1) Can a process ever go from Ready state to Block state? Can a process ever go from Blocked state to running state? Discuss your answer with reasoning.

Let us understand this by a Analogy of day to day example. When we Visit a doctor we go wait in a lobby then when our turns come receptionist escort us to doctor's cabin. If our health is absolutely good or doctor can examine us without prescribing the test then we are good to drop out. Or else doctor prescribe the test like xray, or scanning. Once we get tests done, we cannot go directly to doctor irrespective of the doctor being free or busy. After the tests we are escorted to lobby by some errand boy, then when turn comes based on priority or mechanism used by doctor we are then escorted to the doctor.

Also we can never get tested without doctor's prescription. We need doctor's authorization to get test.

Similarly when the process goes from running state to block when it is waiting for some input. After receiving the input process goes into the ready queue or waiting queue first then when CPU is free that process resumes.

Also ready process can never go directly into the block state. The process gets blocked when it is waiting for some external input or authorization. The ready state can never directly go into blocked without first getting CPU's attention.

2) Difference between user threads and systems threads.

USER THREAD	SYSTEM THREAD
User threads run on top of a run-time system, which is a collection of procedures that manage threads.	The System Thread Do not need run-time system. As they are already ran by system it self.
When threads are managed in user space, each process needs its own private thread table to keep track of the threads in that process. The thread table consist of thread's program counter, stack pointer, registers, state, and so forth. The thread table is managed by the run-time system	As system threads are ran by system it self, kernel has a thread table that keeps track of all the threads in the system so they do not need separate thread table to keep track of process. The thread table is managed by kernel.

User level threads can be created and managed more quickly. Context switch time is less.	When a thread wants to create a new thread or destroy an existing thread, it makes a kernel call, which then does the creation or destruction by updating the kernel thread table therefore it is time consuming. Context switching takes more time.
thread scheduling very fast. They allow each process to have its own customized scheduling algorithm	They also scale better, since kernel threads invariably require some table space and stack space in the kernel, which can be a problem if there are a very large number of threads.
problem of how blocking system calls are faced.. Making system call is unacceptable since this will stop all the threads user-level thread packages is that if a thread starts running, no other thread in that process will ever run unless the first thread voluntarily gives up the CPU	Kernel threads do not require any new, nonblocking system calls.If one kernel thread perform blocking operation then another thread can continue execution.
User threads can be multithreaded but cannot take benefit of multiprocessing	Kernels can be multi threaded
OS can support	Kernel threads are specifically designed by operating system compatibility
In user level threads, switching between threads does not need kernel mode privileges.	When a thread at the kernel level is halted, the kernel can schedule another thread for the same process.
Multithreaded applications on user-level threads cannot benefit from multiprocessing.	Multithreaded applications on user-level threads cannot benefit from multiprocessing.

3) Solve the following using Shortest Job First Scheduling algorithm and find the average waiting time.

	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

For Non Preemptive SJSF

- At time t=0 , P1 will arrive and directly go for execution.

P1

t=0

- At time t=1 as no process is arriving , P1 will keep on executing.

  At time t=2 as P2 is arrived and ready but as P1 is already executing it will continue executing and p2 will wait in waiting queue.

  Waiting Queue

P2

  At time t=3 , p1 will be executing and p2 will be in wait queue

  At time t=4, p3 will arrive but P1 is having more 3 units of executing remaining will execute and P3 will get added in wait queue.

  In waiting queue as burst time of P2 is 4 and P3 is 1, which is less than P2. P3 will get priority for processing.

Waiting Queue

P3	P2			

P1

t=0

At time t=5, P5 will arrive ready to execute, but as still 2 units of execution time of P1 is remaining P1 will continue execution and P5 will go into wait Queue.
Same comparing burst time of 3 processes and giving priority to lowest burst time. P3 will get priority but P2 & P4 is having same burst time, priority is given to P2 as it was already on queue.

P3	P2	P4	

P1

t=0

At time t=6, still same P1 is executing.

At time t=7, P1 will complete execution and P3 will start execution, now P2,P4 will still be in the waiting queue

Waiting Queue

P2	P4

Execution Queue

P1	P3

t=0 t=7

At t=8, P3 will complete execution as its burst time was on 1 unit. It will leave CPU.
Now P2 will start executing on CPU.

P1	P3	P2

t=0 t=7 t=8

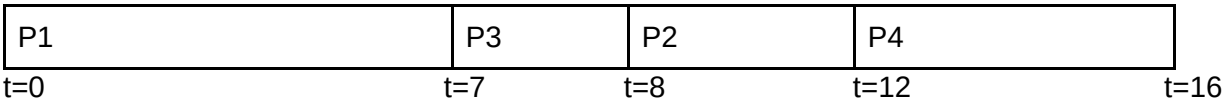
At t=9, P2 will keep on executing as its burst time is 4, it still have not completed execution.
P4 will be in wait queue

At t=10, P2 will keep on executing as its burst time is 4, it still have not completed execution.
P4 will be in wait queue

At t=11, P2 will keep on executing as its burst time is 4, it still have not completed execution. P4 will be in wait queue

At t=12, P2 will complete its executing giving CPU to execute Process P$ having Burst time of 4.
Waiting queue will be empty.

Execution queue



Finally at t=16 all processes will complete their execution.

Calculating the waiting time for each process
Waiting time (wt) = total time spent in ready queue. = time execution started - time of arrival.

P1(wt)= 0-0=0
P2(wt)= 8-2=6
P3(wt)= 7-4=3
P4(wt)= 12-5=7

Average waiting time = sum of total time
—----------
No of process

= 0+6+3+7
—---------
4

= 16
—
4

= 4

Therefore average waiting time is 4.