UNIT-2
PARALLEL PROGRAM CHALLENGES

1. What are the two common metrics for performance?
There are two common metric performances. They are: 1. Items per unit time: This might be transactions per second, jobs per hour, or some other combination of completed tasks and units of time and 2. Time per item: This is a measure of the time to complete a single task

2. Scalability:
Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth. For example, a system is considered scalable if it is capable of increasing its total output under an increased load when resources are added.

3. Data Sharing:
All parallel applications require some element of communication between either the threads or the processes. There is usually an implicit or explicit action of one thread sending data to another thread. For example, one thread might be signaling to another that work is ready for them.

4. Define Synchronization.
It is used to coordinate the activity of multiple threads. There are various situations where it is necessary: this might be to ensure that shared resources are not accessed by multiple threads simultaneously or that all were on those resources is complete before new work starts

5. Define region of code.
The region of code between the acquisition and release of a mutex lock is called a critical region. Code in this region will be executed by only one thread at a time

6. Define Data races.
They are the most common programming error found in parallel code. A data race occurs when multiple threads use the same data item and one or more of those threads are updating.

7. How to avoid data races?
It can be hard to identify data races, avoiding them can be very simple: Make sure that only one thread can update the variable at a time. The easiest way to do this is to place a *synchronization lock* around all accesses to that variable and ensure that before referencing the variable, the thread must acquire the lock.

8. What are synchronization primitives?
Process synchronization refers to the idea that multiple processes are to join up or handshake at a certain point, in order to reach an agreement or commit to a certain sequence of action.

- • Mutexes
- • Locks
- • Semaphores
- • Barriers

9. Mutex and locks with example:

UNIT-2
PARALLEL PROGRAM CHALLENGES

The simplest form of synchronization is a mutually exclusive (*mutex*) lock. Only one thread at a time can acquire a mutex lock, so they can be placed around a data structure to ensure that the data structure is modified by only one thread at a time.

Eg : Placing mutes locks around accesses to variables

int counter;

mutex_lock mutex;

```
void Increment()
{
acquire( &mutex );
counter++;
release( &mutex );
}
```

```
void Decrement()
{
acquire( &mutex );
counter--;
release( &mutex );
}
```

10. Define Semaphores and barriers.

Semaphores are counters that can be either incremented or decremented. They can be used in situations where there is a finite limit to a resource and a mechanism is needed to impose that limit.

There are situations where a number of threads have to all complete their work before any of the threads can start on the next task. In these situations, it is useful to have barrier where the threads will wait until all are present.

11. Differentiate deadlock and livelock:

DEADLOCK	LIVELOCK
A situation in which two or more processes are unable to proceed because each is waiting for one the others to do something.	A situation in which two or more processes continuously change their states in response to changes in the other process(es) without doing any useful work. It is somewhat similar to the deadlock but the difference is processes are getting polite and let other to do the work. This can be happen when a process trying to avoid a deadlock.
For example, consider two processes, P1 and P2, and two resources, R1 and R2. Suppose that each process needs access to both resources to perform part of its function. Then it is possible to have the following situation: the OS assigns R1 to P2, and R2 to P1. Each process is waiting for one of the two resources. Neither will release the resource that it already owns until it has acquired the other resource and performed the function requiring both resources. The two processes are deadlocked	

12. What are conditions under which a deadlock situation may arise?

A deadlock situation can arise if the following four conditions hold simultaneously in a system:

a. Mutual exclusion

b. Hold and wait

c. No pre-emption

UNIT-2
PARALLEL PROGRAM CHALLENGES

13. Define– thread. Mention the use of swapping.
Thread is placeholder information associated with a single use of a program that can handle multiple concurrent users. From the program's point-of-view, a thread is the information needed to serve one individual user or a particular service request. Thepurpose of swapping, or paging, is to access data being stored in hard disk and to bring it into the RAM so that it can be used by the application program

14. List the mechanisms for communication.
• Memory –Direct Communication
• Shared Memory communication
• Memory Mapped Communication
• Communication using condition variables
• Communication by signals
• Event Communication
• Message Queues
• Named Pipes
• Communication through Network stacks

15. Differentiate condiotion variables and signals.

CONDITION VARIABLES	SIGNALS
It communicates readiness between threads by enabling thread to be woken up when a condition become true. Without condition variables, the waiting thread would have to use some form of polling to cheek whether the condition had become true. Condition variables work in conjunction with a mutex. The mutex is there to ensure that only one thread at a time can access the variable.	Signals are a UNIX mechanism where one process can send a signal to another proceed and have a handler in the receiving process perform some task upon the receipt of the message

16. Steps to set up and write data in pipe.
Named pipes are file-like objects that are given a specific name that can be shared between processes. Any process can write into the pipe or read from the pipe. There is no concept of a “message”; the data is treated as a stream of bytes. The method for using a named pipe is much like the method for using a file: The pipe is opened, data is written into it or read from it, and then the pipe is closed.

17. Message Queue:
A message queue is a structure that can be shared between multiple processes. Messages can be placed into the queue and will be removed in the same order in which they were added. Constructing a message queue looks rather like constructing a shared memory segment.

18. Define Named pipes.

It provides a similar mechanism that can be controlled programmatically. They are file-like objects that are given a specific name that can be shared between processes. Any process can write into the pipe or read from the pipe. There is no concept of a "message": the data is treated as a stream of bytes. The method for using a named pipe is much like the method for using a file: the pipe is opened, data is written into it or read from it, and then the pipe is closed

19. Define Signals and Events.

Signals are UNIX mechanisms where one processor can send a signal to another processor and have a handler in the receiving process perform some task upon the receipt of the message. Windows has a similar mechanism for events. The handling of keyboard presses and mouse moves are performed through the event mechanism. Pressing one of the buttons on the mouse will cause a click event to be sent to the target window.

20. Why Algorithmic Complexity is important?

Algorithm complexity represents the expected performance of a section of code as the number of elements being processed increases. In the limit, the code with the greatest algorithmic complexity will dominate the runtime of the application.