* Slide 1 :
* Hello Sir, hello friends
* So we have chosen a programming project on Thread Synchronization and have named it as PoSync .
* Here are my team members :
* Prreti Krishnan
* Myself Sandhya
* Saranya Mohan
* Shruthi Narayanan
* Slide 2 :
* We have chosen a classic problem statement encompassing the coordination of activities between TA and students.
* In this we have dealt with 2 scenarios
* ->Single TA Multiple students
* ->Multiple TA multiple students
* The first scenario goes something like this
* 1)We have a TA who during working hours can attend to only one student at a time.
* 2)So facilitate this , there is a waiting room with five chairs.
* 3) As and when students come seeking the TA’s help, they wait in the waiting room for their turn.
* 4)The TA helps them one after the other and then the student leaves.
* 5)In case the waiting room is completely occupied ,the student leaves immediately .
* The second scenario is similar to the first one but with an extension that there are multiple TA’s
* Slide 3 :
* Before dwelling into the solution ,lets have a brief run through the purpose and concepts involved in the Project.
* The purpose of the project is
* 🡪to understand the concepts of multi-threading, thread synchronization. The issues of parallel or concurrent threads accessing shared data and how the integrity of this shared data should be maintained.
* 🡪We wanted Implement a solution critical section problem using POSIX Pthreads API
* Slide 4 :
* A brief idea of Process Synchronization :
* Synchronization involves the mechanism to ensure concurrently executing processes /threads does not result in the inconsistency of data.it must also avoid race conditions where-in two threads attempt to modify a shared variable at the same time i.e it must provide mutual exclusion.
* It guards busy waiting, avoids deadlocks and starvation.
* Slide 5 :
* Coming to the technicalities of a Semaphore,
* It is a integer Variable and basically has two standard operations : wait() & signal()
* Wait() routines waits on the semaphore value to be greater than 0 to acquire it and then decrement its value by 1.
* Signal() routine indicates release and increment of semaphore value by 1.
* Next we have the types of the semaphore- counting and binary
* Counting semaphore can range over an unrestricted domain where as binary can range only between 0 & 1 and is synonymous to mutex locks.
* Slide 6 :
* POSIX Pthread library are a standards based thread API for C/C++ and are implemented with the pthread.h header and the picture here depicts the basic routines available and their interconnection.
* Using pthread\_Create routine we generate threads that share the data from the master thread and performs concurrent operations, exit their routines and then joins back to the master thread.
* Slide 7 :
* Lets looks at the syntaxes of the pthread\_create and join routines
* **\*thread**
* An identifier for the new thread returned by the subroutine. When a thread is created it is written to the memory location to which this variable points. This identifier enables us to refer to the thread.
* **attr:** An attribute object that may be used to set thread attributes. NULL for default values.
* **start\_routine**: The routine that the thread will execute once it is created.
* **arg:**A single argument that may be passed to start\_routine. NULL may be used if no argument is to be passed.
* Now for join : One Thread can wait on the termination of another by using pthread\_join()
* The first parameter is the thread for which to wait, the identifier that pthread\_create filled in for us.
* The second argument is a pointer to a pointer that points to the return value from the thread. This function returns zero for success and an error code on failure
* Slide 8
* Lets us look at the semaphore routines :
* Sem\_init : It takes a reference to the semaphore ,a pshared integer value which is 1 for indicating the semaphore is being shared by processes and 0 for indicating it is being shared by threads and the third argument is a initial value given to the semaphore.
* Sem\_Wait🡪 it is as describes for wait
* Sem\_Post-> it is as described for signal.
* Slide 9 :
* Lets move to the solution approach for the TA-Student coordination :
* The blue paths indicates the student thread flow, the red path indicates TA thread flow
* Slide 10 : Following is the pseudo code for the Single TA Scenario
* Slide 12 : play out put and explain
* Slide 13 : explain the Multiple TA scenario briefly
* Slide 14 : Following is the pseudo code for the Multiple TA Scenario
* Slide 17 : play output video and explain
* Slide 18 : Summary of the solution
* Slide 19 : Thanks and Questions