CS433 Written Homework 4

(80 points) All question numbers refer to exercises in the textbook (10th edition). Make sure you use the right textbook and answer the right questions. Students must finish written questions individually. Type your answers and necessary steps clearly.

1. (5 points) 6.8 Race conditions are possible in many computer systems. Consider an online auction system where the current highest bid for each item must be maintained. A person who wishes to bid on an item calls the bid(amount) function, which compares the amount being bid to the current highest bid. If the amount exceeds the current highest bid, the highest bid is set to the new amount. This is illustrated below:

```
void bid(double amount) {
  if (amount > highestBid)
      highestBid = amount;
}
```

Describe how a race condition is possible in this situation and what might be done to prevent the race condition from occurring.

2. (5 points) 6.10 The `compare_and_swap()` instruction can be used to design lock-free data structures such as stacks, queues, and lists. The program example shown in Figure E6.18 presents a possible solution to a lock-free stack using CAS instructions, where the stack is represented as a linked list of `Node` elements with `top` representing the top of the stack. Is this implementation free from race conditions?

```
typedef struct node {
  value_t data;
  struct node *next;
} Node;

Node *top; // top of stack

void push(value_t item) {
  Node *old_node;
  Node *new_node;

  new_node = malloc(sizeof(Node));
  new_node->data = item;

  do {
    old_node = top;
    new_node->next = old_node;
  }
  while (compare_and_swap(top,old_node,new_node) != old_node);
}

value_t pop() {
  Node *old_node;
  Node *new_node;

  do {
    old_node = top;
    if (old_node == NULL)
      return NULL;
    new_node = old_node->next;
  }
  while (compare_and_swap(top,old_node,new_node) != old_node);

  return old_node->data;
}
```

3. (5 points) 6.12 Some semaphore implementations provide a function `getValue()` that returns the current value of a semaphore. This function may, for instance, be invoked prior to calling `wait()` so that a process will only call `wait()` if the value of the semaphore is > 0, thereby preventing blocking while waiting for the semaphore. For example:

```
  if (getValue(&sem) gt; 0)
    wait(&sem);
```

Many developers argue against such a function and discourage its use. Describe a potential problem that could occur when using the function `getValue()` in this scenario.

4. (5 points) 6.15 Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs.

5. (5 points) 6.19 Assume that a system has multiple processing cores. For each of the following scenarios, describe which is a better locking mechanism—a spinlock or a mutex lock where waiting processes sleep while waiting for the lock to become available:

- The lock is to be held for a short duration.
- The lock is to be held for a long duration.
- The thread may be put to sleep while holding the lock.

6. (10 points) 6.22 Consider the code example for allocating and releasing processes shown in Figure E6.21.

```
#define MAX_PROCESSES 255
int number_of_processes = 0;

/* the implementation of fork() calls this function */
int allocate_process() {
int new_pid;
```

```
      if (number_of_processes == MAX_PROCESSES)
         return -1;
      else {
         /* allocate necessary process resources */
         ++number_of_processes;

         return new_pid;
      }
    }
    /* the implementation of exit() calls this function */
    void release_process() {
      /* release process resources */
      --number_of_processes;
    }
```

---

a. Identify the race condition(s).

b. Assume you have a mutex lock named `mutex` with the operations `acquire()` and `release()`. Indicate where the locking needs to be placed to prevent the race condition(s).

c. Could we replace the integer variable

```
    int number_of_processes = 0
```

with the atomic integer

```
    atomic_t number_of_processes = 0}
```

to prevent the race condition(s)?

7. (5 points) 6.23 Servers can be designed to limit the number of open connections. For example, a server may wish to have only N socket connections at any point in time. As soon as N connections are made, the server will not accept another incoming connection until an existing connection is released. Illustrate how semaphores can be used by a server to limit the number of concurrent connections.

8. (5 points) 6.26 Describe how the `signal()` operation associated with monitors differs from the corresponding operation defined for semaphores.

9. (5 points) 7.8 The Linux kernel has a policy that a process cannot hold a spinlock while attempting to acquire a semaphore. Explain why this policy is in place.

10. (10 points) 8.18 Which of the six resource-allocation graphs shown in Figure E8.15 illustrate deadlock? For those situations that are deadlocked, provide the cycle of threads and resources. Where there is not a deadlock situation, illustrate the order in which the threads may complete execution.

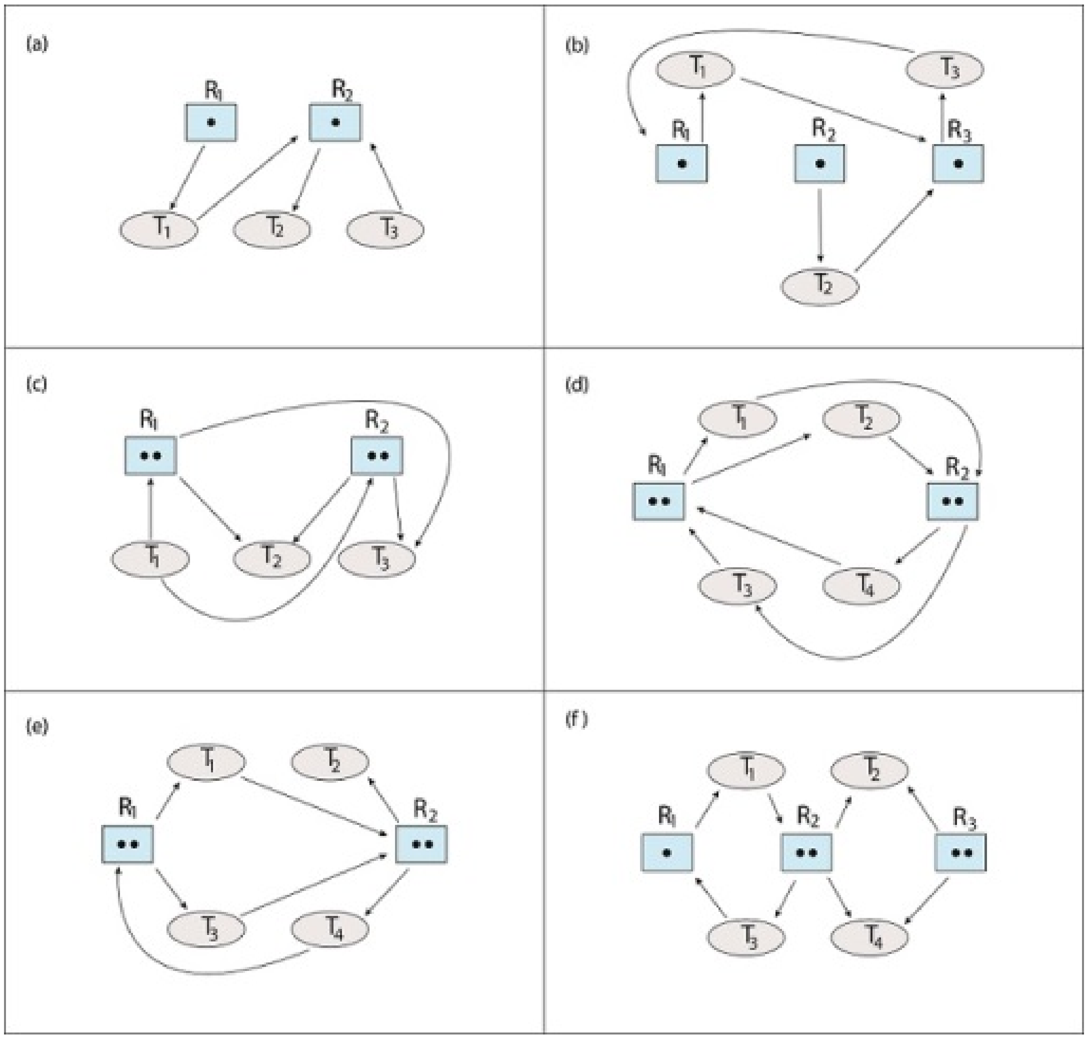

Figure E8.15 Resource-allocation graphs for Exercise 8.18.

11. (10 points) 8.27 Consider the following snapshot of a system:

	Allocation	***Max***
	A B C D	*A B C D*
T_0	1 2 0 2	4 3 1 6
T_1	0 1 1 2	2 4 2 4
T_2	1 2 4 0	3 6 5 1
T_3	1 2 0 1	2 6 2 3
T_4	1 0 0 1	3 1 1 2

a. Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.
b. a. ***Available*** = (2,2,2,3)
c. b. ***Available*** = (4,4,1,1)
d. c. ***Available*** = (3,0,1,4)
e. d. ***Available*** = (1,5,2,2)

12. (10 points) 8.28 Consider the following snapshot of a system:

	Allocation	***Max***	***Available***
	A B C D	*A B C D*	*A B C D*
T_0	3 1 4 1	6 4 7 3	2 2 2 4
T_1	2 1 0 2	4 2 3 2	
T_2	2 4 1 3	2 5 3 3	
T_3	4 1 1 0	6 3 3 2	
T_4	2 2 2 1	5 6 7 5	

Answer the following questions using the banker's algorithm:

a. Illustrate that the system is in a safe state by demonstrating an order in which the threads may complete.

b. If a request from thread T_4 arrives for (2,2,2,4), can the request be granted immediately?
c. If a request from thread T_2 arrives for (0,1,1,0), can the request be granted immediately?
d. If a request from thread T_3 arrives for (2,2,1,2), can the request be granted immediately?