```
Sarah Whelan
slw96
October 1, 2015
EECS 338
Assignment 2

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

nonbinary semaphore mutex = 1;
nonbinary semaphore sWait = 0;
nonbinary semaphore iWait = 0;
nonbinary semaphore dWait = 0;

int sPassingCount = 0;
int sStarvationCount = 0;

int sWaitCount = 0;
int iWaitCount = 0;
int dWaitCount = 0;

int iStarvationCount = 0;
int dStarvationCount = 0;

boolean sBlocked = false;
boolean iBlocked = false;

boolean insertingDuringStarveFlag = false;
boolean inserting = false;
boolean deleting = false;

enum {SEARCHER, INSERTER, DELETER} PROCCESS_TYPE;

process searcher(L, item){
    wait(mutex);

    // Handle entering starvation mode if a new searcher tries to search
    if(sStarvationCount == 10){
        if(sPassingCount == 0){
            if((iWaitCount > 0) || (dWaitCount > 0)){
                sBlocked = true;
                if(iWaitCount > 0){
                    iStarvationCount = iWaitCount;
                    if(inserting){
                        // If an inserter is inserting when starvation mode is entered
                        // let it finish inserting and don't call signal on iWait
                        insertingDuringStarveFlag = true;
                    } else {
                        sStarvationCount = 0;
                        signal(iWait);
                    }
                } else {
                    iBlocked = true;
                    sStarvationCount = 0;
                    dStartvationCount = dWaitCount;
```

```
                    signal(dWait);
                }
                sWaitCount++;
                // Don't call signal on mutex as an inserter or a deleter was called
                that will signal mutex
                wait(sWait);
                sWaitCount--;
            } else {
                // Starvation mode not entered as there were no waiting inserters or
                deleters
                sStarvationCount = 0;
                if(sWaitCount > 0){
                    sWaitCount++;
                    signal(mutex);
                    wait(sWait);
                    sWaitCount--;
                }
            }
        } else {
            sWaitCount++;
            signal(mutex);
            wait(sWait);
            sWaitCount--;
        }
    // In starvation mode or a deleter is deleting block more searchers
    } else if(sBlocked || deleting){
        sWaitCount++;
        signal(mutex);
        wait(sWait);
        sWaitCount--;
    }
    sPassingCount++;
    sStarvationCount++;
    signal(mutex);

    LOG-START-TIME(PROCCESS_TYPE.SEARCHER, item);
    SEARCH-AND-LOG-RESULTS(L, item);

    wait(mutex);
    sPassingCount--;

    // Handle entering starvation mode
    if(sStarvationCount == 10 && sPassingCount == 0){
        if((iWaitCount > 0) || (dWaitCount > 0)){
            sBlocked = true;
            if(iWaitCount > 0){
                iStarvationCount = iWaitCount;
                sStarvationCount == 0;
                if(inserting){
                    insertingDuringStarveFlag = true;
                } else {
                    signal(iWait);
                }
            } else {
```

```
                iBlocked = true;
                sStarvationCount == 0;
                dStarvationCount = dWaitCount;
                signal(dWait);
            }
        } else {
            // Starvation mode not entered as there are no waiting inserters or deleters
            sStarvationCount == 0;
            if(sWaitCount > 0 && !sBlocked){
                signal(sWait);
            } else if(!inserting && iWaitCount > 0 && !deleting){
                signal(iWait);
            } else if(!deleting && dWaitCount > 0 && !inserting && sPassingCount < 0){
                signal(dWait);
            } else {
                signal(mutex);
            }
        }
    } else {
        if(sWaitCount > 0 && !sBlocked){
            signal(sWait);
        } else if(!inserting && iWaitCount > 0 && !deleting){
            signal(iWait);
        } else if(!deleting && dWaitCount > 0 && !inserting && sPassingCount < 0){
            signal(dWait);
        } else {
            signal(mutex);
        }
    }
}

process inserter(L, item){
    wait(mutex);
    if(inserting || deleting || iBlocked){
        iWaitCount++;
        signal(mutex);
        wait(iWait);
        iWaitCount--;
    }
    inserting = true;
    signal(mutex);

    LOG-START-TIME(PROCCESS_TYPE.INSERTER, item);
    INSERT-AND-LOG-RESULTS(L, item);

    wait(mutex);
    inserting = false;
    if(sBlocked && !iBlocked){
        if(insertingDuringStarveFlag){
            // If this inserter was the one that was inserting when starvation mode
            // was set don't decrement starvation count and let the waiting inserters go
            insertingDuringStarveFlag = false;
            if(iWaitCount > 0){
                signal(iWait);
```

```
161                 }
162             } else if(iStarvationCount > 0){
163                 iStarvationCount--;
164                 if(iStarvatiionCount == 0){
165                     iBlocked = true;
166                     if(dWaitCount > 0){
167                         dStarvationCount = dWaitCount;
168                         signal(dWait);
169                     }
170                 } else {
171                     signal(iWait);
172                 }
173             }
174         } else if(sWaitCount > 0 && !deleting){
175             signal(sWait);
176         } else if(iWaitCount > 0 && !inserting && !deleting && !iBlocked){
177             signal(iWait);
178         } else if(dWaitCount > 0 && !inserting && !deleting && sPassingCount <= 0){
179             signal(dWait);
180         } else {
181             signal(mutex);
182         }
183     }
184
185     process deleter(L, item){
186         wait(mutex);
187         if(inserting || deleting || sPassingCount > 0){
188             dWaitCount++;
189             signal(mutex);
190             wait(dWait);
191             dWaitCount--;
192         }
193         deleting = true;
194         signal(mutex);
195
196         LOG-START-TIME(PROCCESS_TYPE.DELETER, item);
197         DELETE-AND-LOG-RESULTS(L, item);
198
199         wait(mutex);
200         deleting = false;
201         if(sBlocked && iBlocked && dStarvationCount > 0){
202             dStarvationCount--;
203             if(dStarvationCount == 0){
204                 sBlocked = false;
205                 iBloccked = false;
206                 if(sWaitCount <= 0 && iWaitCount <= 0){
207                     if(dWaitCount > 0){
208                         signal(dWait);
209                     } else {
210                         signal(mutex);
211                     }
212                 } else if(sWaitCount > 0){
213                     signal(sWait);
214                 } else if(iWaitCount > 0){
```

```
                    signal(iWait);
                }
            } else {
                signal(dWait);
            }
        } else {
            if(sWaitCount > 0){
                signal(sWait);
            } else if(iWaitCount > 0 && !inserting){
                signal(iWait);
            } else if(dWaitCount > 0){
                signal(dWait);
            } else {
                signal(mutex);
            }
        }
    }

    public void LOG-START-TIME(PROCCESS_TYPE, item){
        //Logs that the process' PROCCESS_TYPE, pid, and item along with the start time
    }

    public void SEARCH-AND-LOG-RESULTS(L, item){
        // Searches through the linked list L for item
        // if found log success
        // if !found log failure
    }

    public void INSERT-AND-LOG-RESULTS(L, item){
        // Attempts to insert the item into the linked list L
        // if item doesn't already exist insert and log success
        // if item already exists don't insert and log failure
    }

    public DELETE-AND-LOG-RESULTS(L, item){
        // Attempts to delete item from linked list L
        // if item already exists delete and log success
        // if item doesn't already exist don't delete and log failure
    }

    2. Four-of-a-Kind Problem

    // Indexed from zero indicating it is p0's turn first
    boolean[] turn = {true, false, false, false};
    boolean gameWon = false;
    nonbinary semaphore mutex = 1;
    process player(i, L){
        // access to gameWon can violate mutual exclusion as the
        // race condition to restart the loop does not lead to livelocks
        while(!gameWon){
            wait(mutex);
            if(turn[i]){
                if(HAS-FOUR-OF-KIND(L)){
                    gameWon = true;
```

```
269                 } else {
270                     DISCARD-CARD(i, L);
271                     PICK-UP-CARD((i+1)mod(4), L);
272                     if(HAS-FOUR-OF-KIND(L)){
273                         gameWon = true;
274                     } else {
275                         turn[i] = false;
276                         turn[(i+1)mod(4)] = true;
277                     }
278                 }
279             }
280             signal(mutex);
281         }
282         return;
283     }
284
285     public boolean HAS-FOUR-OF-KIND(L){
286         // Returns if list of cards L is four of a kind
287     }
288
289     public void DISCARD-CARD(i, L){
290         // Chooses a card from L to discard and places it on the ith pile
291     }
292
293     public void PICK-UP-A-CARD(i, L){
294         // Picks up a card from the ith pile and places it into L
295     }
```