2.1 What is the purpose of system calls?

Answer: The purpose of system calls is to provide a controlled interface for user programs to request services from the operating system, such as file management, process control, and communication, allowing applications to interact safely with the hardware.

2.2 What is the purpose of the command interpreter, and why is it separate from the kernel?

Answer: The command interpreter, or shell, allows users to interact with the operating system by executing commands. It is separate from the kernel to provide a user-friendly interface while keeping the kernel focused on core system functions, enhancing security and stability.

2.3 What system calls must be executed by a command interpreter to start a new process on a UNIX system?

Answer: To start a new process on a UNIX system, the command interpreter typically uses the following system calls: `fork()` to create a new process, `exec()` to load the program into the new process's memory, and `wait()` to manage the process's termination and retrieve its exit status.

2.4 What is the purpose of system programs?

Answer: System programs provide utilities and tools that perform specific tasks for users and applications, such as file manipulation, system monitoring, and process management, making it easier to use and interact with the operating system.

2.5 What are the main advantages and disadvantages of the layered approach to system design?

Answer: The main advantage of the layered approach is improved modularity, making it easier to manage, develop, and debug the system. Disadvantages include potential performance overhead due to multiple layers of abstraction and complexity in communication between layers.

2.6 List five services provided by an operating system and their user convenience.

1. **File Management**: Simplifies file creation, deletion, and manipulation, enabling easy access to data.
2. **Process Management**: Manages process scheduling and execution, ensuring efficient CPU utilization.
3. **Memory Management**: Allocates and deallocates memory efficiently, allowing applications to run smoothly.
4. **Device Management**: Provides an interface for interacting with hardware devices, simplifying I/O operations.
5. **Networking Services**: Enables communication between devices over a network, facilitating data exchange.

User-level programs cannot provide these services effectively because they require direct access to hardware and system resources, which can only be safely managed by the operating system for security and stability.

2.7 Why do some systems store the operating system in firmware while others store it on disk?

Answer: Some systems store the operating system in firmware for faster boot times and increased reliability, as it is less susceptible to corruption. Others store it on disk to allow for easier updates and modifications, providing flexibility in system management.

2.8 How could a system be designed to allow a choice of operating systems from which to boot?

Answer: A system could implement a boot manager that presents a menu of installed operating systems at startup. The bootstrap program would need to detect available OS installations, load the selected OS into memory, and transfer control to it, ensuring proper initialization and resource allocation for the chosen system.

2.9 What are the two categories of services provided by an operating system?

Answer: The two main categories of services are **user services** and **system services**. User services facilitate interaction between the user and the system, providing user interfaces, command interpretation, and file management. System services manage hardware resources and provide functionalities like process scheduling, memory management, and device control, ensuring system stability and security.

2.10 What are three general methods for passing parameters to the operating system?

Answer: Three methods for passing parameters to the operating system are:

1. **Registers**: Parameters are placed in specific CPU registers before a system call is made.
2. **Stack**: Parameters are pushed onto the stack, and the operating system retrieves them from there.
3. **Memory**: A block of memory is allocated for parameters, and the address of that memory is passed to the operating system.

2.11 How can you obtain a statistical profile of a program's execution time?

Answer: You can use profiling tools that monitor the execution of a program, tracking how much time is spent in different code sections. This is often done by instrumenting the code or using performance counters. Obtaining such a profile is important for identifying bottlenecks, optimizing performance, and guiding improvements in code efficiency.

2.12 What are the advantages and disadvantages of a unified system-call interface for files and devices?

Answer: Advantages include simplified programming and consistency in accessing resources, as the same system calls can be used for both files and devices. Disadvantages may include performance overhead, as device operations may require different handling compared to file operations, potentially leading to inefficiencies.

2.13 Can a user develop a new command interpreter using the system-call interface?

Answer: Yes, a user can develop a new command interpreter using the system-call interface provided by the operating system. The system calls allow the interpreter to perform tasks such as executing commands, managing processes, and handling input/output.

2.14 Why does Android use ahead-of-time (AOT) compilation?

Answer: Android uses AOT compilation to improve application startup times and performance by compiling applications into native code before they are run. This approach reduces the overhead associated with Just-In-Time (JIT) compilation, making apps run more efficiently on mobile devices with limited resources.

2.15 What are the two models of interprocess communication, and what are their strengths and weaknesses?

Answer: The two models are **message passing** and **shared memory**. Message passing allows processes to communicate by sending messages, which provides strong synchronization but may introduce overhead. Shared memory allows multiple processes to access the same memory space, leading to faster communication but requiring careful synchronization to avoid data inconsistencies.

2.16 Compare an application programming interface (API) and an application binary interface (ABI).

Answer: An **API** defines a set of functions and protocols for building software applications, specifying how software components should interact. An **ABI**, on the other hand, defines the low-level interface between applications and the operating system, including binary formats and calling conventions, ensuring compatibility at the machine level.

2.17 Why is the separation of mechanism and policy desirable?

Answer: Separating mechanism from policy allows for flexibility and adaptability in system design. Mechanisms provide the underlying capabilities, while policies determine how those capabilities are utilized. This separation enables easier updates and modifications to policies without altering the underlying mechanisms, enhancing system maintainability.

2.18 Provide a scenario where tight coupling makes layering difficult.

Answer: A scenario where layering is difficult might involve a memory management component that directly interacts with a file system for caching. Since the file system may rely on memory management decisions to optimize data access, tightly coupling their functionalities can blur the boundaries between layers, making it challenging to implement a clean layered architecture.

2.19 What is the main advantage of the microkernel approach to system design?

Answer: The main advantage of the microkernel approach is improved modularity and fault tolerance, as minimal functionality is implemented in the kernel, reducing complexity. User programs and system services interact through well-defined interfaces, allowing services to run in user space. Disadvantages include potential performance overhead due to increased context switching and communication between user space and kernel space.

2.20 What are the advantages of using loadable kernel modules?

Answer: Loadable kernel modules allow for dynamic loading and unloading of functionality, enabling easier updates, reduced memory usage, and customization of the kernel without requiring a full system reboot. This flexibility simplifies system management and enhances performance by loading only necessary components.

2.21 How are iOS and Android similar, and how are they different?

Answer: Both iOS and Android are mobile operating systems that provide app ecosystems, touch interfaces, and similar core functionalities. However, they differ in their underlying architectures, with iOS being more closed and tightly controlled by Apple, while Android is open-source and allows more customization by manufacturers and users.

2.22 Why don’t Java programs on Android use the standard Java API and virtual machine?

Answer: Android uses a modified version of Java that does not implement the standard Java API or virtual machine due to performance optimization and platform-specific requirements. Instead, it utilizes the Android Runtime (ART) and its own set of libraries designed for mobile environments.

2.23 What are the pros and cons of the Synthesis approach to kernel design?

Answer: Pros of the Synthesis approach include optimized system-call performance and reduced overhead since routines are assembled within kernel space. Cons include reduced modularity and increased complexity, as tightly coupled functionalities may lead to difficulties in maintaining and extending the operating system.

2.24 Write a program to copy the contents of one file to another using the POSIX API.

Answer: Here is a simple C program using the POSIX API to copy a file:

```
c
Copy code
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

void copy_file(const char *source, const char *destination) {
    int src_fd = open(source, O_RDONLY);
    if (src_fd == -1) {
        perror("Error opening source file");
        exit(EXIT_FAILURE);
    }

    int dest_fd = open(destination, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR |
S_IWUSR);
    if (dest_fd == -1) {
        perror("Error opening destination file");
        close(src_fd);
        exit(EXIT_FAILURE);
    }

    char buffer[4096];
    ssize_t bytes_read;
    while ((bytes_read = read(src_fd, buffer, sizeof(buffer))) > 0) {
        write(dest_fd, buffer, bytes_read);
    }

    if (bytes_read == -1) {
        perror("Error reading from source file");
    }

    close(src_fd);
    close(dest_fd);
}

int main() {
    char source[256], destination[256];
    printf("Enter source file name: ");
    scanf("%s", source);
    printf("Enter destination file name: ");
    scanf("%s", destination);
    copy_file(source, destination);
    printf("File copied successfully.\n");
    return 0;
}
```

This program prompts the user for the source and destination file names, checks for errors, and copies the file contents while ensuring that the source file exists.