Part 1

Task 1 :

- Changes Made : Changes were made inside the join() function of nachos.threads.KThread.
- Data structure : Queue
- Basic idea of implementation : A thread, say thread 1, calls join() inside another thread,say thread 2. If thread 1 is done executing, there is no need to join. Again if thread is already joined, the function simply returns. Otherwise, if the thread is not joined, the machine interrupt is disabled as threadQueue.waitForAccess needs the interrupt to be disabled. The current thread in the cpu is added to the joinQueue for regaining access and thread 1 is joined by putting its variable, isTrue's value true. Then calling sleep() blocks the current thread from running and runs the nextThread. After this, the interrupt is restored. In the function, finish(), after finishing the execution of thread 1, thread 2 will be ready to be run from the joinQueue.
- Testing implementation : Three threads were created and inside there is a loop which loops for five times. After forking all these threads, they were joined and they worked perfectly

Task 2 :

- Changes Made : Changes were made inside the sleep(), wake(), wakeAll() functions of nachos.threads.Condition2.
- Data structure : Queue
- Basic idea of implementation : Inside the sleep() function, the lock is released and the interrupt is disabled before putting the current thread to the waitQueue. Then the current queue is put to sleep, interrupt is restored and the lock is acquired. In the wake() function, first the interrupt is disabled and nextThread is taken from the waitQueue. If the thread is not null, it calls ready(). Then the interrupt is restored. In the wakeAll() function, it repeats the task of wake() until the waitQueue is empty.
- Testing implementation : Condition was checked inside the communicator

Task 3 :

- Changes Made : Changes were made inside the constructor function of Alarm class and timerInterrupt(), waitUntil() functions of nachos.threads.Alarm.
- Data structure : Queue
- Basic idea of implementation : The constructor function starts a thread which calls timerInterrupt(). timerInterrupt() function disables the interrupt, pops a

thread from the priorityQueue and makes it ready if the thread's waketime is less than or equal to the machine time, restores the interrupt. waitUntil() function first disables the interrupt, then saves the thread with its wakeTime in the waitQueue and then sends the thread to sleep and lastly restores the interrupt.

- Testing Implementation : After creating threads, we put them into sleep for a certain time to check **waituntil()**.

Task 4:

- Changes Made : Changes were made inside speak(), listen() functions of nachos.threads.Communicator.
- Basic idea of implementation : speak() function tries to acquire the lock so that no speaker or audience can enter. Then the speaker is forced to wait until an audience comes.If there is no active audience, the speaker is sent to sleep and tries to create a listener. speaker sets the data and wakes the next thread from waitQueue. Then the speaker goes to sleep releasing the lock so that the listener can access. If one speaker finds another speaker waiting, it wakes the waited speaker and the lock is released from the speaker. listen() function tries to acquire the lock so that no speakers or audience can enter. If another active audience is there, the listener is put to sleep and wakes a speaker. the listener goes to sleep. When a speaker wakes a listener, it gets the data which was set by the speaker and wakes the next thread from waitQueue. If a listener finds another listener in waiting, it wakes the waited listener and then releases the lock from the listener.
- Testing Implementation : We first create a communicator object on a certain lock obtained by condition. Then we created threads for speakers and listeners and pass this same communicator in these threads to communicate with each other.