Dining Philosophers Problem - Sauhaard Batra

The dining philosophers problem is a special case of drinking philosophers problem, and it captures the essence of many synchronization problems. In the dining philosophers problem, a certain number of philosophers are sitting on a circular table. Every two adjacent philosophers has an edge associated with them, which basically denotes a fork. Every philosopher needs both the fork on his left and the fork on his right to eat. However, a philosopher can not eat while either the fork on his left or right is being used. Hence, any solution to this needs to guarantee the properties of uniqueness, fairness, symmetry, economy, concurrency and boundedness. To distinguish each philosopher from the other, replace the undirected edges with directed edges, where the direction of the edge goes from a philosopher from higher precedence to a philosopher with lower precedence.

For achieving fairness, after a certain buffer period, all edges directed towards a particular philosopher must get reversed to ensure that it is able to yield precedence to its neighbors. For maintaining acyclicity, all edges incident on a particular philosopher must get redirected towards it simultaneously.

However, a clause in this solution arises because its impossible to observe the behavior of all philosophers at the same time. We want to ensure acyclicity as well as fairness (in case of conflict of interest) by implementing the above solutions solely on the basis of the local state of the process. Hence, a distributed scheme is needed for implementing this.

Take two different possible scenarios for all the forks: dirty and clean. A philosopher will use only those forks which are clean, and once he has completed eating, the fork he has used will become dirty. A philosopher cleans a philosopher only while mailing it, and that is the only possible way for cleaning the fork. An eating philosopher does not entertain any cleaning requests. An additional constraint is put to ensure that the forks already clean do not get cleaned again…by making a non-eating philosopher satisfy the cleaning requests for only those forks which are dirty.

The direction of an edge between two neighbors u and v is from u to v (i.e., u has precedence over v) if and only if (1) u holds the fork shared by u and v, and the fork is clean, or (2) v holds the fork, and the fork is dirty, or (3) the fork is in transit from v to u. In addition to this, the direction (from u to v) of the edge can change only when u starts eating. Furthermore, all edges incident on an eating philosopher are directed toward it. For implementing the acyclicity rule: The direction of edges incident on a process may be changed only in the following way--all edges incident on a process may be simultaneously directed toward it. Initially all forks are dirty and are located at philosophers in such a way that the graph is acyclic.

Take 3 arrays of size k, namely fork, request (the terms request and request token have been used interchangeably), and dirty, where k is an integer that denotes the number of philosophers. Fork[u]==f represents that u’th philosopher has the f’th fork, request[u]==f represents that the u’th philosopher has the request token for the f’th fork and dirty[u]==f represents that the f’th fork is dirty for the u’th philosopher. Take one array which can take 4 possible values, thinking=1, hungry=2, eating=3, and done=4 that represent which philosophers are thinking, eating, hungry and completion of the process . Now, the initial conditions are that all the forks are dirty, and every fork and request token is held by a different philosopher, that is, one philosopher holds the fork and another has its token. Also, the forks (edges) have been oriented to ensure that the graph is acyclic. Now, there are a certain set of rules which need to referred to, for implementation:

Rule 1 is that if a philosopher wants to request the fork, it must be hungry, it must have the request token for the fork and it must not have the fork with itself. After it sends a request for the fork, the request is transferred to the current owner of the fork which is being requested.

Rule 2 states that if a philosopher is done with the fork, it can only release the fork if it is possessing it AND not eating it currently. It should have the request token for it. The fork getting released is dirty, and hence the philosopher cleans the fork before releasing it to the philosopher which requested the fork. If a philosopher wants to receive the request token for the fork it must mark the request as true. After receiving the fork the fork is marked as true and it is assumed that the fork is not dirty as the previous philosopher has cleaned the fork.

Taking an example, assume that there are 3 philosophers, namely p, q, r. Initially let us assume p has 2 forks, (1) that belong to p and q and (2) the belong to p and r. Both q and r have the request token for the forks. q also has a fork which belongs to q and r, and r has the request token for the same. So, initially in the precedence graph, the edge between p and q points at p, the one between p and r points at p, and the one between q and r points at q. All the forks initially are dirty. Now, first, q requests the fork from p (this is allowed as it has the request token for the fork but not the fork itself), and waits for the fork to be transferred from p to it. Now since the request token is transferred to p and p is not currently eating (he cannot eat from a dirty fork) and he owns the dirty fork which mutually belongs to q, p cleans the fork and passes it to q. A similar condition happens with the dirty fork in the ownership of p, which mutually belongs to both p and r, when r requests the fork. q still cannot eat as the other fork in its ownership (belonging to it and r) is dirty. So when r requests for the fork from q, q cleans the fork and passes it to r. Now r owns both the forks it needs to start eating, and it starts eating. After he finishes eating, both forks it owns mutually become dirty. Following this, both the edges directed towards it get reversed and get directed away from it. Now, q requests the fork from r (it had gotten the request token from r previously, when he was handing the fork to r), and r gives q the fork after cleaning. Now, q has both clean forks it needs for eating and starts eating, making them dirty, and directing the edges away from it. This is followed by fork requests from p, who acquires clean forks from both q and r, and starts eating, making both forks dirty again. The process starts again.

The solution now satisfies all possible constraints and is fair, as all philosophers (processes) get to eat with clean forks (get executed) completely at some point or the other.