

L04c. Communication

Counting Barrier:

- We have a counter that is initialized by the number of threads N to be synchronized.
- Once a thread arrives to the barrier, it atomically decrements the counter, and spins on it till it becomes zero.
- The last thread to arrive will decrement the counter. Now the counter is zero, so that thread will reset the counter to its initial value N , so that it can be used for the next barrier.
- A problem can arise if one of the threads races to the next barrier before the counter is reset to its initial value N . This thread will directly go through the barrier.
 - To solve this problem, we add another spin loop so that the threads will wait for the counter to become zero, then wait again till it becomes N .

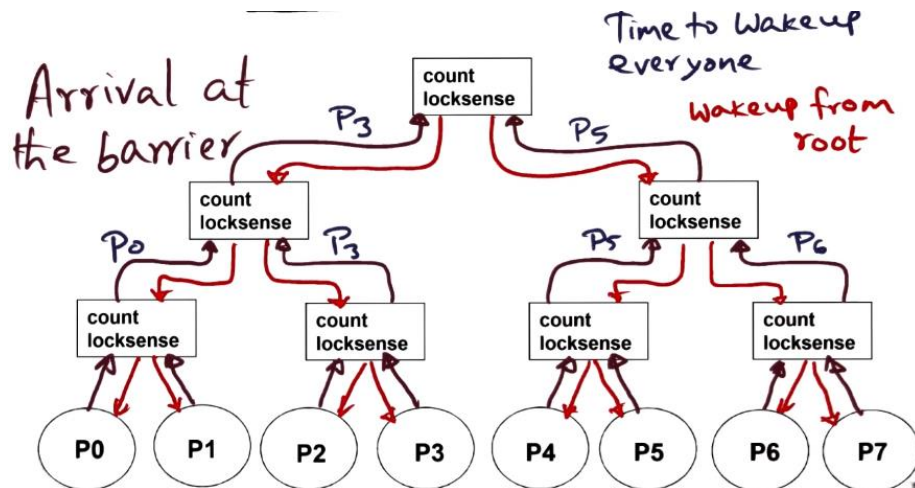


Sense Reversing Barrier:

- In a Counting Barrier we have two spin loops for each thread. One waits for the counter to become zero, and the other one waits for the counter to become N .
- To avoid having two loops, we add another variable "sense" that is shared between all the threads to be synchronized.
- This variable will be true for the current barrier, and false for all the other barriers.
- This way we can determine which barrier we're in at any particular point of time.
- Whenever a thread arrives at the barrier, it decrements the counter and spins on "sense" reversal.
- The last thread will reset the counter to N and reverses the "sense" variable.
- The problem with this technique is that we have two shared variables between all the threads, which decreases the possibilities for scalability (More sharing → Less scalability).

Tree Barrier:

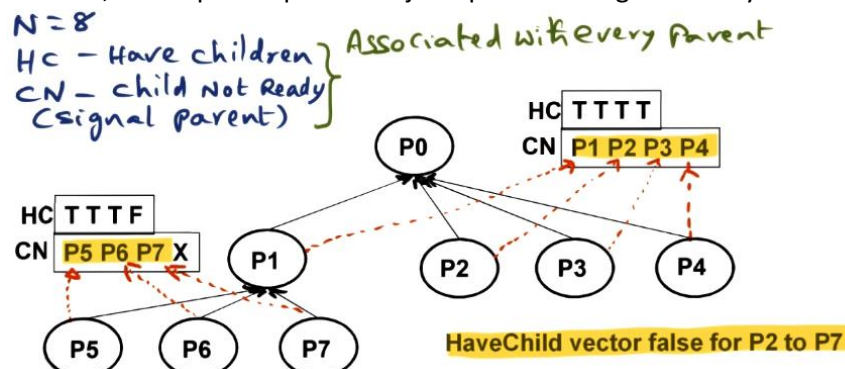
- Instead of having a single pair of "count" and "lock_sense" variables shared between N threads, we will have a pair of variables for each K number of threads, where $K < N$.
- When a thread arrives to the barrier, it decrements the counter and spins on lock_sense.
- Once a thread decrements the counter to zero, it checks if it has a "parent", if yes it decrements the counter of the its parent and spins on the parent's lock_sense.
- This operation is executed recursively until the last thread arrives to the barrier.
- The last thread will decrement the counter at the root of the tree to zero, which indicates that everyone arrived. This thread will then reset the counter to N and reverse the root's lock_sense flag.
- All the threads till the bottom of the tree will wake up recursively.



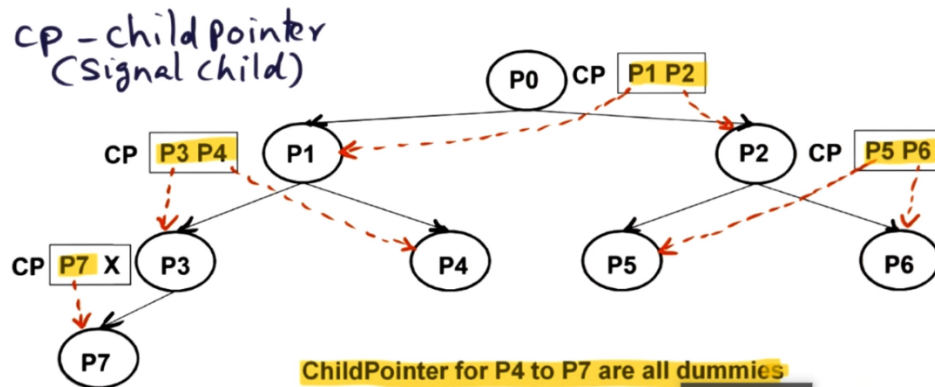
- This allows for higher scalability since the amount of sharing is relatively small.
- Disadvantages:
 - The spin location is not static for each thread. It depends on the time the thread arrives to the barrier.
 - If you have a huge number of threads, the tree will grow too deep and will result in contention on the system.
 - If the system is non-cache coherent, if the memory is distributed, a thread might need to modify/spin on a variable in a remote memory, which adds contention to the system.

MCS Tree Barrier (4-Ary arrival):

- This is a modified Tree Barrier where each node is allowed to have at most 4 children.
- Each node has two data structure:
 - HaveChildren: Indicates if the node is a parent or not, and how many children it has.
 - ChildNotReady: Each child thread has a unique spot in the parent's ChildNotReady structure to signal the arrival of that child thread. These spots are statically determined.
- Whenever a thread arrives to the barrier, it sets its spot in the parent's ChildNotReady structure to True. The thread spins on the arrival of its own children if it has any.
- A 4-Ary tree facilitates the best performance.
- In a cache-coherent system, it can be arranged so that all the ChildNotReady structure can be packed in one word, so the parent processor just spins on a single memory location.

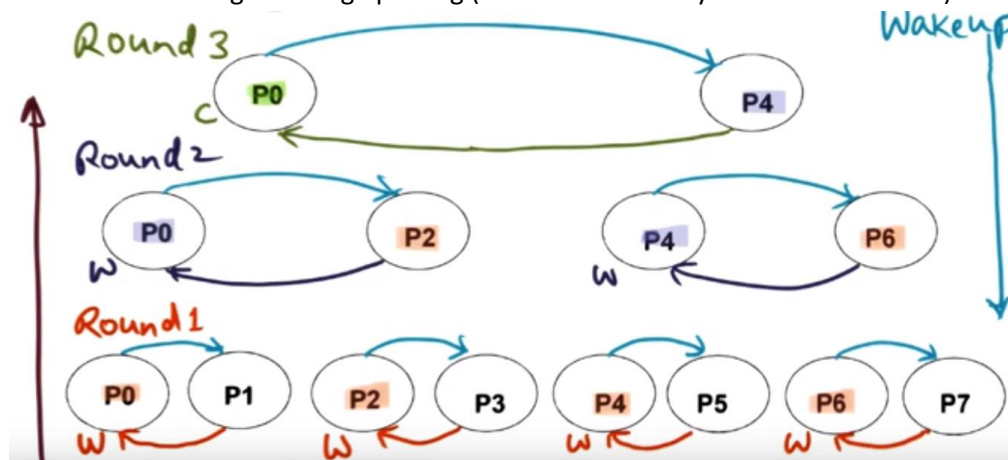


- Although the arrival tree is 4-Ary as shown in the above figure, the wakeup tree is binary.
- Each thread has a ChildPointer that can be used to signal the child to wake up.



Tournament Barrier:

- The barrier is represented by a tournament data structure with N players and $\log_2 N$ rounds.
- The winning thread of each round is predetermined. This allows the spin locations to be static.
- Tournament Barriers don't need a Fetch-and- ϕ operation.
- Tournament Barriers works even if the system doesn't have a shared memory, where threads can only communicate through message passing (No shared memory \rightarrow Cluster machine).



Dissemination Barrier:

- In each round k , whenever a processor P_i arrives at the barrier, it will signal processor $P_{(i+2^k).mod(N)}$.
- This creates a cyclic communication order between the processors/threads.
- A Dissemination Barrier would need $\lceil \log_2 N \rceil$ rounds for all the processors to wake up.
- Spin locations are statically determined.

