DESIGN DOCUMENT

The design of the program is as follows:

The program has a main function, which is executed when the program starts.

In the main function, the program accepts following inputs from the user:

1. number of resources
2. total number of each type of resource
3. deadlock checking interval
4. which heuristic to use at the start of execution
5. time of execution (the program will run for 5*time of execution, where it will use different heuristic for different multiple of time of execution)
6. Number of threads

Then a mutex variable is initialised (this variable will be used to ensure that only one request happens at a time and when deadlock checks happen, there is no request happening). Start time is set, also deadlock interval is also set.

For each thread, two arrays are defined, request matrix and allocation matrix. First allocation matrix is initialised to 0. Then random values are set for request matrix, with range of random values limited to 1 to total resources for the particular resource type.

A struct pointer is used to pass values as arguments to the function called by pthread_create. Thread id, number of resources, request array, available array, allocation array, total resource array, deadlock interval and number of threads are passed.

Pthread create is used to create the worker thread, while calling thread function. In thread function, an infinite loop is created which sends requests for resource allocation using function request_resource_for_allocation. These requests are enclosed in mutex locks to ensure only one resource request happens at a particular time. After requesting for resources, each thread checks if it has obtained all the resources. If all requests of a particular thread are fulfilled, then, enclosed in mutex locks, the thread releases all its resources. Then it makes new request for resources.

Inside main, after worker threads are created, a structure variable is used to pass arguments to a new worker thread created by pthread_create. This new thread will do deadlock detection at every fixed interval. In deadlock_detection function, an infinite loop runs, which at detection intervals, runs deadlock detection algorithm.

The algorithm for deadlock detection used is as follows:

1. A marked array is created, with each entry either true or false. The entry is true, when all the values of allocation matrix for the particular thread is 0, else entry is false.
2. A temporary array W is created, which is initialised to be equal to available array.
3. Then a loop runs, which checks if for a thread_id, marked array has false value. If it is so, then it checks if all values of array W are greater than or equal to request array for that thread or not. If no such thread exists, then algorithm terminates.
4. If such a thread exists , then the thread is marked true and W is increased by the value of allocation array of that thread. Return to step 3.

At the end of the algorithm, if there are threads, for which marked array has false values, then those threads are in deadlock.

Then the deadlock function uses a particular heuristic, to resolve the deadlock situation. After each preemption, deadlock detection algorithm is again used to determine if deadlock has been resolved or not. If not then again a particular heuristic is used to preempt a thread and the deadlock detection algorithm is involked again.

Description of heuristics used are:

1. Heuristic 1: In heuristic 1, all deadlocked threads are preempted.
2. Heuristic 2: Threads are preempted on the basis of thread_id, that is the one with least thread_id is preempted.
3. Heuristic 3: The thread with largest remaining requests, (that is sum of values of request array) is preempted.
4. Heuristic 4: The thread with maximum allocated resources is preempted.
5. Heuristic 5: The thread with minimum allocated resources is preempted.

Finally, in main, there is an infinite loop, which keeps checking if time is greater than 5*time of execution or not. It it is so, it prints time between deadlocks for different heuristics and the one with largest and shortest one.