Q-3} Everytime it gives a different value, as new ID is been creating. The program returns the process ID of the child process to the parent process.

semaphore then checks it. This can give wrong output in terms of wait

Q-4) Mode switch between threads may be cheaper than a mode switch between processes.

→ The memory management is much simpler for threads than for processes. Threads share their memory so during mode switching memory information does not have to be exchanged ; pages and page table should not have to be switched. This makes the thread context switch much cheaper than for processes. In case of processes, the memory pieces, pages have to be exchanged. Also threads easily exchange their local variables within its scope but exchange value between process stole more CPU cycles.

Q-5) 3 advantages of ULTs over KLTs.

①→ Thread switching does not require kernel mode privileges because all of the thread management data structures are within user address space of a single process. Therefore, the process does not switch to the kernel mode to do thread managem This saves the overhead of two mode switches.

②→ Scheduling can be application specific. One application may benefit most from a simple round robin scheduling algorithm, while another might benefit from [illegible] time.

a priority based scheduling algorithm. The scheduling algorithm can be tailored to the application without disturbing the underlying OS scheduler.

③ ULTs can run on any operating system. There are no changes required to the underlying kernel to support ULTs. The thread library is a set of application-level functions shared by all applications.

Q-6) 2 disadvantages of ULTs over KLTs.

→ ① In an OS, many system calls are blocking. So, when a ULT executes a system call, not only is that thread blocked, ~~but~~ but also the threads will be blocked within the process.

② In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing. A kernel assigns one process to only one processor at a time. Therefore, only a single thread within a process can execute at a time. In effect, we have application level multiprogramming within a single process.

7) User process functions seperately from kernel processes. That is thread structure of a process is not visible to the OS/kernel, (which schedules on the basis of process. The kernel continues to schedule the process as a unit and assigns a single execution state to that process. So, when once one thread is blocked, the whole process is blocked and consequently all threads in that process are blocked.

8) As in this environment we have one to one mapping, context switching will be easier. Though multiprogramming does not play a role in this system, but multi-threading will become faster as they do not have to wait for other threads.

Q-9] The thread will also terminate if the process it is running in terminates. The thread is dependent upon the process it is running. If the process dies the thread dies.

Q-10] Competing - Compete for resources. For eg. two independent applications may both want to access the same disk or file or printer. The OS must regulate these accesses

Co-operating - Share resources. May or may not be aware of each other. Some processes are designed to cooperate together on the same activity and share resources. They may also be aware of each other by process id

-11] Strong processes specify the order in which processes are removed from the queue, which gaurantees avoiding starvation. Weak semaphores do not specify the order in which processes are removed from the queue

12] Monitor - Monitor is a programming language construct that provides equivalent functionality to that of semaphores and that is easier to control.

P-13} → Blocking send or recieve for messages means that the sender and/or reciever is blocked unit the message is delivered.
→ In a non-blocking send/ recieve, neither the sender or reciever has to wait.

14} Yes, because busy waiting consumes useless instruction cycles. However, in a particular case, if a process comes to a point in the program where it must wait for a condition to be satisfied and if that condition is already satisfied. Then the busy wait will find that out immediately whereas the blocking wait will consume OS resources switching out of and back into the process.

15} No, we cannot substitute one set for the other without altering the meaning of the program. As the first program first checks the value of semaphore then updates it. While second program updates the value of semaphore then updates it. While second program updates the value of semaphore then checks it. This can give wrong output in terms of wait time.