# DESIGN OF FIRST PROGRAM

- In the first program ,we have created an array “fibo” as gobal and used this to store all the elements
- Function “malloc ” dynamically generates the required size of array
- In the main program ,we have generated first 5 fibo numbers .
- Next pthread_t workers[] creates worker threads and now in the for loop ,we have created each thread using “pthread_create” continuosly and this calls “runner” function.
- Here each time the function is called it generates a fibonocci number and stores it in the common array “fibo”
- Here we have passed the value of the array ,array[i] into the function which is the index of the array so using “temp” pointer and casting it into void.
- The main motto is here to take the index and generate the fibo number in the function and put it back in array using the index we got.
- Finally we have waited for completion of all the threads using “pthread_join” and joined it with main function so here by we get the required fibonocci numbers in the array.
- We have also included “<time.h>” to calculate the time taken for our program using clock() function and got it printed at the end.

# DESIGN OF SECOND PROGRAM

- In the second program ,we have created an array “fibo” as gobal and used this to store all the elements
- The main problem is to created a shared array to store numbers ,as child and parent gets different copies of resourses and are independent of each other
- Function “mmap” is used to create a virtual memory mapping to the created array which we required.This is typecasted into a integer pointer and given to our required “fibo” (array ) .
- In the main program ,we have generated first 5 fibo numbers .
- Next , in the for loop ,we have created a child process using fork() and it is directed to “function “ to generate a fibonocci number and then exit there itself .
- This way as the loop runs , each number is created and stored in array “fibo”

- Now the parent process goes down and waits for the termination of all the process using “wait(&status)” in the while loop until all the child process created in for loop exits.
- Here “status ” gives the return status of exited program and wait() is used to collect it.
- Finally we have waited for completion of all the child using “wait” and so here by we get the required fibonocci numbers in the array.
- We have also included “<time.h>” to calculate the time taken for our program using clock() function and got it printed at the end.
-

# GRAPH RESULTS

- As it is clear from the graph , the time taken for specific value using threads is greater than using process as per our theory .
- Threads can share gobal variables but whereas in multiprocess systems it is not.
- As the value increases the difference in time taken is huge and hence multithreading is preffered.