Shreyansh Agrawal
-2020ucs0097

OPERATING SYSTEMS

Lab Assignment 4 Q1

Conflicts occur when two or more processes attempt to carry out specific actions at the same time. To resolve such a conflict, some processes must be treated differently than others. We must ensure that at least one process in each set of conflicting processes is distinguishable from the other processes in the set, which is known as a distinguishable property. Also, we must ensure fairness, which means that conflicts should not always be resolved to the detriment of a specific process, and outcomes should not always be the same in the same state.

A fork is either clean or dirty. After one philosopher eates, the fork becomes dirty. Only after the philosopher finishes eating, he cleans the fork when mailing. The philosophers are assumed to be the nodes of a precedence graph H. It is a directed graph in which the edge from node p to q signifies that the node p has higher precedence than q. A term depth is introduced as the number of predecessors of a node is known as the depth of that node. In case of conflict the process with lower precedence must yield to the process with higher precedence. After execution of a process, it gives higher precedence to all its neighbors. This is done by changing the direction of arrows in the graph. The depth of the process with highest precedence becomes zero every time. This way at each state, the depth of the process is helping us to distinguish out one process which has to be implemented and fairness is achieved as every process gets highest precedence. Also, at a particular state, all the edges are pointing towards a node or pointing away from it simultaneously. So there is no possibility of cycle formation, which implies that H will remain acyclic throughout the program execution.

Initially all forks are kept dirty and H is acyclic. That implies at least one of the philosophers has the current precedence and its depth is 0. The proposed solution introduces a request token. The person who is having the request token can only request for the fork. After the exchange of the fork, the process which had the fork earlier gets the request token now. Means that the philosopher can again request for the fork when he feels hungry. So initially, in every pair of neighboring philosophers, one is holding the request token and the other one is holding the fork.

We can create four boolean variables-

1. fork(f): To check if the current philosopher has the fork f;
2. request_token(f): to find if the current philosopher has the request token for fork f.
3. dirty(f): returns if fork f is dirty.
4. Hungry: returns if the current philosopher is hungry.

The philosopher can find himself in one of the four situations:

1. **Requesting the fork:-** While requesting the fork f, if hungry and request_token(f) is true and fork(f) is false, then the philosopher will send the request token to its neighboring philosopher with whom the fork f is shared. Then he turns the request_token false.
2. **Receiving the token:-** If a philosopher gets a request token he turns request_token(f) true.
3. **Releasing the fork:-** If the philosopher has finished eating, he will check if he has received request token for fork f i.e. request_token(f). Now if dirty(f) is also true, he will send over the fork to the philosopher who has sent him the request token. Now he turns dirty(f) false (as he cleans the fork before transmitting it) and fork(f) also false (as he no longer has access to fork f).

4. **Receiving the fork:-** The philosopher gains access for fork f so he turns fork(f) true. When he finishes eating, he turns dirty(f) true.

Let's take three philosophers p, q and r sitting at the nodes of a graph. As shown below in **Fig1**

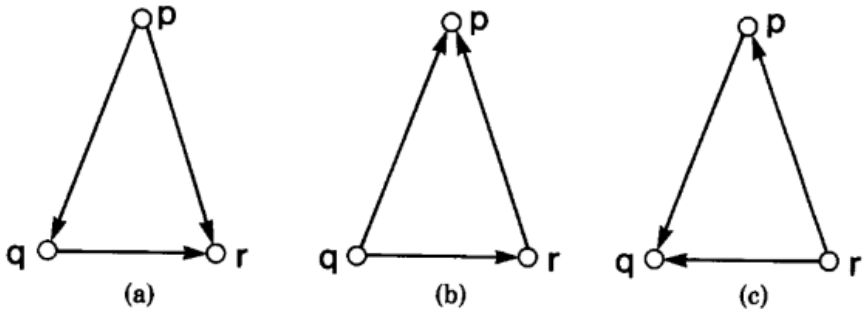

Fig1

Here the philosopher p has highest precedence at initial state. It sends request tokens to philosophers q and r who are having dirty forks. Current depths of p,q and r are 0, 1 and 2 respectively. The processes q and r receives the request tokens and transfers one clean fork each to p. Now p can use the forks to eat. While p is eating, if the context switch happens, the process q and r are still waiting for p to finish eating to get the fork they share with p, so nothing

will happen until p finishes eating. Now, when p finishes eating, it has two dirty forks. We can redirect the edges towards p now (Fig1(b)). Now the depth of p, q and r are 2, 0, 1 respectively. As q gets the highest precedence, it requests one fork each from its neighbors i.e. p and r. Same mechanism will repeat and q will get clean forks which he uses to eat. When q finishes, it has two dirty forks which are shared with p and r respectively. P and r have request tokens and q has two dirty forks. We will again change the edge directions associated with q (Fig1(c)). Now the depth of p, q and r are 1, 2, 0 respectively. Now r can start eating.

Here, all the 3 philosophers get a chance to eat. The depth of the philosopher node in the precedence graph helps to distinguish it from the other processes and it is executed. Since each node will be at highest precedence at least once, every philosopher will get to eat and none of them will remain empty. In case with more than 3 philosophers, the philosophers at the same depth will be eating simultaneously. This way there will be multiple philosophers eating at the same time which ensures concurrency of processes. A philosopher will receive a request token from all its neighbors, so it will receive a finite number of messages. Moreover, if a philosopher gets one clean fork but the other one is dirty, he will wait with one fork in hand for the other philosopher to finish eating. As the processes finish in finite time, infinite postponement for a philosopher will not occur.
