Operating Systems Laboratory (Gx) (2017-18)

Implement of a shared linked list to be accessed by concurrent processes

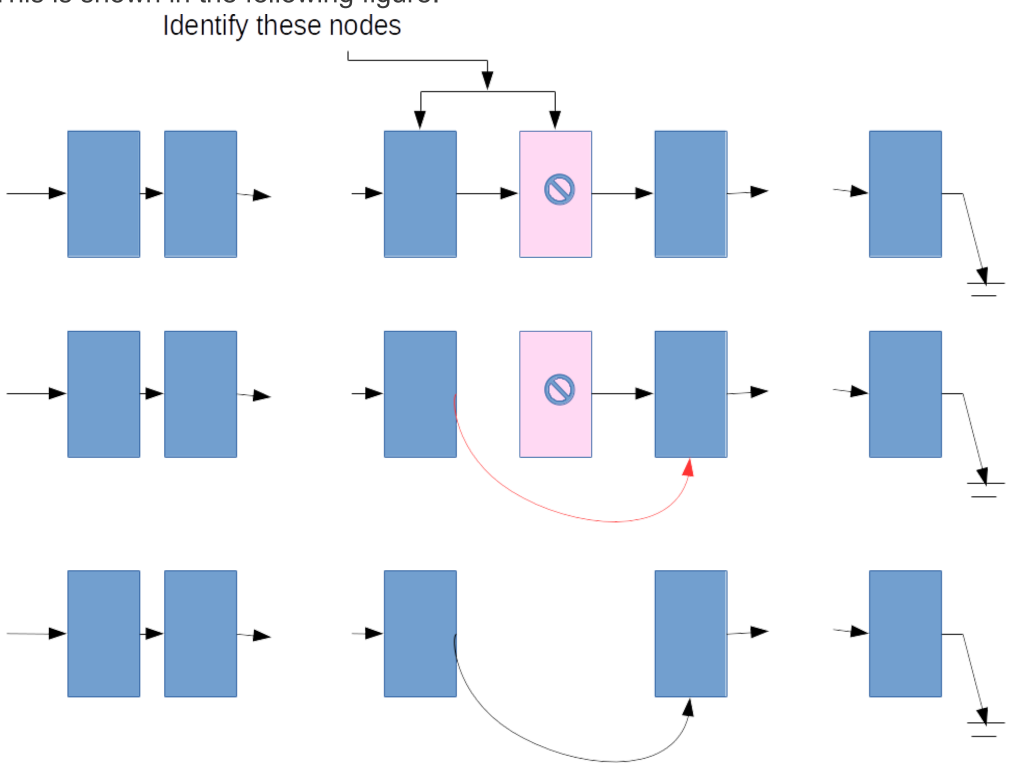
We are aware of ***linked list*** data structure which we have been implementing and using since our second semester of our undergraduate programme. Now we will attempt to build linked lists which can be used by concurrent processes. We shall try to define functions which can be called from within a program for creation of a linked list, adding node(s) to it, deleting its node(s), searching for a particular node in the linked list, etc.

Subsequently we shall package these functions as a library. That is, we shall provide a header file (say, sharedll.h) and a library file (say, libsharedll.so and/or libsharedll.a) that can be used by other programmers.

Moreover, at the beginning, assume that every node (**struct** in C) of the linked list has only one integer data field (obviously other than the fields we need for implementing the linked list). Subsequently we shall attempt to provide support for generic linked list, that is, where arbitrary data can be kept in a node.

1. Multiple linked lists have to be supported.
2. Try to identify the functions (Interface, API,...) that we are to implement along with the global variables, if any, we need.
3. Identify the Data Structures needed - for individual linked lists as well as common (to be used for all linked lists) ones.
4. Since multiple independent processes will be using a particular shared linked list, there must be some mechanism for getting the handle of an existing shared linked list (like semget() or shmget() the id for one existing.
5. Since multiple processes may attempt to modify (add a node, delete a node, etc.) a linked list, **race condition** may occur. Judicious identification and handling of critical section is of utmost importance. For example, deletion of a node involves the following steps.
    1. Find out the ***previous node*** of the node to be deleted. This does not any involve modification but takes most of the time.
    2. Modify the ***previous node*** identified in the previous step so that it points to the next to next node.
    3. Destroy the node to be deleted.

This is shown in the following figure.



Please note that treating the all 3 steps together as a critical section will unnecessarily block all other deserving processes

6. Propose an efficient scheme for handling critical sections so that race condition can be avoided, there is no chances for deadlock and waiting is ***minimalized.***
7. ***Try to comprehend and document your scheme first and then start coding.***

Please ensure that your program(s) are well-documented and properly indented.

Submission status

Attempt number	This is attempt 1.
Submission status	Submitted for grading
Grading status	Not graded
Due date	Tuesday, 6 February 2018, 4:35 PM
Time remaining	Assignment was submitted 11 secs early
Last modified	Tuesday, 6 February 2018, 4:34 PM