?

- The new_balance in threads can be modified without lock() statements, as the lock() only guarantees data-race freedom.
- The “Arc” makes the variable accessible within the thread
- The code won’t print the final balance until all the threads join
- The code compiles correctly
- The “Mutex” makes the variable accessible and mutable within the thread

Question 16

Not yet graded / 1 pts

(A gift from Sanghyun to you) Having finished the exam, how do you feel about it (choose all numbers that apply)?

1. 😊
2. 😄
3. 😎
4. 😍
5. 😑
6. 😢
7. 😭
8. 😰
9. 🤮
10. 😴

Your Answer:

1. 😊
2. 😄
3. 😎
4. 😍

Question 17

Not yet graded / 1 pts

(A gift from Sanghyun to you) Is there anything you'd like to tell the course staff (such as feedback about the class, programming assignments, or exam, suspicious activity during the exam, extra credit assignments you suggest, etc.)?

Your Answer:

I wish there would be some in-person office hours

Quiz Score: **30** out of 32