**DM510 Operating Systems**
Spring 2026

---

# Exercise Sheet 5

Complete before tutorial on Thursday, March 12th

---

## Learning goals

Be able to

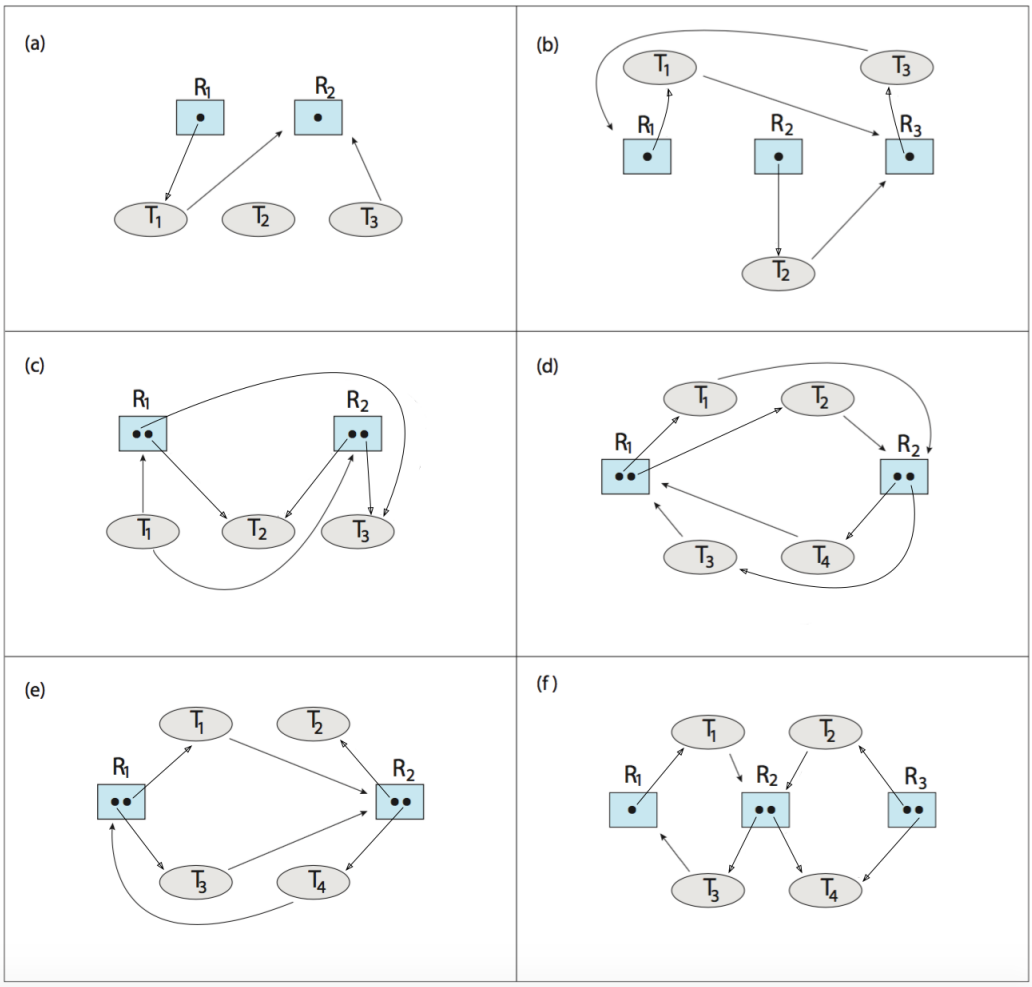
- identify deadlocks in code and in resource-allocation graphs
- describe the role of safe states in Banker's algorithm test whether a state is safe
- apply Banker's algorithm to determine if a request can be granted
- describe different variants of I/O
- describe the role of device registers and how they are used with memory-mapped I/O

## Chapter 8

**Exercise 1.** In a Mexican standoff, a number of participants point firearms at each other. Each participant can shoot another participant, lower their weapon, or wait. Compare this situation with the deadlocks considered in the lecture.

**Exercise 2.** Is it possible to have a deadlock involving only one single-threaded process? Explain your answer.

**Exercise 3.** Which of the six resource-allocation graphs shown below show a deadlock? Run the algorithm for deadlock detection from the lecture on (c). The others you may also intuitively answer. For those situations that are deadlocked, provide the cycle of threads and resources. In case of no deadlock, illustrate the order in which the threads may complete execution




**Exercise 4.** Consider the following source code:

```
/* thread_one runs in this function */
void do_work_one(void *param)
{
        pthread_mutex_lock(&first_mutex);
        pthread_mutex_lock(&second_mutex);
        /**
          * Do some work
        */
        pthread_mutex_unlock(&second_mutex);
        pthread_mutex_unlock(&first_mutex);
}
```

```
/* thread_two runs in this function */
void do_work_two(void *param)
{
        pthread_mutex_lock(&second_mutex);
        pthread_mutex_lock(&first_mutex);
        /**
          * Do some work
        */
        pthread_mutex_unlock(&first_mutex);
        pthread_mutex_unlock(&second_mutex);
}
```

a. Draw the resource-allocation graph that illustrates deadlock that could occur.

b. The program doesn't always lead to deadlock. Describe what role the CPU scheduler plays in this program. Would a critical section around the two locking instructions guarantee deadlock-freeness?

c. Suggest an alternative approach to avoiding the potential deadlock here and compare with the previous one.

**Exercise 5.** Consider the following snapshot of a system:

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

a. Is the system in a safe state?

b. If a request from thread T1 arrives for $(0, 4, 2, 0)$, can the request be granted immediately?

**Exercise 6.** Give an example of a system in an unsafe state (according to definition from lectures) where it is possible for the threads to complete their execution without entering a deadlocked state.

**Exercise 7** (Optional, for additional practice of Banker's algorithm)**.** 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 |

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.

- Available = $(2, 2, 2, 3)$
- Available = $(4, 4, 1, 1)$
- Available = $(3, 0, 1, 4)$
- Available = $(1, 5, 2, 2)$

# Chapter 12

**Exercise 8.** Discuss advantages and disadvantages of placing functionality in a device controller, a driver, or a user program. For example, a network controller could implement large parts of the TCP/IP protocol or it could only expose the physical bit stream and let the CPU implement the protocol.

**Exercise 9.** Why might a system use interrupt-driven I/O to manage a single serial port that connects to a keyboard and polling I/O to manage incoming network traffic?

**Exercise 10.** Describe circumstances under which blocking I/O should be used. Describe circumstances under which nonblocking I/O should be used. Why not just implement nonblocking I/O and have processes busy-wait until their devices are ready?

**Exercise 11.** Describe which registers the GPIO controller of the Raspberry Pi has and what their role is. For this, look up Chapter 6.1 of the peripherals documentation for the Raspberry Pi: https://pip-assets.raspberrypi.com/categories/579-raspberry-pi-zero/documents/RP-008249-DS-1-bcm2835-peripherals.pdf?disposition=inline
How does the CPU access these registers?