CS604- OPERATING SYSTEM
(SOLVED Current Subjective)
FROM MIDTERM PAPERS
LECTURE (1-22)

BC190202640@vu.edu.pk

Junaidfazal08@gmail.com

For More Visit: vulmshelp.com

JUNAID MALIK
(0304-1659294)

AL-JUNAID TECH INSTITUTE

1. **Exit system call Wait system call**

ANSWER:

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction. Child process may terminate due to any of these: It calls exit();

2. **why mknod() call fail give two reason.**

ANSWER:

There are several reasons why a call to the mknod() function could fail:

The process does not have sufficient permissions to create the file. In order to create a file using mknod(), the process must have the appropriate permissions for the directory in which the file is being created.The file name is invalid. There are certain characters and conventions that are not allowed in file names, and if the name specified in the mknod() call contains such characters or does not follow the correct conventions, the call will fail.The file name is too long. There is a maximum length for file names on most systems, and if the name specified in the mknod() call exceeds this maximum length, the call will fail.The file system is full. If the file system does not have sufficient free space to create the file, the mknod() call will fail.The file already exists. If a file with the same name already exists in the specified directory, the mknod() call will fail unless the O_EXCL flag is set, in which case the call will fail with the EEXIST error.

3. **CPU schedulers**
   - **Long term**
   - **Short term**
   - **Middle term**

ANSWER:

In a computer operating system, the CPU scheduler is a component that manages the execution of processes on the CPU. There are three types of CPU schedulers:

Long-term scheduler (also called the job scheduler): This scheduler determines which processes should be admitted to the ready queue for execution. It is

responsible for balancing the trade-off between CPU utilization and response time. The long-term scheduler is responsible for deciding which programs will be executed in the future and for how long. It is a slower-acting scheduler and is usually not invoked very often.

Short-term scheduler (also called the CPU scheduler or dispatcher): This scheduler selects which process should be executed next and allocates the CPU to it. It is a faster-acting scheduler that is invoked very frequently (e.g., every few milliseconds).

Middle-term scheduler: This is a hybrid of the long-term and short-term schedulers. It is invoked less frequently than the short-term scheduler, but more frequently than the long-term scheduler. The middle-term scheduler is responsible for adjusting the priority of processes that have been in the system for a while and have not yet completed. This can help prevent processes that have been in the system for a long time (e.g., "starving" processes) from being starved of CPU resources.

4. ps command And working of Ps-e Ps-u Ps-l

ANSWER:

The ps command is a command-line utility that displays information about the processes running on a system. It can be used to display the processes of a specific user, the processes with a specific terminal, or all processes on the system.

The ps command has many options that can be used to modify its behavior. Some common options include:

-e: This option tells ps to display information about all processes on the system, including processes that have been terminated but have not yet been cleaned up by the system.

-u: This option tells ps to display information about processes owned by a specific user. The user's name or user ID can be specified as an argument to the -u option.

-l: This option tells ps to display information in a "long" format, which includes additional details such as the process's priority, nice value, and process ID.

For example, to display information about all processes on the system in a long format, you could use the command ps -el. To display information about processes owned by the user john in a long format, you could use the command ps -ul john.

5. Similarities between threads and process

ANSWER:

There are several similarities between threads and processes:

- Both threads and processes are independent sequences of execution that can run concurrently on a computer.
- Both threads and processes can have their own set of resources, such as memory, file handles, and network sockets.
- Both threads and processes can be scheduled by the operating system to run on a CPU.
- Both threads and processes can be created and terminated by the operating system or by other threads or processes.
- Both threads and processes can communicate with each other through various means, such as shared memory, message passing, or signaling.
- Both threads and processes can be used to divide a larger task into smaller units of work that can be executed concurrently.
- Both threads and processes can be used to improve the performance and scalability of an application by allowing it to make better use of multiple CPU cores.

6. Differentiate between absolute and relative path

ANSWER:

An absolute path is the full, exact path to a file or directory, starting from the root directory of the file system. It includes all directories and subdirectories that must be traversed to reach the file or directory. An absolute path is not affected by the current working directory of the process or the location of the file that is being executed.

A relative path, on the other hand, is a path to a file or directory relative to the current working directory of the process. It does not include the root directory of the file system, and it specifies only the directories and subdirectories that must be traversed from the current working directory to reach the file or directory.

For example, consider the following file system layout:

```
/
|-home
  |-user1
    |-file1.txt
  |-user2
    |-file2.txt
```

If the current working directory is /home/user1, then the absolute path to file1.txt is /home/user1/file1.txt, and the relative path to file1.txt is simply file1.txt. The absolute path to file2.txt is /home/user2/file2.txt, and the relative path to file2.txt from the current working directory is ../user2/file2.txt.

7. Can user process use the kernal code,justify.

ANSWER:

No, user processes cannot use kernel code directly. The kernel is the core of the operating system and provides services to user processes through system calls and other interfaces. It is protected from direct access by user processes, and user processes are not allowed to execute kernel code or access kernel data structures directly.

This separation between the kernel and user processes is important for several reasons:

- It helps to ensure the stability and security of the system. If user processes were allowed to execute kernel code or access kernel data structures directly, they could potentially crash the system or gain unauthorized access to sensitive information.
- It allows the kernel to be more efficient. By isolating the kernel from user processes, the kernel can be optimized for its specific tasks and does not need to worry about the unpredictable behavior of user processes.
- It allows for the development of multiple user-level libraries and programming languages. Because user processes cannot execute kernel code directly, they must rely on user-level libraries and programming

languages to perform certain tasks. This allows for greater flexibility and innovation in the development of user-level software.

8. Purpose of following commands

- **Fg**
- **Bg**
- **Ps**
- **Kill**
- **Jobs**

ANSWER:

fg: The fg command stands for "foreground." It is used to bring a background or suspended process to the foreground, allowing it to receive input from and display output to the terminal.

bg: The bg command stands for "background." It is used to resume a suspended process in the background, allowing it to continue running without taking up the terminal.

ps: The ps command stands for "process status." It is used to display information about the processes running on a system, including the process ID, command name, and status.

kill: The kill command is used to send a signal to a process, causing it to terminate. It can be used to terminate processes that are stuck or behaving unexpectedly.

jobs: The jobs command is used to display a list of jobs that are running or stopped in the background or foreground of the current shell. It can be used to monitor and manage background processes.

9. Why we use execple() system call after fork() system call

ANSWER:

The exec family of functions (e.g., execve(), execvp(), execl(), etc.) is used to replace the current process image with a new process image. This can be used to execute a different program, or to execute the same program with different arguments or environment variables.

The fork() system call is used to create a new process, which is a copy of the calling process. It is often used in combination with one of the exec functions to create a new process that runs a different program.

After a call to fork(), the new process (child process) and the original process (parent process) both continue to execute from the point at which the fork() call was made. However, the child process typically calls an exec function immediately after the fork() call to replace its process image with a new program. The parent process may continue to execute other code, or it may wait for the child process to complete using the wait() system call.

The combination of fork() and exec() is often used in Unix-like operating systems to create new processes that run different programs. It allows a process to create a child process that runs a specific program, while the parent process can continue to perform other tasks or monitor the child process.

10. Kernel Bottleneck

ANSWEWR:

A kernel bottleneck is a situation where the kernel (the core of the operating system) becomes a limiting factor in the performance of the system. This can occur when the kernel is unable to keep up with the demand for its services, or when it consumes too many resources and leaves insufficient resources for user processes.

There are several potential causes of kernel bottlenecks:

- High system load: If the system is under heavy load, with many processes competing for the same resources, the kernel may become a bottleneck as it tries to manage the resource contention.
- Inefficient kernel design: If the kernel is not well-optimized for the hardware and workload of the system, it may become a bottleneck as it performs poorly compared to other components of the system.
- Resource contention: If the kernel and user processes are competing for the same resources (e.g., CPU time, memory, I/O bandwidth), the kernel may become a bottleneck as it tries to allocate resources fairly among the processes.
- Hardware limitations: If the hardware of the system is not sufficient to support the workload of the kernel and user processes, the kernel may become a bottleneck as it tries to compensate for the limited resources.

To address a kernel bottleneck, it may be necessary to optimize the kernel, upgrade the hardware, reduce the system load, or change the workload of the system.

11.Kernel Level Threads

ANSWER:

Kernel-level threads, also known as kernel threads or system threads, are threads that are managed and scheduled by the operating system kernel. They are implemented directly in the kernel and are managed as part of the kernel's process control block (PCB) data structure.

In contrast, user-level threads are implemented in user space and are managed by a thread library, such as the pthreads library. They are scheduled by the operating system, but they do not appear as distinct entities to the kernel. Instead, they are multiplexed onto a smaller number of kernel-level threads, which are then scheduled by the kernel.

Kernel-level threads have several advantages over user-level threads:

- They can be scheduled more efficiently by the kernel, which has a better view of the system's resources and can make more informed decisions about thread scheduling.
- They can take advantage of multiple CPU cores and processors more effectively, as they can be scheduled independently by the kernel.
- They can make use of special hardware features, such as real-time scheduling or advanced debugging features, that are not available to user-level threads.

However, kernel-level threads also have some disadvantages. They require more overhead to create and manage, as they are part of the kernel's data structures. They also require more system resources, as each kernel-level thread requires its own kernel stack and PCB. This can make it more difficult to create and manage large numbers of threads.

12.Critical solution problem and it's approches are necessary

ANSWER:

A critical section problem is a situation where multiple processes or threads need to access shared resources, and the integrity of the system depends on how

these accesses are coordinated. The critical section problem occurs when two or more processes attempt to access a shared resource simultaneously, which can lead to race conditions or other synchronization issues.

There are several approaches to solving the critical section problem:

- Mutual exclusion: This approach ensures that only one process or thread can access the critical section at a time. It can be implemented using locks, semaphores, or other synchronization mechanisms.
- Priority inversion: This approach allows processes or threads with higher priority to access the critical section first, preventing lower-priority processes from indefinitely waiting for the resource. It can be implemented using priority inheritance or priority ceiling protocols.
- Deadlock prevention: This approach prevents processes or threads from entering into a state where they are waiting indefinitely for a resource that is held by another process or thread. It can be implemented using lock ordering, resource allocation graphs, or other techniques.
- Deadlock detection and recovery: This approach allows processes or threads to detect when they are in a deadlock state and take corrective action to break the deadlock. It can be implemented using a deadlock detection algorithm or a watchdog process.
- Timing constraints: This approach allows processes or threads to access the critical section only within certain time intervals or with certain periods of separation. It can be implemented using time-based semaphores or other timing mechanisms.

13.Linux commands

- Fg
- Kill
- Ps
- Bg
- Cp1

ANSWER:

- fg: The fg command stands for "foreground." It is used to bring a background or suspended process to the foreground, allowing it to receive input from and display output to the terminal.

- kill: The kill command is used to send a signal to a process, causing it to terminate. It can be used to terminate processes that are stuck or behaving unexpectedly.
- ps: The ps command stands for "process status." It is used to display information about the processes running on a system, including the process ID, command name, and status.
- bg: The bg command stands for "background." It is used to resume a suspended process in the background, allowing it to continue running without taking up the terminal.
- cp: The cp command stands for "copy." It is used to copy files and directories from one location to another. It can also be used to create copies of files with different names or in different directories. For example, to copy the file file.txt from the current directory to the /tmp directory, you could use the command cp file.txt /tmp.