1. Consider the following snapshot of a system:

	Allocation	***Max***	***Available***
	A B C D	*A B C D*	*A B C D*
*P*0	2 0 0 1	4 2 1 2	3 3 2 1
*P*1	3 1 2 1	5 2 5 2	
*P*2	2 1 0 3	2 3 1 6	
*P*3	1 3 1 2	1 4 2 4	
*P*4	1 4 3 2	3 6 6 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 processes may complete.

b. If a request from process *P*1 arrives for (1, 1, 0, 0), can the request be granted immediately?

c. If a request from process *P*4 arrives for (0, 0, 2, 0), can the request be granted immediately?

Answer:

The values of Need for processes P0 through P4 respectively are (2, 2, 1, 1), (2, 1, 3, 1), (0, 2, 1, 3), (0, 1, 1, 2), and (2, 2, 3, 3).

a. With **Available** being equal to (3, 3, 2, 1) only process P0 could run. Once process P0 runs, it releases its resources (2, 0, 0, 1) which are added to the **Available.**

With **Available** being equal to (5, 3, 2, 2), only process P3 could run. Once process P3 runs, it releases its resources (1, 3, 1, 2) which are added to the **Available.**

Available is (6, 6, 3, 4) which allows all other existing processes to run.

The system is in a safe state.

Any order that starts with <P0, P3, …> can complete. One ordering of processes that can finish is <P0, P3, P1, P2, P4>

b. The request can be granted immediately?

This reduces in the value of **Available** to (2, 2, 2, 1). With the addition of (1, 1, 0, 0) P1's **Allocation** is (4,2, 2, 1) and **Need** reduced to (1, 0, 3, 1)

Available is (2, 2, 2, 1). Values of **Need** for processes P0 through P4 respectively are (2, 2, 1, 1), (1, 0, 3, 1), (0, 2, 1, 3), (0, 1, 1, 2), and (2, 2, 3, 3). Follow the instructions in part (a) to complete. Answer is yes, if you get a safe sequence as shown above.

c. Similar to part b. Follow the same steps.

2. Assume there are three resources, R1, R2, and R3 that are each assigned unique integer values 15, 10, and 25, respectively. What is a resource ordering which prevents a circular wait?

   Follow the integer order. Answer is R2, R1, and R3

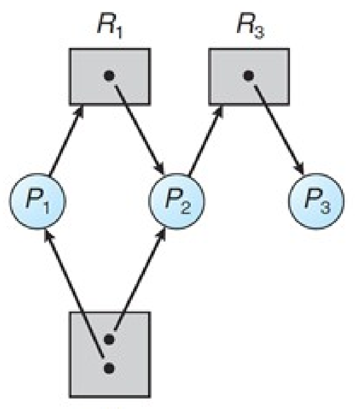
3. A resource-allocation graph is shown in Figure (a).
   a. Is the system deadlocked? Why or why not?

      No. When P3 finishes R3 will be available for P2. All processes can finish in the order P3, P2, & P1

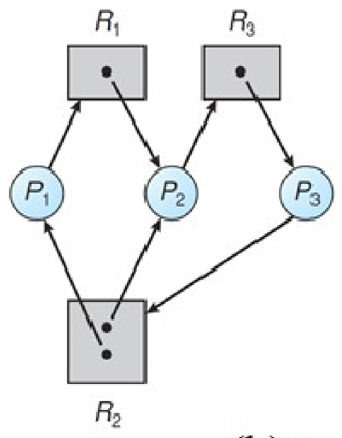
   Suppose that process P3 requests an instance of resource type R2. We add a request edge P3→ R2 to the graph as shown Figure (b).

   b. Is the system deadlocked? Why or why not?

      Yes. P1 is waiting for R1 held by P2, which is waiting for R3 held by P3, which in turn is waiting for R2 form either P1 or P2



(a)



(b)

4. Answers to all the rest of questions are directly in the slides given.

Chapter 9 Main Memory

Questions:

1. Name two differences between logical and physical addresses.
2. Why are page sizes always powers of 2?
3. Consider a logical address space of 64 pages of 1,024 words each, mapped onto a physical memory of 32 frames.
    a. How many bits are there in the logical address?
    b. How many bits are there in the physical address?
4. Explain the difference between internal and external fragmentation.
5. Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)? Rank the algorithms in terms of how efficiently they use memory.
6. Explain why mobile operating systems such as iOS and Android do not support swapping.
7. Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers)?
    a. 308
    b. 42095
    c. 215201
    d. 650000
    e. 2000001
8. The BTV operating system has a 21-bit virtual address, yet on certain embedded devices, it has only a 16-bit physical address. It also has a 2-KB page size.
    a. How many entries are there in a single-level page table?
    b. What is the maximum amount of physical memory?
9. Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.
    a. How many bits are required in the logical address?
    b. How many bits are required in the physical address?
10. Consider a computer system with a 32-bit logical address and 4-KB page size. The system supports up to 512 MB of physical memory.
    How many entries are there in a single-level page table?
11. What is the purpose of paging the page tables?
12. On a system with paging, a process cannot access memory that it does not own. Why? (hint: page table)

Solutions:

1. A logical address does not refer to an actual existing address; rather, it refers to an abstract address in an abstract address space. Contrast this with a physical address that refers to an actual physical address in memory. A logical address is generated by

the CPU and is translated into a physical address by the memory management unit (MMU). Therefore, physical addresses are generated by the MMU.

2. Recall that paging is implemented by breaking up an address into a page and offset number. It is most efficient to break the address into X page bits and Y offset bits, rather than perform arithmetic on the address to calculate the page number and offset. Because each bit position represents a power of 2, splitting an address between bits results in a page size that is a power of 2.
3. Consider a logical address space of 64 pages of 1024 words each, mapped onto a physical memory of 32 frames.

    a. Logical address:
        Number of pages = 64 = 2^6
        Words in each page = 1024 = 2^{10}.
        Total entries in the logical memory = $2^6 * 2^{10} = 2^{6+10} = 2^{16}$
        Number of bits = 16

    b. Physical address:
        Number of frames = 32 = 2^5
        Words (entries) in each page = 2^{10}
        Total entries = $2^5 * 2^{10} = 2^{5+10} = 2^{15}$
        Number of bits = 15

6. There are three reasons: First is that these mobile devices typically use flash memory with limited capacity and swapping is avoided because of this space constraint. Second, flash memory can support a limited number of write operations before it becomes less reliable. Lastly, there is typically poor throughput between main memory and flash memory.

7. Assuming a 1-KB page size, what are the page numbers and offset

    Quotient is the page number
    page number = 308 / 1024 = 0

    Remainder is the offset.
    308 mod 1024 = 308

8. The BTV operating system has a 21-bit virtual address, yet on certain embedded devices, it has only a 16-bit physical address. It also has a 2-KB page size.

    Logical address bits = 21
    Page size = 2KB = 2^{11}

   Number of Pages = $2^{21/}2^{11}$ = 2^{10} pages, each page number is in the page table. An entry for every page.

   a. Conventional, single-level page table will have 2^{10} = 1024 entries.
   b. 16-bit physical address can hold a maximum of holds 2^{16} addresses.
      2^{16} = 65536 B = 64 KB

9. Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.
    a. Logical address:
        Number of pages = 256 = 2^8
        Page size = 4*1024 = $2^2*2^{10} = 2^{12}$
        Total entries = $2^8 * 2^{12} = 2^{8+12} = 2^{20}$
        Number of bits = 20
    b. Physical address:
        Number of frames = 64 = 2^6
        Page size = 2^{12}
        Total entries = $2^6 * 2^{12} = 2^{18}$
        Number of bits = 18
10. An entry for each page. $2^{32} / 2^{12} = 2^{20}$
11. In certain situations, the page tables could become large enough that by paging the page tables, one could simplify the memory allocation problem.

1. Race conditions are possible in many computer systems. Consider a banking system with two methods: deposit(amount) and withdraw(amount). These two methods are passed the amount that is to be deposited or withdrawn from a bank account. Assume that a husband and wife share a bank account and that concurrently the husband calls the withdraw() method and the wife calls deposit(). Describe how a race condition is possible and what might be done to prevent the race condition from occurring.

   Answer:

   Assume the balance in the account is $250.00 and the husband calls withdraw($50) and the wife calls deposit($100). Obviously, the correct value should be $300.00. Since these two transactions will be serialized, the local value of balance for the husband becomes $200.00, but before he can commit the transaction, the deposit($100) operation takes place and updates the shared value of balance to $350.00. We then switch back to the husband and the value of the shared balance is set to $200.00 - obviously an incorrect value.

2. Race conditions are possible in many computer systems. Consider the producer–consumer problem, which is representative of operating systems, consisting of cooperating sequential processes, running asynchronously and sharing data using a bounded buffer. An integer variable counter, initialized to 0 is incremented every time we add a new item to the buffer and is decremented every time we remove one item from the buffer.

   The code for the producer process is as follows:

```
while (true) {
        /* produce an item in next_produced */
        while (counter == BUFFER_SIZE)  ;         /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
        counter++;
}
```

   The code for the consumer process is as follows:

```
while (true)
        { while (counter == 0) ;              /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        counter--;
        /* consume the item in next _consumed */
}
```

   Note that the statement "counter++" may be implemented in machine language (where register1 is one of the local CPU registers) as follows:

   $register_1$ = counter
   $register_1 = register_1 + 1$
   counter = $register_1$

Similarly, the statement “counter--” is implemented as follows:

> $register_2$ = counter
> $register_2 = register_2 - 1$
> counter = $register_2$

   a. Describe how a race condition is possible Page 259 in book.
   b. What data have a race condition? counter
   c. What might be done to prevent the race condition from occurring? Page 259 in book

3. Define:
   a. Critical-section problem Page 260.
   b. Race condition page 259
   c. Busy waiting Page 272
4. What are the 3 requirements that a solution to the critical-section problem must satisfy? Page 260
5. What is a semaphore? What is it used for? Page 272, 273
6. Consider two concurrently running processes: *P*1 with a statement *S*1 and *P*2 with a statement *S*2. Suppose we require that *S*2 be executed only after *S*1 has completed. How do you implement this scheme using a semaphore? [Hint: Refer section 6.6.1]

   We can implement this scheme readily by letting P1 and P2 share a common semaphore synch, initialized to 0.

   In process P1, we insert the statements
   ```
   S1;
   signal(synch);
   ```

   In process P2, we insert the statements
   ```
   wait(synch);
   S2;
   ```
   Because synch is initialized to 0, P2 will execute S2 only after P1 has invoked signal(synch), which is after statement S1 has been executed.
7. Write the code for the two Semaphore operations. Page 273 Why are they atomic? For uninterrupted updates of shared data
8. Assume that a system has multiple processing cores. For each of the following scenarios, describe which is a better locking mechanism—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. Spinlock
   - The lock is to be held for a long duration. Mutex lock
9. Explain how deadlock is possible with the dining-philosophers problem.

   Suppose that all five philosophers become hungry at the same time and each grabs the left chopstick. All the elements of chopstick will now be equal to 0. When each philosopher tries to grab the right chopstick, deadlock happens.
10. Consider a system consisting of two processes, *P*0 and *P*1, each accessing two semaphores, S and Q, set to the value 1:

```
     P0            P1

wait(S);       wait(Q);
wait(Q);       wait(S);
     .              .
     .              .
     .              .
signal(S);     signal(Q);
signal(Q);     signal(S);
```

Suppose that P0 executes wait(S) and then P1 executes wait(Q). When P0 executes wait(Q), it must wait until P1 executes signal(Q). Similarly, when P1 executes wait(S), it must wait until P0 executes signal(S). Is the system deadlocked? Explain why or why not.

Suppose that P_0 executes wait(S) and then P_1 executes wait(Q).
When P_0 executes wait(Q), it must wait until P_1 executes signal(Q).
Similarly, when P_1 executes wait(S), it must wait until P_0 executes signal(S).
Since these signal() operations cannot be executed, P_0 and P_1 are deadlocked.

11. Suppose that a process interchanges the order in which the wait() and signal() operations on the semaphore mutex are executed, resulting in the following execution:

```
signal(mutex);
...
critical section
...
wait(mutex);
```

Find the error generated by the use semaphores incorrectly.

In this situation, several processes may be executing in their critical section simultaneously, violating the mutual-exclusion requirement.

12. The structure of the Producer Process is given below. What are the purposes of the semaphores in the code?

```
do {
     . . .
   /* produce an item in next_produced */
     . . .
   wait(empty);
   wait(mutex);
     . . .
   /* add next_produced to the buffer */
     . . .
   signal(mutex);
   signal(full);
} while (true);
```

The mutex semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1.
The empty and full semaphores count the number of empty and full buffers.
The semaphore empty is initialized to the value n; the semaphore full is initialized to the value 0.