Answers

1. The conditions required for **Deadlock:**
   - Mutual exclusion
     The resources involved must be unshareable ; otherwise, the processes would not be prevented from using the resource when necessary.
   - Hold and wait or partial allocation
     The processes must hold the resources they have already been allocated while waiting for other (requested) resources. If the process had to release its resources when a new resource or resources were requested, deadlock could not occur because the process would not prevent others from using resources that it controlled.
   - no pre-emption
     The processes must not have resources taken away while that resource is being used. Otherwise, deadlock could not occur since the operating system could simply take enough resources from running processes to enable any process to finish.
   - resource waiting or circular wait
     A circular chain of processes, with each process holding resources which are currently being requested by the next process in the chain, cannot exist. If it does, the cycle theorem (which states that "a cycle in the resource graph is necessary for deadlock to occur") indicated that deadlock could occur.

   Reference: http://nob.cs.ucdavis.edu/classes/ecs150-1999-02/dl-cond.html

2. The differences between **Deadlock** and **Starvation**:
   Deadlock refers to the situation when processes are stuck in circular waiting for the resources. On the other hand, starvation occurs when a process waits for a resource indefinitely. Deadlock implies starvation but starvation does not imply deadlock.

	Deadlock	**Starvation**
Definition	Deadlock occurs when none of the processes in the set is able to move ahead due to occupancy of the required resources by some other process	Starvation occurs when a process waits for an indefinite period of time to get the resource it requires.
Other name	Circular waiting	Lived lock
Arising conditions	These four conditions arising simultaneously – mutual exclusion, hold and wait, no-preemption and circular wit	Uncontrolled management of resources Process priorities being strictly enforces

		Use of random selection Scarcity of resources
Avoidance/ prevention Techniques	• Infinite resources • Waiting is not allowed • Sharing is not allowed • Preempt the resources • All Requests made at the starting	• Independent manager for each resources • No strict enforcement of the priorities • Avoidance of random selection • Providing more resources
Progress	No process can make progress	Apart from the victim process other processes can progress or proceed
Ending	Requires external intervention	May or may not require external intervention

Reference: http://www.differencebetween.info/difference-between-deadlock-and-starvation

3. There are many forms of threads but two types are in common use on fasteners.
   - **Machine Screw Threads**: -used on bolts, setscrews, machine screws and designed to mate with preformed threads in nuts or tapped holes. *Exceptions* may be thread forming screws like Taptite or self-drilling screws like Teksor thread cutters like Type 23's, which form or cut their own machine screw thread.
   - **Spaced Threads: -**uses on woodscrews, self-tapping screws, coach screws and Type 25 thread cutters. Designed to form its own thread, usually in a pre-drilled hole.
     Exceptions may be self piercing screws such as needle points or self-drilling screws like Type 17's which create their own hole; some Teksmay also have spaced threads.

     Reference: https://www.jamesglen.com.au/thread-types/

4. The differences between **Thread** and **Process :**

Process	**Thread**
• System calls involved in process. • Context switching required. • Different process have different copies of code and data. • Operating system treats different process differently. • If a process got blocked, remaining process continue their work. • Processes are independent. • Process run in separate memory space. • Communication between processes requires some time. • Processes don't share the memory with any other process. • Process have overhead.	• No system calls involved. • No context switching required. • Sharing same copy of code and data can be possible among different threads. • All user level threads treated as single task for operating system. • If a user level thread got blocked, all other threads get blocked since they are treated as single task to OS. (Noted: This is can be avoided in kernel level threads). • Threads exist as subsets of a process. They are dependent. • Threads run in shared memory space. And use memory of process which it belong to. • Threads share Code section, data section, Address space with other threads • Communication between processes requires less time than processes. • Threads share the memory with other threads of the same process • Threads have no overhead.

Reference::https://www.thecrazyprogrammer.com/2018/01/difference-process-thread.html