**Design Document**

1. **Design of the application**

The program essentially consists of a number of worker threads and a deadlock detection thread. The idea behind the simulation is to create a situation where the concurrent execution of the threads and their corresponding demands lead to a deadlock between the threads. The next task is to detect and consequently, resolve the deadlock by terminating threads in the deadlock, following some heuristics for deciding which thread to terminate first.

1. **The rationale for major decision choices**

**Ensuring mutual exclusion between threads:**

Mutual exclusion between concurrent threads trying to acquire resources from a shared set is essential to ensure synchronization amongst updates. For the purpose of ensuring mutual exclusion, a mutex lock has been used in the implementation. The lock is acquired by a worker thread during multiple instances. Firstly, the worker thread acquires the lock for generating the request set Rt and modifying the global *request* matrix. Next, the worker thread acquires the lock while requesting and acquiring a certain number of instances of a particular resource type. Now, if the number of requested instances of the particular resource is greater than the total number of available instances of the particular resource, then the thread is made to wait on a conditional variable *cond,* releasing the previously acquired mutex. If not, the requested instances of the specific resource are allocated to the thread, the updates are made and then the mutex is released.

Similarly, this mutex lock is also acquired before all the acquired resources are released by a thread. Once this is done, the thread releases the mutex.

Acquiring the mutex is also essential for the deadlock detection thread. In this case, the mutex is acquired at the start of the detection process and released at the end of it. This ensures synchronization.

Finally, the mutex also helps in the efficient implementation of the sig\_handler() function. Before the function starts executing its program termination code, it ensures that the mutex is available and acquires the same. This prevents the abrupt termination of the code(for instance, termination of the process, while the deadlock detection thread was in its middle of identifying and resolving deadlock).

**Explanation of heuristics:**

The following heuristics were used to determine the thread to be terminated, in order to resolve a detected deadlock:

1. Terminate thread with the maximum number of total resources allocated.

2. Terminate thread with maximum instances of any resource allocated.

3. Terminate thread with the minimum number of total resources allocated.

4. Terminate thread with minimum instances of any resource allocated.

5. Terminate thread in a linear order of deadlocked threads.

**Analysis of the heuristics:**

* **Commands used:**

**Heuristic 1:** ./a.out 8 3 1 900 42 4 A 10 B 8 C 9 D 7

**Heuristic 2:** ./a.out 8 3 2 900 42 4 A 10 B 8 C 9 D 7

**Heuristic 3:** ./a.out 8 3 3 900 42 4 A 10 B 8 C 9 D 7

**Heuristic 4:** ./a.out 8 3 4 900 42 4 A 10 B 8 C 9 D 7

**Heuristic 5:** ./a.out 8 3 5 900 42 4 A 10 B 8 C 9 D 7

* **Analyzing the time between successive deadlocks**

| **No. of sample run** | **Heuristic Type** | **Number of deadlocks** | **Average time between successive deadlocks(Td)** |
| --- | --- | --- | --- |
| Run 1 | Heuristic 1 | 213 | 4.211654 sec |
| Heuristic 2 | 200 | 4.470420 sec |
| Heuristic 3 | 169 | 5.308178 sec |
| Heuristic 4 | 185 | 4.849099 sec |
| Heuristic 5 | 185 | 4.832811 sec |
| Run 2 | Heuristic 1 | 221 | 4.045720 sec |
| Heuristic 2 | 198 | 4.530840 sec |
| Heuristic 3 | 161 | 5.571950 sec |
| Heuristic 4 | 196 | 4.561741 sec |
| Heuristic 5 | 178 | 5.040026 sec |
| Run 3 | Heuristic 1 | 217 | 4.134127 sec |
| Heuristic 2 | 204 | 4.397751 sec |
| Heuristic 3 | 165 | 5.418869 sec |
| Heuristic 4 | 178 | 5.040038 sec |
| Heuristic 5 | 181 | 4.956342 sec |

| **Heuristic Type** | **Average Td across the three runs** |
| --- | --- |
| Heuristic 1 | 4.1305003 sec |
| Heuristic 2 | 4.466337 sec |
| Heuristic 3 | 5.432999 sec |
| Heuristic 4 | 4.816959 sec |
| Heuristic 5 | 4.943059 sec |

**From the above analysis,**

* The **longest** average time between deadlocks is reported in the case of **Heuristic 3.**
* The **shortest** average time between deadlocks is reported in the case of **Heuristic 1.**