**Five** **philosophers problem**

1. **Solution in pseudocode:**

**We have 5 philosophers have three states THINKING,HUNGRY,EATING and two only can have the mutex and start eating all is occurred by multithreading process and three functions GRAP,RELEASE functions.**

**Grap function:**

**Grap function use synchronization and check for the free of left and right chopstick by while loop**

**While is not free do:**

**If one of right or left is eating thread sleep for random time from 100 to 2000 ms**

**If the two sides are free acquire the permit from the semaphore and let right and left chopsticks true**

**In the array by this algorithm**

***chopsticks[phnum] = true;***

***chopsticks[(phnum + 4) % 5] = true;***

**Phnum represents the philosopher number.**

**And also make ISEATING true in array in index of philosopher number.**

**After this NOTIFYALL and return true as it return boolean number.**

**Release function:**

**It use synchronization also it check if the philosopher is eating if it is eating true**

**let right and left chopsticks false**

**In the array by this algorithm**

***chopsticks[phnum] = false;***

***chopsticks[(phnum + 4) % 5] = false;***

**And also make ISEATING false in array in index of philosopher number.**

**After this NOTIFYALL.**

**Also we have EAT function to make the philosopher thread sleep for 8 seconds to prevent the starvation**

1. **Examples of Deadlock:**

**Deadlock is defined as a situation where set of processes are blocked because each process holding a resource and waiting to acquire a resource held by another process.**

**In our module, the five philosophers have only five chopsticks. Each one needs two chopsticks to eat so only two philosophers can eat at the same time. The deadlock in our problem took place if the five philosophers are hungry at the same time, as each philosopher checks the eating state of the two philosophers beside him to check the free state of the left and right chopsticks. So, if the two philosophers are eating and the condition didn’t acquire the philosopher will wait until the condition be ensued and the neighbored philosophers are not eating.**

**As shown in the program when the five philosophers are in hungry state, the program enters Deadlock case.**

1. **How did solve the Deadlock:**

**We solved the deadlock in our module by generating random number from 100 to 2000 to use it as sleep time for the thread, so when the philosopher check for the free of chopsticks in his left and right sides and it isn’t free, his thread sleep for randomized time where the threads with less sleep time will acquire the chopsticks and enter the eating state, and when it finishes eating it NOTIFYALL the threads to make them take place in checking and start to eat.**

**As shown in the program the deadlock case is solved as two philosophers start in eating state after all was hungry.**

1. **Examples of starvation:**

**Starvation is the problem that occurs when some processes keep executing and other processes get blocked for an indefinite time. In a heavily loaded computer system, a steady stream of processes can prevent other processes from ever getting the CPU.**

**In our module, the starvation state occurred as philosopher in eating state is this state for indefinite time. So, the waiting philosophers will wait for the mutex for an indefinite time.**

**As shown in the program, two philosophers are in an eating state while all other philosophers are waiting for unlimited time.**

1. **How did solve the starvation:**

**We solve the starvation in our module by giving a limited time for the eating state so the next philosopher can enter eating state after that limited time.**

**It is done by making the thread in the critical section sleep for a limited period of time so after this, the thread will release the mutex and make other threads acquire it and this process continues till the waiting queue of threads ends.**

**As shown in the program where starvation problem solved, and the waiting philosophers entered the eating state.**

**Explanation for real world application and how did apply the problem:**

**Cooperating processes that need to share limited resources. And represent it by multithreads thread for each philosopher.**

**Multithreading is the ability of a program or an operating system to enable more than one user at a time without requiring multiple copies of the program running on the computer. Multithreading can also handle multiple requests from the same user.**

**Each user request for a program or system service is tracked as a thread with a separate identity. As programs work on behalf of the initial thread request and are interrupted by other requests, the work status of the initial request is tracked until the work is completed. In this context, a user can also be another program.**

**Each CPU cycle executes a single thread that links to all other threads in its stream. This synchronization process occurs so quickly that it appears all the streams are executing at the same time. This can be described as a multithreaded program, as it can execute many threads while processing.**

**Each thread contains information about how it relates to the overall program. While in the asynchronous processing stream, some threads are executed while others wait for their turn. Multithreading requires programmers to pay careful attention to prevent race conditions and deadlock.**