操作系统作业1

蔡佳伟 3220104519

知识点1：中断处理

Question 1:

In an interrupt-driven operating system, when an interrupt occurs, the operating system saves the CPU state using __ and the __. To determine the type of interrupt, the system can use either polling through a __ or a __ interrupt system with an interrupt table. The interrupt handling is then managed by specific code segments that decide the appropriate action for each type of interrupt.

Answer:

1. **registers**
2. **program counter**
3. **generic routine**
4. **vector**

Explanation:

- When an interrupt occurs, the operating system saves the CPU's state using the **registers** and the **program counter**. This allows the system to resume the interrupted process once the interrupt has been handled.
- The system then identifies the type of interrupt. This can be done by polling (actively checking) using a **generic routine** or by using a **vector** system, where each interrupt type has a specific entry in an interrupt table that points to the appropriate handler.
- Specific code segments determine the correct action for each type of interrupt after the interrupt is identified.

答案正确性和题目的难易程度：答案正确。操作系统通过存储寄存器和程序计数器来保存CPU的状态。确定发生了哪种类型的中断:通过泛型例程(generic routine)进行轮询(polling)或者矢量(vector)中断系统(中断表)。单独的代码段决定对每种类型的中断应该采取什么行动

题目难度中等。

Question 2:

Which of the following describes the **main purpose** of interrupt handling in an operating system?

A) To allow the operating system to perform background computations.
B) To ensure the CPU can focus on executing user-level processes without interruptions.
C) To immediately stop the CPU from executing the current process and transfer control to a special interrupt handler.
D) To prevent processes from being preempted by the kernel during execution.

Answer:

- **C) To immediately stop the CPU from executing the current process and transfer control to a special interrupt handler.**

Explanation:

- The primary purpose of **interrupt handling** is to allow the CPU to respond to external events (e.g., hardware signals) or internal events (e.g., software exceptions) by interrupting the current execution flow and transferring control to an **interrupt handler**. This helps the operating system manage time-sensitive tasks or system events efficiently.
- Option A is incorrect because interrupt handling doesn't focus on background computations. Option B and D are incorrect because interrupts allow the CPU to deal with higher-priority tasks, such as I/O, rather than letting it execute indefinitely.

答案正确性和题目的难易程度：答案正确，难度较低，中断比较明显是和stop相关，其他选项不太对。

Question 3:

Which of the following is true about the **interrupt vector** in interrupt handling?

A) It stores the addresses of all system processes.
B) It contains a list of available hardware devices.
C) It holds the addresses of interrupt service routines (ISRs) for different types of interrupts.
D) It is used to manage the CPU's memory allocation.

Answer:

- **C) It holds the addresses of interrupt service routines (ISRs) for different types of interrupts.**

Explanation:

- The **interrupt vector** is a table used by the operating system to store the addresses of **Interrupt Service Routines (ISRs)**. When an interrupt occurs, the CPU looks up the appropriate ISR in the interrupt vector to handle the interrupt.
- Option A is incorrect because it relates to process management, not interrupt handling. Option B is incorrect because the interrupt vector is not about hardware devices, but about handling interrupts. Option D is incorrect because the interrupt vector is unrelated to memory allocation.

答案正确性和题目的难易程度：答案正确，题目较容易，因为其他选项不对比较明显，而且有俗话三短一长选最长（。这个题目也是关于中断定义和具体操作的。

知识点2：系统调用

Question 1:

Which of the following statements are true about the relationship between System Calls and APIs in an operating system?

A) System Calls are lower-level operations that are directly invoked by the operating system.

B) APIs provide an interface for user programs to interact with the operating system, often without directly invoking system calls.

C) System Calls are typically invoked by user programs through an API, not directly.

D) APIs allow user programs to interact with the operating system at a low level, directly using hardware resources.

Answer:

- **A) True**
- **B) True**

- **C) True**
- **D) False**

Explanation:

- **A) True**: System Calls are indeed lower-level operations that provide essential services like file management, process control, etc., and they are invoked by the operating system.
- **B) True**: APIs provide a higher-level interface for user programs, abstracting the complexity of system calls. User programs typically interact with the system via APIs, not directly making system calls.
- **C) True**: System Calls are often accessed indirectly by user programs through APIs, which make it easier to perform operations without needing to call the system calls directly.
- **D) False**: APIs provide an abstracted interface for user programs to interact with the OS, but they do not directly allow access to low-level hardware resources. System Calls are the ones that interact with the hardware at a lower level.

答案正确性和题目的难易程度：答案正确。用户不会直接使用系统调用，而是通过API提供的接口访问。

题目难度较容易。

Question 2:

Which of the following is a **characteristic** of a **system call**?

A) System calls are executed directly by the user application without kernel involvement.
B) System calls provide an interface between user-level processes and the kernel.
C) System calls are typically used for computation-intensive tasks, such as mathematical operations.
D) System calls operate entirely in user space and do not require switching to kernel space.

Answer:

- **B) System calls provide an interface between user-level processes and the kernel.**

Explanation:

- **System calls** act as a bridge between user applications and the operating system kernel, allowing user processes to request services (like file operations, memory management, process control, etc.) from the kernel.
- Options A, C, and D are incorrect because system calls involve kernel involvement, especially for operations that require access to system resources, and they involve a switch to kernel space.

答案正确性和题目的难易程度：答案正确，题目较容易，正确选项很明显，只要了解系统调用就能答对。

Question 3:

When should a process typically use a **system call**?

A) When a process wants to perform arithmetic operations.
B) When a process wants to interact with hardware, such as reading from a disk.
C) When a process needs to manage its own memory allocation.
D) When a process is in the **running** state and needs to continue executing.

Answer:

- **B) When a process wants to interact with hardware, such as reading from a disk.**

Explanation:

- **System calls** are used when a process needs to interact with system resources such as hardware (disk, network devices), manage memory, or perform I/O operations. Accessing hardware requires kernel-level services, and system calls provide that mechanism.
- Arithmetic operations (A) and memory management (C) can often be done in user space without a system call, and option D is incorrect because system calls are used for specific requests, not for general process execution.

答案正确性和题目的难易程度：答案正确。系统调用硬件资源可以被使用。难度中等。

知识点3：进程的状态转换图

Question 1:

Which of the following state transitions are valid in a process state diagram?

A) A process can transition from **Ready** to **New**.

B) A process can transition from **Running** to **Waiting**.

C) A process can transition from **Waiting** to **Ready**.

D) A process can transition from **Terminate** to **Ready**.

Answer:

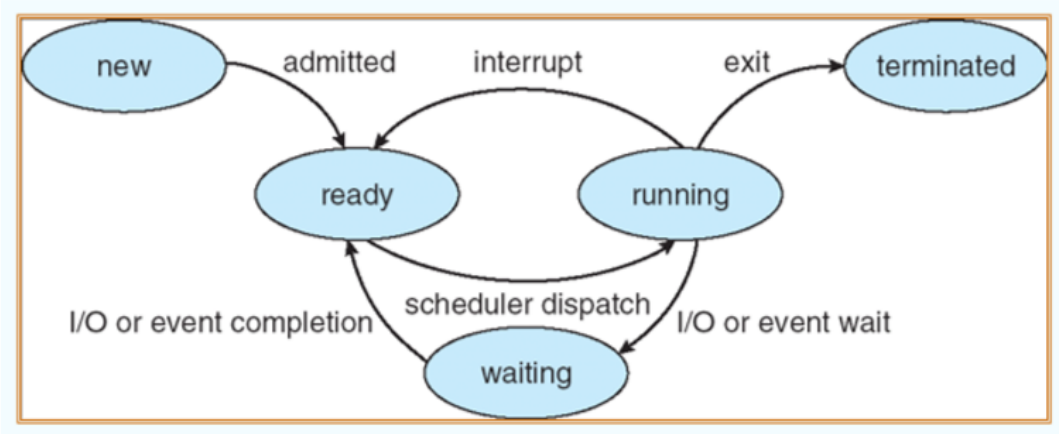
- **B) True**
- **C) True**

Explanation:

- **A) False**: A process cannot transition from **Ready** to **New**. The **New** state refers to the process being created, whereas **Ready** is the state when a process is ready to execute, so a transition directly from **Ready** to **New** does not make sense.
- **B) True**: A process can transition from **Running** to **Waiting**. This happens when a process is running and needs to wait for some event or resource, such as input/output completion, causing it to move to the **Waiting** state.
- **C) True**: A process can transition from **Waiting** to **Ready**. After the event or resource the process was waiting for becomes available, the process can move back to the **Ready** state to wait for CPU time to run.
- **D) False**: A process cannot transition from **Terminate** to **Ready**. Once a process is **Terminated**, it has finished execution and is removed from the system, so it cannot go back to the **Ready** state.

答案正确性和题目的难易程度：答案正确，原因如图

状态转换图：



题目较容易

Question 2:

Which of the following events would cause a process to transition from the **running** state to the **waiting** state in a typical process lifecycle?

A) The process completes its execution.
B) The process requests an I/O operation.
C) The process is preempted by the scheduler.
D) The process terminates and releases all resources.

Answer:

- **B) The process requests an I/O operation.**

Explanation:

- When a process requests an **I/O operation**, it is unable to continue executing until the operation completes, so it transitions from the **running** state to the **waiting** state (also known as the **blocked** state). The process will stay in the waiting state until the I/O is finished.
- A process transitions to the **ready** state if it is preempted (option C) or to the **terminated** state if it finishes its execution (option A or D).

答案正确性和题目的难易程度：答案正确，难度中等，因为很容易难以区分waiting态和ready态，我自己差一点就认为答案是错误的了。

Question 3:

Under which condition will a process move from the **ready** state to the **running** state?

A) The process has completed its execution.
B) The process has requested a resource and is waiting for it.
C) The scheduler selects the process for execution.
D) The process is preempted by the operating system.

Answer:

- **C) The scheduler selects the process for execution.**

Explanation:

- A process moves from the **ready** state to the **running** state when the **scheduler** selects it to execute. This happens when the process is ready to execute and the CPU is assigned to it.

- The other options describe scenarios where a process might transition to a different state, such as completing execution (A), requesting resources (B), or being preempted (D).

答案正确性和题目的难易程度：答案正确，题目难度较低，因为跑到running态比较明显。

知识点4：用户级线程和内核级线程

Question 1:

Which of the following statements are true about User-Level Threads and Kernel-Level Threads?

A) User-Level Threads are managed by the operating system kernel.

B) Kernel-Level Threads can utilize multiple CPUs effectively.

C) User-Level Threads are unaware of the operating system kernel and are managed entirely in user space.

D) Kernel-Level Threads cannot be scheduled when one thread in the process is blocked.

Answer:

- **B) True**
- **C) True**

Explanation:

- **A) False**: User-Level Threads are managed entirely by user-level libraries or the application itself. The operating system kernel is unaware of the existence of these threads, as the management occurs in user space.
- **B) True**: Kernel-Level Threads are managed by the operating system kernel, which allows them to take advantage of multiple CPUs. The kernel can schedule these threads on different CPUs, thus improving the system's overall efficiency.
- **C) True**: User-Level Threads are managed entirely by user-space programs or libraries. The operating system kernel is unaware of their existence, and all the thread management, including scheduling and synchronization, happens in user space.
- **D) False**: Kernel-Level Threads can still be scheduled even if one thread in the process is blocked. The kernel can continue running other threads in the same process or other processes, even if one thread is waiting for an event or resource.

答案正确性和题目的难易程度：答案正确，题目较容易，解答的原因已经给出。

Question 2:

Which of the following is a characteristic of **user-level threads**?

A) User-level threads are managed directly by the operating system kernel.
B) User-level threads can be scheduled and managed without kernel intervention.
C) User-level threads cannot be blocked by system calls.
D) User-level threads require a separate thread for each user process.

Answer:

- **B) User-level threads can be scheduled and managed without kernel intervention.**

Explanation:

- **User-level threads** are managed entirely in user space, without the kernel's involvement. This allows for efficient context switching between threads, but the kernel is unaware of the

individual threads. The kernel sees only the process, not the threads within it.
- In contrast, **kernel-level threads** are managed by the kernel and can be preempted or blocked by system calls.

答案正确性和题目的难易程度：答案正确，题目难度中等，主要需要区分出用户级线程和内核级线程的特点。

Question 3:

Which of the following is an advantage of **kernel-level threads** over **user-level threads**?

A) Kernel-level threads can be more efficiently scheduled as the kernel has direct control.
B) User-level threads can perform better in multi-core systems than kernel-level threads.
C) Kernel-level threads cannot be preempted by the operating system.
D) User-level threads can easily take advantage of multiple processors without any overhead.

Answer:

- **A) Kernel-level threads can be more efficiently scheduled as the kernel has direct control.**

Explanation:

- **Kernel-level threads** are managed by the operating system, which allows the kernel to perform efficient scheduling and preemption. This means the kernel can make decisions about how to distribute threads across processors or handle blocking operations.
- **User-level threads**, while fast for context switching, do not benefit from kernel management, which can limit their ability to fully utilize multi-core processors or handle blocking system calls efficiently.

答案正确性和题目的难易程度：答案正确，题目难度中等，内核级线程可以采用多线程技术，也可以发挥多CPU优势。

知识点5：CPU调度

Question 1:

Which of the following CPU scheduling algorithms is **non-preemptive**?

A) **First-Come, First-Served (FCFS)**
B) **Round Robin (RR)**
C) **Shortest Job Next (SJN)**
D) **Priority Scheduling (Preemptive)**

Answer:

- **A) First-Come, First-Served (FCFS)**

Explanation:

- **First-Come, First-Served (FCFS)** is a **non-preemptive** scheduling algorithm, meaning that once a process starts executing, it runs to completion without being interrupted.
- In contrast, **Round Robin (RR)**, **Shortest Job Next (SJN)**, and **Priority Scheduling (Preemptive)** can all involve preemption, where a process can be interrupted before it finishes.

答案正确性和题目的难易程度：答案正确，题目较容易，先来先到永远都是非抢占的方法。

Question 2:

Which CPU scheduling algorithm is best suited for minimizing the average waiting time of processes?

A) **First-Come, First-Served (FCFS)**
B) **Round Robin (RR)**
C) **Shortest Job First (SJF)**
D) **Priority Scheduling**

Answer:

- **C) Shortest Job First (SJF)**

Explanation:

- **Shortest Job First (SJF)** is optimal for minimizing the average waiting time, as it prioritizes processes with the shortest burst time. By executing the shortest jobs first, it minimizes the waiting time for other processes in the queue.
- **FCFS** and **RR** may lead to higher waiting times, especially for longer processes, while **Priority Scheduling** depends on the assigned priorities, which may not always minimize waiting time effectively.

答案正确性和题目的难易程度：答案正确，由于贪心算法，SJF的等待时间永远是最短的，因为让最短时间的优先去做（ads课上学过），题目难度中等，因为不理解的话就很难选择。

Question3:

Consider the following set of processes with their arrival times and burst times. Calculate the average waiting time and turnaround time for both **Preemptive** and **Non-Preemptive** scheduling algorithms: **Shortest Job First (SJF)**.

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	2
P4	3	1

For both scheduling methods, calculate the average **Waiting Time (WT)** and **Turnaround Time (TAT)**.

Definitions:

- **Waiting Time (WT)** is the total time a process spends in the ready queue.
- **Turnaround Time (TAT)** is the total time from the arrival of the process to its completion, i.e., **TAT = Completion Time - Arrival Time**.

Answer:

Non-Preemptive Shortest Job First (SJF):

- **Order of execution: P1, P2, P3, P4** (SJF executes the shortest job first, and once a process starts, it runs to completion).

1. **Process P1** arrives at 0 and runs for 8 units of time (completes at time 8).

- Waiting time for P1: **0** (P1 starts execution immediately).
- Turnaround time for P1: **8 - 0 = 8**.

2. **Process P2** arrives at time 1 but must wait for P1 to finish, so it starts at time 8 and runs for 4 units of time (completes at time 12).
   - Waiting time for P2: **8 - 1 = 7** (P2 waited for P1 to finish).
   - Turnaround time for P2: **12 - 1 = 11**.
3. **Process P3** arrives at time 2, but after P2 starts, it runs next because it has the shortest burst time. P3 runs from time 12 to time 14 (2 units).
   - Waiting time for P3: **12 - 2 = 10**.
   - Turnaround time for P3: **14 - 2 = 12**.
4. **Process P4** arrives at time 3, and it runs next after P3, taking 1 unit of time. P4 runs from time 14 to time 15.
   - Waiting time for P4: **14 - 3 = 11**.
   - Turnaround time for P4: **15 - 3 = 12**.

Average Waiting Time for Non-Preemptive SJF:

$$\text{Average WT} = \frac{0 + 7 + 10 + 11}{4} = \frac{28}{4} = 7$$

Average Turnaround Time for Non-Preemptive SJF:

$$\text{Average TAT} = \frac{8 + 11 + 12 + 12}{4} = \frac{43}{4} = 10.75$$

Preemptive Shortest Job First (SJF):

- **Order of execution:** The process with the shortest burst time runs first, and it can preempt any currently running process if another process arrives with a shorter burst time.

1. **Process P1** starts first, but it is preempted by P2 at time 1. It runs for 1 unit (until time 1).
   - P1's remaining burst time: 7 units.
2. **Process P2** starts at time 1 and runs for 4 units (from time 1 to time 5).
   - P2 completes at time 5.
   - Waiting time for P2: **1 - 1 = 0**.
   - Turnaround time for P2: **5 - 1 = 4**.
3. **Process P3** arrives at time 2 and is preempted by P2. P3 runs for 2 units from time 5 to time 7 (no preemption happens after this).
   - P3 completes at time 7.
   - Waiting time for P3: **5 - 2 = 3**.
   - Turnaround time for P3: **7 - 2 = 5**.
4. **Process P1** resumes after P2 and P3, running from time 7 to time 15.
   - Waiting time for P1: **15 - 8 = 7**.
   - Turnaround time for P1: **15 - 0 = 15**.
5. **Process P4** runs from time 7 to time 8 (1 unit), completing at time 8.
   - Waiting time for P4: **7 - 3 = 4**.
   - Turnaround time for P4: **8 - 3 = 5**.

Average Waiting Time for Preemptive SJF:

$$\text{Average WT} = \frac{7 + 0 + 3 + 4}{4} = \frac{14}{4} = 3.5$$

Average Turnaround Time for Preemptive SJF:

$$\text{Average TAT} = \frac{15 + 4 + 5 + 5}{4} = \frac{29}{4} = 7.25$$

Summary of Results:

- **Non-Preemptive SJF:**
  - Average Waiting Time: 7
  - Average Turnaround Time: 10.75
- **Preemptive SJF:**
  - Average Waiting Time: 3.5
  - Average Turnaround Time: 7.25

Conclusion:
Preemptive SJF results in lower average waiting time and turnaround time compared to non-preemptive SJF, as it allows shorter jobs to preempt running processes, minimizing delays.

答案正确性和题目的难易程度：答案错误，

非抢占式：p1执行完0-8以后，p4由于时间短，应该先执行，顺序应该是p1-p4-p3-p2，p1等待时间是0，周转时间是8，p2等待时间是8+2+1-1=10，周转时间是14，p3等待时间是8+1-2=7，周转时间是9，p4等待时间是8-3=5，周转时间是6。平均下来等待时间是5.5，周转时间是9.25.

抢占式：p1执行0-1，然后被p2抢占执行1-2，然后被p3抢占执行2-3，然后p3继续执行3-4，p4在p3执行完后执行4-5，然后p2执行5-8结束，p1执行8-15结束。p1的等待时间是7，周转时间是15，p2等待时间是3，周转时间是7，p3等待时间是0，周转时间是4，p4等待时间是1，周转时间是2。平均等待时间是2.75，周转时间是7。

题目难度较高。

知识点6：临界区互斥遵循的准则

Question 1:

The solution to the **Critical Section Problem** must satisfy the following three conditions:

1. __: If a process Pi is executing in its critical section, no other process can execute in its critical section.
2. __: If no process is executing in its critical section and some processes wish to enter their critical sections, the selection of the next process to enter the critical section cannot be delayed indefinitely.
3. __: After a process requests to enter its critical section, the number of times other processes can enter their critical sections before it is granted permission must be bounded.

Additionally, processes should follow the principle of __, where they should release the processor immediately if they cannot enter the critical section to avoid busy-waiting.

Answer:

1. **Mutual Exclusion**
2. **Progress**
3. **Bounded Waiting**

4. **Preemption**

Explanation:

- **Mutual Exclusion** ensures that no two processes can execute in their critical sections at the same time.
- **Progress** ensures that if no process is in the critical section, and some processes want to enter, the selection of the next process to enter cannot be delayed indefinitely.
- **Bounded Waiting** ensures that there is a limit to how many times other processes can enter their critical sections before a process waiting for entry is granted permission.
- **Preemption** is a principle that suggests processes should release the processor if they cannot enter the critical section, preventing busy-waiting (i.e., waiting for a long time while still holding the processor).

答案正确性和题目的难易程度：答案正确，解释也比较直白，题目难度较高（如果是选择题，那将是较简单，但名词很难填出来）

Question 2:

Which of the following conditions from the **Critical Section Problem** does the statement below represent?

"If a process Pi is executing in its critical section, no other process can execute in its critical section."

A) **Mutual Exclusion**
B) **Progress**
C) **Bounded Waiting**
D) **Preemption**

Answer:

- **A) Mutual Exclusion**

Explanation:
The statement describes the **Mutual Exclusion** condition, which ensures that if one process is executing in its critical section, no other process can enter or execute in its critical section at the same time.

答案正确性和题目的难易程度：答案正确，这个描述是说满足互斥原则。难度简单，因为没有什么干扰项同时比较直白。

Question 3:

Consider the following example:

- Process P1 holds Resource R1 and is requesting Resource R2.
- Process P2 holds Resource R2 and is requesting Resource R1.
- There are no other processes or resources involved.

Which condition from the **Critical Section Problem** is **not satisfied** in this scenario?

A) **Mutual Exclusion**
B) **Progress**
C) **Bounded Waiting**
D) **Preemption**

Answer:

- **B) Progress**

Explanation:
In this scenario, **Progress** is not satisfied because both P1 and P2 are deadlocked, waiting for each other to release resources. There is no process that can proceed, and thus the selection of the next process to enter the critical section is indefinitely delayed.

答案正确性和题目的难易程度：答案正确，利用死锁的例子演示出不满足空闲让进。难度中等，因为不太好区分空闲让进和有限等待。

知识点7：死锁产生

Question1:

Which of the following are the necessary conditions for a **deadlock** to occur?

A) **Mutual Exclusion**: Resources are shared by multiple processes, and no process can execute without holding at least one resource.

B) **Hold and Wait**: A process holding one resource is waiting to acquire additional resources that are being held by other processes.

C) **No Preemption**: Resources can only be released by the process holding them voluntarily after completing its task.

D) **Circular Wait**: A set of processes {P0, P1, ..., Pn} are waiting for resources in such a way that each process in the set is waiting for a resource held by the next process in the set, with the last process waiting for a resource held by the first process.

Answer:

- **B) True**
- **C) True**
- **D) True**
- **A) False**

Explanation:

- **A) False**: The **Mutual Exclusion** condition requires that a resource is held by only one process at a time, but it does not mean that resources are shared by multiple processes. The incorrect statement in this option is the claim that resources are shared by multiple processes, which contradicts the **mutual exclusion** condition.
- **B) True**: **Hold and Wait** means that a process is holding at least one resource and is waiting for additional resources held by other processes, which is one of the key conditions for a deadlock to occur.
- **C) True**: **No Preemption** implies that once a process holds a resource, it cannot be forcibly taken away until the process voluntarily releases it. This condition is essential for deadlock as it ensures that processes cannot be interrupted to resolve resource contention.
- **D) True**: **Circular Wait** occurs when there is a circular chain of processes, where each process in the chain is waiting for a resource held by the next process in the chain, ultimately leading back to the first process. This cyclic waiting is a necessary condition for deadlock.

答案正确性和题目的难易程度：答案正确，题目难度中等，因为互斥条件中的小错误（资源由多个进程占用），不容易被发现。知识点来源：产生死锁的四个必要条件:

1. Mutual exclusion（互斥条件）：进程要求对所分配的资源进行排他性使用，即在一段时间内某资源仅为一个进程所占有。此时若有其他进程请求该资源，则只能等待。

2. Hold and wait（请求并保持条件）：持有至少一个资源的进程正在等待获取其他进程持有的额外资源。
3. No preemption（不剥夺条件）：资源只能由持有该资源的进程在完成其任务后自愿释放。
4. Circular wait(循环等待/环路等待条件)：存在一组{P0, P1,…， Pn}的等待进程，使得P0正在等待由P1持有的资源，P1正在等待由P2持有的资源，…， Pn - 1在等待一个由Pn持有的资源，Pn在等待一个由P0持有的资源。

Question 2:

Consider the following resource allocation graph:

- Processes: P1, P2, P3
- Resources: R1, R2

The edges in the graph represent the following:

- A directed edge from a process to a resource indicates that the process is requesting that resource.
- A directed edge from a resource to a process indicates that the process is holding that resource.

The graph shows the following relationships:

- P1 is holding R1.
- P2 is holding R2.
- P1 is requesting R2.
- P2 is requesting R1.

Is there a **deadlock** in this system?

Answer:

Yes, **there is a deadlock** in this system.

Explanation:

- P1 is holding R1 and requesting R2.
- P2 is holding R2 and requesting R1.
- This forms a cycle of waiting, where P1 is waiting for P2 to release R2 and P2 is waiting for P1 to release R1, resulting in a deadlock.

答案正确性和题目的难易程度：答案正确，题目难度简单，死锁很明显。

Question 3:

Consider the following resource allocation graph:

- Processes: P1, P2, P3
- Resources: R1, R2, R3

The edges in the graph represent the following:

- A directed edge from a process to a resource indicates that the process is requesting that resource.
- A directed edge from a resource to a process indicates that the process is holding that resource.

The graph shows the following relationships:

- P1 is holding R1 and requesting R2.

- P2 is holding R2 and requesting R3.
- P3 is holding R3 and requesting R1.

Is there a **deadlock** in this system?

Answer:

Yes, **there is a deadlock** in this system.

Explanation:

- P1 is holding R1 and requesting R2.
- P2 is holding R2 and requesting R3.
- P3 is holding R3 and requesting R1.
- This forms a cycle where each process is waiting for a resource held by another process (P1 → P2 → P3 → P1), resulting in a deadlock.

答案正确性和题目的难易程度：答案正确，题目难度简单，死锁也很明显，三个成环了。