1) Difference between signal and interrupt:

- Signal
  - generated by software
  - used to notify a process that an event has occurred, such as a keyboard input or a timer expiration
  - Signals are optional process can choose to ignore a signal, or it can handle the signal in some way
  - handle the signal in some way, such as by displaying a message or by terminating the process
- Interrupt
  - generated by hardware
  - used to notify the CPU that an external device needs attention, such as a disk drive or a network card
  - Interrupts are mandatory. The CPU must handle an interrupt immediately
  - handled by the CPU.
  - The CPU will typically save the current state of the CPU, then switch to a special interrupt handler routine
  - Once the interrupt handler has finished, the CPU will restore the previous state of the CPU and continue executing the interrupted process

2) What is Disk copy of Inode? List out its fields.

- disk copy of an inode (index node) is a data structure
- used to represent a file in a file system
- inode contains metadata about the file, such as its size, permissions, timestamps, and pointers to the data blocks that store the actual content of the file
- disk copy of an inode is stored on disk

- used by the operating system to access and manipulate the file
- fields typically found in a disk copy of an inode can vary depending on the specific file system implementation
- some common fields found in an inode :
    1. **Inode number**: A unique identifier for the inode. used by the operating system to locate the inode on the disk
    2. **File type**: file type indicates the type of file, such as regular file, directory, or symbolic link
    3. **File owner**: The user who owns the file. special privileges, such as the ability to delete the file or change its permissions
    4. **File group**: The group that owns the file. privileges as the file owner
    5. **File permissions:** The permissions that control who can access the file. permissions combination of three characters: read (r), write (w), and execute (x)
    6. **File size**: The size of the file in bytes
    7. **File creation time**: The time at which the file was created

3) If reference count of an incore inode is greater than 0,it can be put on free list of inode.

- Reference count indicating the number of instances of the file that are active
- Inode can be put on free list only if its reference count is 0
- Meaning that the kernel can reallocate the in-core inode to another disk inode
- Free list of inodes thus serves as a cache of inactive inodes
- Kennel places the inode on the free list of inodes effectively caching the inode in case it is needed again soon
- So if reference count of an incore inode is greater than 0 it can not be put on free list of inode

4) The process enters the kernel context(Layer 1) when it executes the system call.

- When a process makes a system call, it enters the kernel context
- kernel context is a special mode of operation that allows the process to access kernel resources, such as hardware devices and memory

- kernel context is divided into layers, with each layer representing a different level of privilege
- first layer is the user-level context - which is where the process was executing before it made the system call
- second layer is the kernel context - which is where the process is executing while it is in the kernel
- third layer and above are reserved for interrupts and other special events
- When a process makes a system call, the kernel saves the current state of the process's user-level context, including the process's registers, memory, and stack.
- kernel then switches to the kernel context and executes the system call.
- When the system call is complete, the kernel restores the process's user-level context and returns control to the process
- kernel context is a critical part of the operating system.
- allows the operating system to protect the process's user-level context from unauthorized access and to ensure that the process does not have access to kernel resources that it is not authorized to use
- so it’s true process enters the kernel context(Layer 1) when it executes the system call

5) An unnamed pipe is created using first two entries of UFDT.

- unnamed pipe is a unidirectional communication channel between two processes
- created using the `pipe()` system call
- which returns two file descriptors, one for the read end of the pipe and one for the write end
- read end of the pipe can be used to read data that has been written to the write end of the pipe by another process
- first two entries of the UFDT (User File Descriptor Table) are used to store the file descriptors for the read and write ends of the pipe
- UFDT(User File Descriptor Table) is maintained by the operating system for each process
- UFDT(User File Descriptor Table) is a data structure that is used to store information about open files

- maintained by the kernel and is accessible to user-space processes through the fcntl() system call
- The read end of the pipe is assigned the lowest unused file descriptor, typically 0 (stdin)
- while the write end is assigned the next lowest unused file descriptor, which is 1 (stdout)

6) What are different ways of inter process communication.

- There are many different ways of inter-process communication (IPC) in advanced operating systems
- Some of the most common methods include
    1. **Shared memory:**
        - two or more processes share a common block of memory
        - allows processes to share a region of memory, which can be accessed by multiple processes simultaneously
        - allows them to read and write to the same data
        - useful for tasks such as communication between a parent and child process, or between two different applications
    2. **Message passing:**
        - processes send messages to each other
        - Messages can be simple data structures, or they can be complex objects
        - used for communication between different applications, or between different parts of a large application
        - done either through shared memory or through a specific message passing mechanism provided by the operating system
        - Message passing can be synchronous (blocking) or asynchronous (non-blocking)
    3. **Pipes:**
        - allows processes to communicate with each other in a one-way fashion
        - used for communication between parent and child processes, or between two different applications
        - data is transferred between processes in a sequential manner

- pipe has two ends, a read end and a write end.
- One process writes data to the write end, and the other process reads from the read end.
- Pipes can be either anonymous or named

4. **Semaphores:**
    - allows processes to synchronize their access to shared resources
    - used to prevent two processes from accessing the same data at the same time
    - used for signaling or for process synchronization
5. **Sockets:**
    - allows processes to communicate over a network or between processes on the same machine
    - used for communication between different computers, or between different applications on the same computer
    - provide a standard interface for communication and can be either stream-based (TCP) or datagram-based (UDP)
    - enable interprocess communication across different machines as well

7) explain race condition in ialloc algorithm

- ialloc algorithm is for allocating memory in an operating system
- first checking if there is any free memory in the system
- If there is, the algorithm allocates that memory and returns it to the caller.
- If there is no free memory, the algorithm blocks the caller until some memory becomes available
- race condition in the ialloc algorithm when two or more processes try to allocate memory at the same time
- This can happen if two processes are both trying to access the same memory page at the same time.
- If this happens, the operating system will not be able to determine which process should get the memory.
- This can lead to data corruption or other problems.
- One way is to use a mutex lock to protect the memory allocation code

- use a semaphore to signal when memory is available
- Process 1:
  - Check if there is free memory
  - If there is, allocate the memory and return it to the caller
  - If there is no free memory, block the caller until some memory becomes available
- Process 2:
  - Check if there is free memory
  - If there is, allocate the memory and return it to the caller
  - If there is no free memory, block the caller until some memory becomes available

8) What is context of a process? What are different situations under which kernel needs to save the context of a process.

- context of a process refers to the set of information that the operating system needs to save and restore in order to switch between different processes
- includes the values of CPU registers, program counters, stack pointers, and other relevant information that is necessary for the process to resume its execution
- kernel, which is the core component of an operating system, is responsible for managing processes and ensuring their proper execution
- kernel needs to save the context of a process in the following situations
  - When the process is preempted by another process
    - because another process has a higher priority
    - kernel saves the context of the preempted process so that it can be restored later
  - When the process makes a system call
    - requesting a service from the kernel
    - saves the context of the process so that it can be restored later, after the system call has been completed
  - When the process blocks on an I/O operation
    - process blocks on an I/O operation, it is waiting for the I/O operation to complete

    - kernel saves the context of the process so that it can be restored later, after the I/O operation has completed
  - When the process terminates
    - kernel saves its context so that it can be cleaned up
    - includes freeing the process's memory and releasing any resources that the process is using
- When the kernel saves the context of a process, it stores the following information:
  - value of the process's registers
  - contents of the process's memory
  - state of the process's kernel data structures
- kernel then restores the context of a process when it is ready to run again. This allows the process to continue from where it left off

9) Why is virtualization useful?

-

10) Explain usage of SIGALRM system call.

- used to set a timer in an operating system
- When the timer expires, the SIGALRM signal is sent to the process that set the timer
- SIGALRM signal can be used to implement a variety of features, such as timeouts, periodic events, and load balancing
- used to implement timeouts
- timeout is a period of time after which an event will occur.
- SIGALRM system call can also be used to implement periodic events
- periodic event is an event that occurs at regular intervals
- SIGALRM system call can be used to implement load balancing
- Load balancing is a technique for distributing work evenly among a group of servers.
- A load balancer might use the SIGALRM system call to send requests to different servers in a round-robin fashion
- SIGALRM system call takes one argument, which is the number of seconds to wait before sending the SIGALRM signal

- SIGALRM system call returns the number of seconds remaining before the next alarm
- SIGALRM system call can be used to set a single alarm or to repeatedly send the SIGALRM signal at regular intervals
- For example, alarm(5) will set an alarm to go off after 5 seconds

11) Explain race condition in unlink system call.

- race condition is a situation where the outcome of a computation depends on the order or timing of events that are not synchronized
- race condition can occur in various system calls, including the unlink system call
- unlink system call is used to delete or remove a file from a file system
- takes the file path as an argument and removes the corresponding file entry from the directory
- context of the unlink system call, a race condition can occur when two processes are trying to delete the same file
- one process deletes the file before the other process has a chance to open it, the second process will get an error when it tries to open the file
- avoid race conditions, it is important to use the fsync system call before deleting a file
- fsync system call flushes all changes to the file to disk, ensuring that the file is in a consistent state before it is deleted
- fsync system call is a simple way to avoid race conditions when deleting files

12) Suppose the user executes following shell commands

i. mount /dev12/dsk11 /usr1

ii. cd /usr1/src/uts

iii. cd../../..

Describe what kernel does during execution of third command.

- The mount command tells the kernel to attach the filesystem found on the device /dev12/dsk11 to the directory /usr1
- kernel will first check to see if the user has permission to mount the device. If the user does have permission, the kernel will then check to see if there is already a filesystem mounted on the directory /usr1.
- If there is not, the kernel will create a new filesystem on the device /dev12/dsk11 and then mount it on the directory /usr1. If there is already a filesystem mounted on the directory /usr1, the kernel will replace the existing filesystem with the new filesystem from the device /dev12/dsk11
- cd command tells the kernel to change the current working directory to the parent directory of the current working directory.
- In this case, the current working directory is /usr1/src/uts. The parent directory of /usr1/src/uts is /usr1.
- Therefore, the kernel will change the current working directory to /usr1
- When the user executes the command "cd ../../.." in the shell, the kernel performs the following actions
    - kernel receives the cd ../../.. command from the shell
    - kernel parses the command and determines that the user wants to change to the parent directory of the current directory
    - kernel locates the parent directory of the current directory
    - kernel changes the current directory to the parent directory
- current directory is /usr1/src/uts.
- parent directory of /usr1/src/uts is /usr1. Therefore, the kernel changes the current directory to /usr1

13) What are the benefits of using virtual machine?

-

14) Explain algorithm for setjump and longjump.

- setjmp and longjmp functions are used to transfer control to a specific point in a C program
- setjmp and longjmp functions can be used to implement error handling, debugging, and other advanced programming techniques

- saves the current state of the program, including the value of all registers, and returns a value that can be used by longjmp to restore the program to that state
- setjump
  - Save the current state of the program in a buffer
  - Return 0
  - sets up a "jump point" in the program flow
  - Save the current state in data structure called a "jmp_buf"
  - typically represented as an array or structure that holds the necessary information to restore the program state
  - int setjmp(jmp_buf env);
- longjump
  - Restore the program state from the buffer
  - Jump to the specified address
  - performs a non-local jump to a previously saved setjmp() context
  - resume execution at the corresponding setjmp() call
  - void longjmp(jmp_buf env, int value);

15) Discuss the following cases with example:

- **Process sleeping at not interruptible priority and return zero.**
- **Process sleeping at interruptible priority and return zero.**
- **Process sleeping at not interruptible priority and return one.**
- **Process sleeping at interruptible priority and return one**

- processes can be put to sleep, allowing them to wait for certain events or conditions to occur before they can continue executing
- priority of a process determines its importance and the order in which it receives system resources
- process is put to sleep, it may return a value indicating its current state or the result of its sleep operation
- **Process sleeping at non-interruptible priority and returning zero :**
  - process is sleeping at a priority that is not interruptible
  - process cannot be woken up by a signal

- signal arrives while the process is sleeping, the signal will be queued and the process will continue to sleep.
- The process will only wake up when the event that it is waiting for occurs
- Ex process waiting for a disk I/O operation to complete. Once the disk I/O is finished, the process wakes up and returns zero, indicating successful completion

- **Process sleeping at interruptible priority and returning zero**
  - process is sleeping at a priority that is interruptible
  - process can be woken up by a signal
  - signal arrives while the process is sleeping, the process will wake up and handle the signal.
  - The process will then continue from where it left off before it was interrupted
  - Ex process waiting for user input. If the user provides the required input, the process wakes up and returns zero, indicating successful completion. However, if a higher priority task interrupts the process before it receives the input, it may also return zero to indicate that it was interrupted
- **Process sleeping at non-interruptible priority and returning one**
  - process is sleeping at a priority that is not interruptible
  - process cannot be woken up by a signal.
  - signal arrives while the process is sleeping, the signal will be lost.
  - The process will continue to sleep until the event that it is waiting for occurs
  - Ex process tries to acquire a lock or resource that is currently held by another process, it may be put to sleep and return one when it wakes up, indicating that it was unable to acquire the required resource
- **Process sleeping at interruptible priority and returning one**
  - process is sleeping at a priority that is interruptible
  - process can be woken up by a signal.
  - signal arrives while the process is sleeping, the process will wake up and handle the signal.
  - process will then return 1 to indicate that it was woken up by a signal

- ex process is waiting for a network connection to be established, but the connection attempt fails or times out, the process may wake up and return one to indicate the failure to establish the connection

16) Write a note on Semaphores.

- semaphore is a variable or abstract data type used to control access to a common resource by multiple threads and avoid critical section problems in a concurrent system such as a multitasking operating system.
- Semaphores are a type of synchronization primitive
- semaphore is typically implemented as an integer variable that can be either 0 or 1.
- A value of 0 indicates that the resource is unavailable
- value of 1 indicates that the resource is available
- take on a non-negative integer value and supports two primary operations: wait (P) and signal (V)
- wait operation decrements the semaphore value
- signal operation increments the semaphore value
- used to solve the critical section problem
- problem that arises when multiple threads need to access a shared resource.
- If the threads access the resource at the same time, they can corrupt the data in the resource
- critical section problem by ensuring that only one thread can access the resource at a time.
- difficult to debug
- can lead to deadlocks

17) What is nice system call?

- system call called nice() used to change the priority of a process
- priority of a process is a number that determines how much CPU time the process is given
- higher priority means that the process will be given more CPU time, while a lower priority means that the process will be given less CPU time

- nice system call takes one argument, which is the increment to the process's nice value
- positive increment will decrease the process's priority
- negative increment will increase the process's priority
- range of nice values is from -20 to 19, with -20 being the highest priority and 19 being the lowest priority
- used by users to run time-consuming processes in the background
- used by system administrators to control the priority of system processes
- user can use the nice system call to run a time-consuming process in the background

18) What are daemons?

-

19) What is message queue?

- message queue is a software structure that allows two or more processes to communicate with each other without having to be aware of each other's existence
- Messages are placed in the queue by one process and retrieved by another process.
- The messages are stored in the queue until they are retrieved, and they are processed in the order in which they are received
- used to improve the performance and reliability of a system
- if two processes need to communicate with each other, but one process is busy, the other process can place a message in the queue and the first process can retrieve the message when it is ready.
- This allows the two processes to communicate without having to wait for each other, which can improve performance
- if one process fails, the other processes can still communicate with each other by placing messages in the queue.
- The failed process can then retrieve the messages when it restarts
- help to reduce complexity by providing a simple way for processes to communicate

20) What is an unreliable signal?

- unreliable signal is a signal that can be lost or ignored by the operating system
- Unreliable signals can be caused by various factors, including interference, noise, signal degradation, or transmission errors
- This can happen operating system is busy or if the signal is received while the process is in a critical section
- Unreliable signals can be a problem for processes that need to be able to respond to signals quickly
- Dealing with unreliable signals is an important aspect of operating system design
- Ex a process that is handling user input needs to be able to respond to keyboard interrupts quickly, or else the user will not be able to type anything
- number of ways to deal with unreliable signals
- use a signal handler, which is a function that is called when a signal is received
- use a signal mask, which is a set of signals that are blocked from being received by the process

21) What is Incore copy of Inode? List all its fields.

- incore copy of an inode is a copy of an inode that is stored in memory
- used by the operating system to keep track of the file's metadata, such as its size, permissions, and ownership
- incore copy is updated whenever the file is modified, and it is used to perform operations on the file, such as reading, writing, and deleting
- fields of an incore copy of an inode in advance operating system
    1. **Inode number**: A unique identifier for the inode. used by the operating system to locate the inode on the disk
    2. **File type**: file type indicates the type of file, such as regular file, directory, or symbolic link
    3. **File owner**: The user who owns the file. special privileges, such as the ability to delete the file or change its permissions

4. **File group**: The group that owns the file. privileges as the file owner
5. **File permissions:** The permissions that control who can access the file. permissions combination of three characters: read (r), write (w), and execute (x)
6. **File size**: The size of the file in bytes
7. **File creation time**: The time at which the file was created
8. **Reference count:** The number of processes that have the file open

22) What is the syntax of nice system call?

- Syntax:

  ```
  #include <unistd.h>
  int nice(int increment);
  ```
- system call called nice() used to change the priority of a process
- priority of a process is a number that determines how much CPU time the process is given
- higher priority means that the process will be given more CPU time, while a lower priority means that the process will be given less CPU time
- nice system call takes one argument, which is the increment to the process's nice value
- positive increment will decrease the process's priority
- negative increment will increase the process's priority
- range of nice values is from -20 to 19, with -20 being the highest priority and 19 being the lowest priority
- used by users to run time-consuming processes in the background
- used by system administrators to control the priority of system processes
- user can use the nice system call to run a time-consuming process in the background

23) What is “no delay” flag in pipe?

- "no delay" flag in a pipe is a way to improve the performance of communication between two processes.
- When this flag is set, the kernel will not wait for the pipe to fill up before sending data to the other process.

- This can improve performance in cases where the two processes are communicating frequently and there is a lot of data to be transferred
- To set the "no delay" flag on a pipe, you can use the O_NONBLOCK flag when opening the pipe
- Once the pipe has been opened with the "no delay" flag set, data can be written to the pipe without having to worry about it being blocked.
- The data will be sent to the other process as soon as it is available
- setting this flag can also lead to data loss if the other process is not able to keep up with the data being sent to it
- "no delay" flag can be set on any file descriptor, not just pipes

24) List the fields of region table entries

- fields of region table entries in advanced operating systems can vary depending on the specific operating system, but some common fields include
  - Region start address: The virtual address of the start of the region
  - Region end address: The virtual address of the end of the region
  - Region size: The size of the region in bytes
  - Region type: The type of region, such as code, data, or stack
  - Region permissions: The permissions for the region, such as read, write, and execute
  - Region owner: The process that owns the region
  - Region status: The status of the region, such as loaded, unloaded, or shared
  - Region reference count: The number of processes that are currently using the region
- allows the operating system to keep track of all of the regions of memory that are currently in use

25) When System call is executed, context layer is pushed on user stack.

- system call is executed, the kernel pushes a context layer on the user stack in advance

- when a system call is executed, the context layer is not necessarily pushed on the user stack.
- The specific implementation may vary depending on the operating system and its design
- This is done to ensure that the kernel can restore the process's state if an error occurs during the system call
- context layer includes the following information:
  - process's current state, such as the values of its registers and flags
  - process's current state, such as the values of its registers and flags
  - process's open files and other resources
- If an error occurs during the system call, the kernel can use this information to restore the process's state and return it to user mode.
- This helps to prevent data loss and other problems that can occur when a process is interrupted
- context layer is pushed on the user stack by the CPU hardware
- When the CPU executes a system call instruction, it automatically pushes the current context onto the stack.
- ensures that the kernel can restore the process's state even if the CPU is interrupted by an external event, such as a hardware interrupt
- context layer is popped off the stack by the kernel when the system call completes.
- This restores the process's state and returns it to user mode
- context layer is a critical part of the system call mechanism
- ensure that processes can be interrupted and resumed safely

26) Explain the difference between virtualization and containerization.

- Virtualization and containerization are two technologies that allow multiple applications or workloads to run on a single physical machine or host
- Virtualization
  - creates a layer of abstraction between the physical hardware and the operating system
  - allows multiple operating systems to run on the same physical hardware, each in its own virtual machine

    - Virtual machines are isolated from each other, so they cannot interfere with each other's resources
    - virtualization is a more heavyweight technology than containerization
    - require more resources than containers
    - more complex to set up and manage
- Containerization
    - technology that packages an application and its dependencies into a single unit called a container
    - Containers share the same operating system kernel as the host machine, but they have their own isolated filesystem and network stack
    - allows multiple containers to run on the same host machine without interfering with each other
    - Containers are a lighter-weight alternative to virtual machines
    - require fewer resources and are easier to set up and manage

27) What are the three different ways to handle a signal?

- Handle the signal with a signal handler
    - most common way to handle a signal
    - signal handler is a function that is called when the process receives the signal
    - signal handler can do whatever tasks you want, such as logging the signal, displaying a message, or taking corrective action
    - handler can perform custom actions in response to the signal
- Ignore the signal
    - process will not take any action when it receives the signal
    - useful for signals that are not important or that you do not want to handle
    - program continues its execution without interruption
- Block the signal
    - process will not receive the signal
    - useful for signals that you do not want to be interrupted by

28) How a process can synchronize its execution with the termination of a child process.

- There are two ways for a process to synchronize its execution with the termination of a child process in advance
- Inter-process Communication mechanisms like pipes, message queues, or shared memory can be used for synchronization

1. Wait for the child process to terminate :
   - parent process can use the `wait()` system call to wait for the child process to terminate.
   - The `wait()` system call will block the parent process until the child process terminates
   - Ex

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
  pid_t pid;

  // Create a child process.
  pid = fork();

  // If this is the child process, exit.
  if (pid == 0) {
    exit(0);
  }

  // Wait for the child process to terminate.
  int status;
  wait(&status);

  // Print the child process's exit status.
  printf("Child process exited with status %d\n",
WEXITSTATUS(status));

  return 0;
}
```

2. Use a signal:
   - parent process can use a signal to be notified when the child process terminates.
   - The parent process can then use the `signal()` system call to register a handler for the signal.
   - The handler will be called when the child process terminates
   - Ex

```c
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void child_handler(int signal) {
  printf("Child process terminated\n");
}

int main() {
  pid_t pid;

  // Create a child process.
  pid = fork();

  // If this is the child process, exit.
  if (pid == 0) {
    exit(0);
  }

  // Register a signal handler for SIGCHLD.
  signal(SIGCHLD, child_handler);

  // Continue executing.
  while (1) {
    // Do something...
  }

  return 0;
}
```

29) Explain the block diagram of system kernel.

- block diagram of a system kernel in advanced operating systems is a representation of the different components of the kernel and their relationships to each other
- **The process control subsystem**:
  - responsible for managing the execution of processes, including scheduling, dispatching, and synchronization
- **The memory management subsystem:**
  - responsible for managing the allocation and deallocation of memory to processes
  - includes allocating memory to processes when they are created
  - freeing memory when processes are terminated
  - managing the paging of memory to and from disk
- **The file system subsystem:**
  - responsible for managing the storage and retrieval of files
  - includes creating, deleting, opening, closing, reading, and writing files

- **The device driver subsystem:**
  - responsible for managing the interaction between the kernel and hardware devices
  - includes loading device drivers, initializing devices, and transferring data between the kernel and devices
- **The interrupt handling subsystem:**
  - responsible for handling interrupts from hardware devices
  - includes identifying the source of the interrupt, servicing the interrupt, and resuming the execution of the interrupted process
- **The system call interface:**
  - This interface provides a way for user programs to interact with the kernel
  - includes providing a way for user programs to request services from the kernel, such as allocating memory, creating files, and reading data from devices

30) What do you mean by host and guest operating systems?

- a host operating system is the primary operating system that is installed on a physical computer.
- A guest operating system is an operating system that is installed in a virtual machine (VM) on top of the host operating system
- host operating system is responsible for managing the physical hardware resources of the computer, such as the CPU, memory, and storage.
- The guest operating system is isolated from the physical hardware and runs in its own virtual environment.
- allows multiple guest operating systems to run on the same physical computer without interfering with each other
- Host and guest operating systems can be of different types.
- For example, a host operating system can be Windows or Linux, and a guest operating system can be Windows, Linux, or macOS
- Host and guest operating systems can share the same physical hardware resources, such as CPU, memory, and storage
- Host and guest operating systems can be used to run different operating systems on the same physical computer

- Running multiple operating systems on the same physical computer can impact performance
- Managing host and guest operating systems can be complex
- Host and guest operating systems can be more expensive than running a single operating system on a physical computer

31) Draw and explain process sleeping on events and events mapping to addresses

- sleeping on events refers to a mechanism that allows a process or thread to suspend its execution and wait for a specific event to occur.
- This concept is often used in multitasking systems to efficiently utilize system resources
- When a process wants to sleep on an event, it calls the sleep() system call.
- The sleep() system call takes two arguments: the address of the event and the process's priority
- kernel stores the process's state and context in memory.
- The kernel then sets the process's state to sleeping and its priority to the value that was passed to the sleep() system call
- kernel then removes the process from the ready queue and adds it to the sleep queue
- When the event occurs, the kernel wakes up all of the processes that are sleeping on the event
- kernel then puts the processes back into the ready queue in order of priority
- kernel then selects the highest-priority process from the ready queue and schedules it to run

32) What do you mean by a process sleeping at interruptible priority? Explain in detail.

- interruptible priority sleeping process is a process that has voluntarily given up the CPU and is waiting for an event to occur.
- The process can be woken up by a signal or by the completion of the event it is waiting for
- processes are typically assigned a priority value.

- The priority value determines how likely the process is to be scheduled to run.
- A process with a higher priority is more likely to be scheduled than a process with a lower priority
- When a process sleeps, it can specify a priority value.
- If the priority value is above a certain threshold, the process cannot be woken up by a signal. This is known as an uninterruptible sleep.
- If the priority value is below the threshold, the process can be woken up by a signal. This is known as an interruptible sleep
- One reason is to allow other processes to run.
- Another reason is to avoid wasting CPU time if the event the process is waiting for is not likely to occur soon
- When a process sleeps at an interruptible priority, it is still considered to be running.
- process can still be affected by signals

33) Explain shared memory options in IPC.

- Shared memory is an inter-process communication (IPC) mechanism that allows multiple processes to access the same memory segment.
- useful for sharing data between processes, or for coordinating the execution of multiple processes
- two main types of shared memory
    1. Anonymous shared memory
        - not associated with any particular file or object
        - created by the operating system
        - used by any process that has the appropriate permissions
    2. File-backed shared memory
        - backed by a file on disk
        - process attaches to a file-backed shared memory segment, the operating system reads the contents of the file into memory
        - useful for sharing large amounts of data, or for sharing data that needs to be persistent across reboots
- Shared memory can be used to communicate between processes

- one process could write data to a shared memory segment, and another process could read the data from the shared memory segment
- database server could use shared memory to share data with client processes
- Shared memory is the fastest IPC mechanism because processes can access shared memory without having to go through the kernel
- used to share data between processes without having to copy the data

34) Explain sigqueue() function.

- sigqueue() function is a system call that allows a process to send a signal to another process, along with an additional value
- signal is delivered to the receiving process's signal handler, just like any other signal
- additional value is passed to the signal handler as an argument
- This can be used to pass information to the signal handler, such as the identity of the sending process
- used in advanced operating systems to implement inter-process communication (IPC)
- process might use sigqueue() to send a notification to another process when an event occurs
- receiving process could then use the additional value to determine the specific event that occurred
- used to implement a variety of IPC and real-time applications
- important to use it carefully, as it can also be used to disrupt the operation of other processes
- sigqueue() function requires the sending process to have appropriate permissions to send signals to the target process
- sigqueue() function enhances inter-process communication by enabling the exchange of signals along with additional data

35) Explain any four fields of incore copy of inode which is not present in disk copy of Inode.

- File offset:
  - stores the current offset of the file being read or written

    - necessary because the file may be split across multiple blocks on disk, and the kernel needs to know where to start reading or writing from
- Reference count:
    - stores the number of processes that have the file open
    - used to prevent the file from being deleted while it is still in use
- Inode status:
    - stores the current state of the inode, such as whether it is open, closed, or deleted
    - used by the kernel to manage the file system
- File attributes:
    - stores additional information about the file, such as its permissions, ownership, and creation time
    - information is not stored on disk because it is not necessary for the file to be stored or retrieved

36) Two different processes can share same FT entry.

- two different processes can share the same FT entry in advanced operating systems. This is called "file sharing"
- common way for processes to access the same data
- When two processes share the same FT entry, the operating system must take steps to ensure that the data is accessed safely and correctly
- When a process opens a file, the operating system creates a new FT entry for that process
- Two different processes can share the same FT (File Table) entry.
- FT contains information about the opened files, such as the file pointer and the file status flags.
- Since multiple processes can open the same file simultaneously, they can share the same FT entry

37) Pipe uses only direct blocks of inode for greater flexibility.

- Pipes are a special type of file that allows two processes to communicate with each other.
- created using the pipe() system call
- When a pipe is created, the kernel allocates two direct blocks for it.

- first block is used for the read end of the pipe and the second block is used for the write end.
- data that is written to the write end of the pipe is stored in the first block and the data that is read from the read end of the pipe is read from the first block
- kernel uses a circular queue to manage the data in the pipe.
- read and write pointers are used to keep track of where the data is being read from and written to.
- read pointer is always pointing to the next block of data that can be read from the pipe and the write pointer is always pointing to the next block of data that can be written to the pipe
- When a process reads from a pipe, the kernel reads the data from the block that the read pointer is pointing to.
- kernel then increments the read pointer and moves on to the next block of data.
- If there is no more data in the pipe, the kernel will return an error
- direct blocks of an inode are the first 10 blocks that are allocated to a file
- When a pipe is created, the kernel allocates two direct blocks for it
- first block is used for the read end of the pipe and the second block is used for the write end of the pipe
- allows the kernel to access the data in the pipe very quickly.
- allows the kernel to expand the size of the pipe without having to reallocate the inode

38) What is the use of nice system call?

- nice system call is used to change the priority of a process
- "nice" system call is used to adjust the priority of a process
- higher nice value means a lower priority
- lower nice value means a higher priority
- higher nice value means a lower priority, and a lower nice value means a higher priority
- used to improve the overall performance of a system by giving more CPU time to important processes and less CPU time to less important processes

- user can use the nice system call to run a long-running process in the background with a lower priority so that it does not interfere with other processes that are running in the foreground
- also be used to improve the fairness of a system by giving all processes a fair share of CPU time
- nice system call carefully, as giving a process too high of a priority can lead to system instability
- use the nice system call to give more CPU time to important processes and less CPU time to less important processes

39) Explain major and minor number.

- major and minor numbers in an operating system version number refer to the major and minor revisions of the operating system
- major number indicates a significant change to the operating system
- such as a new kernel or a major new feature
- minor number indicates a smaller change, such as a bug fix or a minor feature update
- operating system version 10.1.2 has a major number of 10, a minor number of 1, and a build number of 2.
- major number indicates that this is the 10th major revision of the operating system, the minor number indicates that this is the first minor revision of the 10th major revision
- used to help users and developers track the changes made to an operating system
- By utilizing major and minor numbers, the operating system can efficiently manage and control various devices or resources, such as disks, printers, network interfaces, or serial ports
- major and minor numbers provide a mechanism for device identification and management in advanced operating systems, enabling efficient and organized access to various hardware resources

40) Explain fork() system call.

- fork() system call is a system call that creates a new process from an existing process
- new process is called a child process and the original process is called the parent process
- fork() system call takes no arguments
- returns a value of 0 to the child process
- return negative for an error occurred
- return positive to the parent process
- child process inherits all of the resources of the parent process, including the memory, open files, and environment variables
- child process can then modify these resources as needed
- ex

```
#include <stdio.h>
#include <stdlib.h>

int main() {
  pid_t pid;

  pid = fork();

  if (pid == 0) {
    printf("I am the child process\n");
  } else {
    printf("I am the parent process\n");
  }

  return 0;
}
```

- When this program is executed, it will create two processes, a parent process and a child process.
- After the fork() system call, both the parent and child processes start executing from the point immediately after the fork() call
- The child process will print "I am the child process" and the parent process will print "I am the parent process"

41) “lseek can be used with pipes”. Justify

- lseek() function in the file system is used to change the current position of the file pointer within a file.
- pipes are not files and do not have an associated file position

- Pipes are a form of interprocess communication, allowing data to be passed between processes.
- They are typically used to connect the output of one process to the input of another process, forming a pipeline
- Therefore, it does not make sense to use lseek() with pipes, as the concept of a file position does not apply to them.
- Instead, pipes are typically used with functions like read() and write() to transfer data between processes

42) The kernel never invokes the grow region to increase the size of shared region.

- The kernel never invokes the grow region to increase the size of shared region in advance" is referring to the fact that the kernel does not automatically increase the size of a shared region when a process attempts to write to it
- kernel will only increase the size of the shared region when a process explicitly requests
- This is done to prevent the kernel from wasting memory by allocating more space than is actually needed
- Ex process that has a shared region of 1MB. If the process only ever writes to the first 500KB of the shared region, then there is no need for the kernel to allocate the remaining 500KB of memory. By only allocating memory as needed, the kernel can help to improve the overall performance of the system
- There are a few cases where the kernel will automatically increase the size of a shared region
- process attempts to write to an address that is outside of the current size of the shared region, then the kernel will automatically increase the size of the shared region to accommodate the write
- kernel may automatically increase the size of a shared region if it determines that the process is likely to need more space in the future

43) U-area contains the fields that must always be accessible to kernel.

- the term "U-area" refers to a user area or user segment
- It is a data structure maintained by the operating system for each user process
- contains information and resources that need to be accessible to the kernel while executing user-level code
- the u-area (user area) is a region of memory that is allocated to each process in the operating system
- contains information that is specific to the process, such as the process's open files, current directory, signal actions, and accounting information
- u-area is also used by the kernel to store information about the process, such as the process's state, priority, and scheduling information
- u-area is always accessible to the kernel, even when the process is not running
- because the kernel needs to be able to access this information in order to manage the process and to handle system calls
- u-area is typically located in kernel memory, so that the kernel can access it directly without having to go through the process's user space
- some of the fields that are typically found in the u-area:
  - Process ID (PID): A unique identifier for the process
  - Parent PID: The PID of the process that created this process
  - Process state: The current state of the process, such as running, waiting, or blocked
  - Priority: The priority of the process, which determines when it will be scheduled to run.
  - Open files: A list of the files that the process has opened.
  - Current directory: The current working directory of the process

44) What is docker? What are benefits of using docker over Hypervisor?

-

45) Explain blocking of signals.

- blocking of signals is a mechanism that prevents a process from receiving a signal
- This can be done for a variety of reasons, such as to prevent a process from being interrupted while it is performing a critical task
- protect a process from malicious signals
- There are two main ways to block signals
    1. Signal masking:
        - temporarily block all signals from being received by a process
        - done using the sigprocmask() system call
    2. Signal catching:
        - specify a function that will be called when a particular signal is received by a process
        - done using the signal() system call
- When a signal is blocked, it is not delivered to the process until the block is removed
- Signal blocking can be a useful tool for managing the behavior of processes in an advanced operating system
- important to use it carefully, as blocking signals can also prevent processes from responding to important events
- Signal blocking can prevent a process from being interrupted while it is performing a critical tas
- If a process is blocked from receiving signals, it can be more difficult to debug the process if it crashes
- This functionality can be useful in certain scenarios where signal interference or unauthorized communication needs to be managed

46) Explain architecture of UNIX Operating system.

- Unix operating system is a layered architecture
- it is composed of a number of layers that are stacked on top of each other
- The layers are:
    - Hardware
        - lowest layer and it consists of the physical components of the computer, such as the CPU, memory, and storage devices

        - responsible for providing the basic resources that the operating system needs to function, such as processing power, memory, and storage space
    - Kernel
        - next layer up and it is responsible for managing the hardware and providing basic services to the other layers
        - responsible for tasks such as memory management, process scheduling, and file I/O
    - System calls
        - responsible for providing a way for user programs to interact with the kernel
        - provide a way for user programs to request services from the kernel, such as reading files, writing files, and creating processes
    - Shell
        - command-line interpreter that allows users to interact with the operating system
        - provides a way for users to enter commands and to see the output of those commands
        - also responsible for starting user programs
    - User programs
        - highest layer and they are the applications that users run on the operating system
        - programs can be anything from word processors to web browsers to games
        - responsible for providing the functionality that users need to get their work done
- easily adapted to different hardware platforms and different needs.
- dividing the operating system into a number of independent modules that can be easily replaced or updated

47) Explain how to allocate and free a disk block

- When a file is created or data is written to disk, the file system needs to allocate disk blocks to store the file's contents

- When a new file is created, the file system determines the size of the file and calculates the number of disk blocks required to store its data
- file system searches for a contiguous sequence of free disk blocks that can accommodate the file's size. Different allocation strategies may be employed, such as contiguous, linked list, or indexed allocation
- Once a suitable sequence of free blocks is found, the file system marks them as allocated in the metadata structures, indicating that they are reserved for the file
- operating system first checks the free block list to see if there are any free blocks available
- If there are free blocks available, the operating system selects the first free block on the list and allocates it to the requesting process
- When a file is deleted, the file system locates the metadata associated with the file and marks it as deleted
- operating system then updates the free block list to reflect the fact that the block is no longer free
- adjacent free disk blocks can be merged together to form larger contiguous free blocks. This process is called coalescing, and it helps reduce fragmentation
- process that is done with the disk block calls the operating system's free() function
- operating system then updates the free block list to reflect the fact that the block is now free
- freed disk blocks are now available for future allocation to new files or data modifications

48) Explain docker architecture.

-