Sarah Whelan
slw96
10/8/2015
Assignment 3

Concurrent Programming with Monitors

1. Priority-based Searchers/Inserters/Deleters Problem without starvation: Monitor-based solution.

Alternatives:
pi issues x.wait and pj issues x.signal
1. pj waits until pi leaves monitor (or blocks at a wait statement).
2. pi waits until pj leaves monitor (or blocks at a wait statement).
3. pj exits monitor immediately; pi resumes.

I am using alternative 2 such that the signaled waits until the signaler leaves the monitor (or blocks at a wait statement).

```
type SearchInsertDelete = monitor;

var inserting:boolean;
var deleting:boolean;
var sPassingCount:int;

var sPassedCount:int;
var sWaitCount:int;
var iWaitCount:int;
var dWaitCount:int;

var sBlocked:boolean;
var iBlocked:boolean;

var search:condition;
var insert:condition;
var delete:condition;
var starvation:condition;

process entry SearcherEnter(){
    if(sBlocked || deleting || (sPassingCount + sPassedCount) == 10){
        sWaitCount++;
        search.wait;
        sWaitCount--;
    }
    sPassingCount++;
}

process entry SearcherExit(){
    sPassingCount--;
    sPassedCount++;

    // Should we go into starvation mode?
    if((sPassedCount + sPasssingCount) == 10){
        sBlocked = true;
        // If there are other searchers still passing fall out and let the last one deal
        if(sPassingCount <= 0){
            sPassedCount = 0;
            if(inserting){
                // If there is an inserter still inserting wait it out
                insertingBeforeStarvation = true;
                starvation.wait;
            }
            if(iWaitCount > 0){
                iStarvationCount = iWaitCount;
                insert.signal;
            } else if(dWaitCount > 0){
                iBlocked = true;
                dStarvationCount = dWaitCount;
                delete.signal;
            } else {
                // Nothing was waiting
                sBlocked = false;
                if(sWaitCount > 0){
                    search.signal;
                }
            }
        }
    } else {
        // Normal Operation
        if(iWaitCount <=0 && sWaitCount <= 0 && !inserting && sPassingCount <= 0){
            delete.signal;
        } else {
            search.signal;
            if(!inserting){
                insert.signal;
            }
        }
    }

}

process entry InserterEntry(){
    if(iBlocked || inserting || deleting){
        iWaitCount++;
        insert.wait;
        iWaitCount--;
    }
    inserting = true;
}

process entry InserterExit(){
    inserting = false;
    if(insertingBeforeStarvation){
        // if this was the inserter inserting while attempting to enter starvation mode start starvation mode
        starvation.signal
        // and leave
    } else {
        if(sBlocked){
            // Starvation Mode
            iStarvationCount--;
            if(iStarvationCount == 0){
                iBlocked = true;

                if(dWaitCount > 0){
                    // This assumes that we are to allow the deleters that came to wait during the inserters, during starvation mode,
                    // also get to go during the starvation service (ie not just the deleters that were waiting when the searchers reached 10).
                    // If we only wanted the deleters that were waiting at the 10 searcher count we'd put this line before the 10th searcher signaled insert
                    dStarvationCount = dWaitCount;
                    delete.signal;
                } else {
                    // End starvation mode
                    sBlocked = false;
                    iBlocked = false;
                    search.signal;
                    insert.signal;
                }
            } else {
                // Not finished with waiting inserters
                insert.signal;
            }
        } else {
            // Normal Operation
            if(iWaitCount > 0 || sWaitCount > 0){
                search.signal;
                insert.signal;
            } else if(sPassingCount <= 0 && sWaitCount <= 0){
                delete.signal;
            }
        }
    }
}

process entry DeleterEntry(){
    if(sPassingCount > 0 || inserting){
        dWaitCount++;
        delete.wait;
        dWaitCount--;
    }
    deleting = true;
}

process entry DeleterExit(){
    deleting = false;
    // if We are in starvation mode
    if(sBlocked && iBlocked){
        dStarvationCount--;
        if(dStarvationCount > 0){
            delete.signal;
        } else {
            // No more deleters for starvation mode end starvation mode
            sBlocked = false;
            iBlocked = false;
            if(sWaitCount > 0 || iWaitCount > 0){
                search.signal;
                insert.signal;
            } else {
                delete.signal;
            }
        }
    } else {
        // Normal Mode
        if(sWaitCount > 0 || iWaitCount > 0){
            search.signal;
            insert.signal;
        } else {
            delete.signal;
        }
    }
}

// Initialize variables
begin
sWaitCount = 0;
iWaitCount = 0;
dWaitCount = 0;
inserting = false;
deleting = false;
sPassingCount = 0;
sPassedCount = 0;
sBlocked = false;
iBlocked = false;
end

// The processes using the monitor as defined above:

// Declare and initialize the shared monitor to be used by each forked process
var monitor:SearchInsertDelete;

process searcher(L, item){
    monitor.SearcherEntry();
    SEARCH-AND-LOG-RESULTS(L, item);
    monitor.SearcherExit();
}

process inserter(L, item){
    monitor.InserterEntry();
    INSERT-AND-LOG-RESULTS(L, item);
    monitor.InserterExit();
}

process deleter(L, item){
    monitor.DeleterEntry();
    DELETE-AND-LOG-RESULTS(L, item);
    monitor.DeleterExit();
}
```

2. Four-of-a-Kind Problem

Alternatives:
pi issues x.wait and pj issues x.signal
1. pj waits until pi leaves monitor (or blocks at a wait statement).
2. pi waits until pj leaves monitor (or blocks at a wait statement).
3. pj exits monitor immediately; pi resumes.

I am using alternative 2 such that the signaled waits until the signaler leaves the monitor (or blocks at a wait statement).

```
type FourOfAKind = monitor;

var turnId:int;
var gameWon:boolean;

// A deck of 24 cards split into 6 different kinds 4 cards of each kind
enumerated card: {
    1a, 1b, 1c, 1d,
    2a, 2b, 2c, 2d,
    3a, 3b, 3c, 3d,
    4a, 4b, 4c, 4d,
    5a, 5b, 5c, 5d,
    6a, 6b, 6c, 6d
}

// An array of arrays of cards for the player's hands
card[][] hands;

// An array of arrays of cards for the discard/pickup piles
card[][] piles;

var turn:condition;

process entry boolean play(i){
    if(!gameWon){
        if(turnId != i){
            turn.wait;
        }
        if(turnId == i){
            if(FOUR-OF-A-KIND(hands[i])){
                PRINT("I player " + i + "win!");
                gameWon = true;
            } else {
                DISCARD-TO-PILE(piles[i], hands[i]);
                PICK-UP-CARD-FROM-PILE(piles[(i+1)mod4], hands[i]);
                if(FOUR-OF-A-KIND(hands[i])){
                    PRINT("I player " + i + "win!");
                    gameWon = true;
                } else {
                    turn = (turn + 1) mod 4;
                }
            }
        }
        turn.signal;
        return !gameWon;
    }
    return false; //don't keep looping
}

process boolean FOUR-OF-A-KIND(card[] hand){
    // returns if hand (an array of cards of length four) is a four of a kind
}

process DISCARD-TO-PILE(card[] pile, card[] hand){
    // Takes a card from hand and adds it to pile must handle the re-sizing of the array
    // All these are pointers and if the pile here is changed the shared variable (piles) is also changed
}

process PICK-UP-CARD-FROM-PILE(card[] pile, card[] hand){
    // Takes a card from pile and adds it to hand hand should have 4 spots so no resizing there but may want to make the pile array smaller
    // All these are pointers and if the pile here is changed the shared variable (piles) is also changed
}

process entry DEAL(){
    // Deal out the cards to the hands and piles
    // Does not use any condition variables but does use hands and piles which are monitor variables so this goes here.
}

// Initialize variables
begin
turnId = 0;
gameWon = false;
// Initialize the array to have four arrays of length four of cards
hands = new card[4][4];
// Initialize the array to have four arrays of length two of cards
piles = new card[4][2];
end

// The shared monitor for the player processes
var monitor:FourOfAKind;

// The parent "game" process must "deal" the cards and give values of cards to the hands and piles arrays
monitor.DEAL();

// Then the parent process will fork four children that will use the following process to play the game

```

```
process player(i){
    while(monitor.play(i));
}
```