```
// bufpool.c ... buffer pool implementation
// Implements a clock-sweep replacement strategy

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "bufpool.h"

#define MAXID 4
#define NONE  (-1)
#define MAX_USAGE 3

// one buffer
struct buffer {
    char  id[MAXID];    // which page it holds e.g. R02
    int   pin;          // how many "processes" are using it
    int   dirty;        // has it been modified?
    int   pop;          // "popularity" measure, for clock-sweep
    char  *data;        // actual data held in buffer
};

// collection of buffers + stats
struct bufPool {
    int   nbufs;         // how many buffers
    int   nrequests;     // stats counters
    int   nreleases;
    int   nhits;
    int   nreads;
    int   nwrites;
    int   clock;         // clock hand
    struct buffer *bufs; // actual buffers
};

// Helper Functions (private)

// pageInBuf(pool,slot)
// - return the name of the page current stored in specified slot

static
char *pageInBuf(BufPool pool, int slot)
{
    char *pname;
    pname = pool->bufs[slot].id;
    if (pname[0] == '\0')
        return "_";
    else
        return pname;
}

// pageInPool(BufPool pool, char rel, int page)
```

```
// - check whether page from rel is already in the pool
// - returns the slot containing this page, else returns NONE1

static
int pageInPool(BufPool pool, char rel, int page)
{
    int i;  char id[MAXID];
    sprintf(id,"%c%02d", rel, page);
    for (i = 0; i < pool->nbufs; i++) {
        if (strcmp(id,pool->bufs[i].id) == 0) {
            return i;
        }
    }
    return NONE;
}

// findFree(pool)
// - use the first available free buffer

static
int findFree(BufPool pool)
{
    int i;
    int slot = NONE;
    for (i = 0; i < pool->nbufs; i++) {
        // free buffers have empty id
        if (pool->bufs[i].id[0] == '\0') {
            slot = i;
            break;
        }
    }
    return slot;
}

// findVictim(pool)
// - finds "best" buffer pool slot to replace
// - "best" is determined by the clock sweep
// - if the replaced page is dirty, write it out
// - initialise the chosen slot to hold the new page
// - if there are no available slots, return NONE

static
int findVictim(BufPool pool)
{
    int slot = NONE;

    // TODO ... complete this function
    // Should take around 10-15 lines of code
    int attempts = 0;
    // keep going until we scan the entire pool without
    // doing anything
    while (attempts < pool->nbufs) {
```

```
        struct buffer *buf = &pool->bufs[pool->clock];

        if (buf->pin == 0 && buf->pop == 0) {
            slot = pool->clock;
            break;
        } else if (buf->pop > 0) {
            buf->pop--;
            attempts = 0;
        } else {
            attempts++;
        }

        pool->clock = (pool->clock + 1) % pool->nbufs;
    }

    if (slot == NONE) return NONE;
    if (pool->bufs[slot].dirty) pool->nwrites++;
    pool->clock = (pool->clock + 1) % pool->nbufs;

    return slot;
}


// Interface Functions


// initBufPool(nbufs)
// - initialise a buffer pool with nbufs
// - buffer pool uses supplied replacement strategy

BufPool initBufPool(int nbufs)
{
    BufPool newPool;

    newPool = malloc(sizeof(struct bufPool));
    assert(newPool != NULL);
    newPool->nbufs = nbufs;
    newPool->nrequests = 0;
    newPool->nreleases = 0;
    newPool->nhits = 0;
    newPool->nreads = 0;
    newPool->nwrites = 0;
    newPool->clock = 0;
    newPool->bufs = malloc(nbufs * sizeof(struct buffer));
    assert(newPool->bufs != NULL);

    int i;
    for (i = 0; i < nbufs; i++) {
        newPool->bufs[i].id[0] = '\0';
        newPool->bufs[i].pin = 0;
        newPool->bufs[i].dirty = 0;
        newPool->bufs[i].pop = 0;
```

```
    }
    return newPool;
}

int request_page(BufPool pool, char rel, int page)
{
    int slot;
#if 0
    printf("\nRequest %c%02d\n", rel, page);  // for debugging
#endif
    pool->nrequests++;
    slot = pageInPool(pool, rel, page);
    if (slot != NONE)
        pool->nhits++;
    else { // page is not already in pool
        slot = findFree(pool);
        if (slot == NONE) {
            slot = findVictim(pool);
        }
        if (slot == NONE) {
            fprintf(stderr, "Failed to find slot for %c%02d\n", rel,
page);
            exit(1);
        }
        pool->nreads++;
        sprintf(pool->bufs[slot].id,"%c%02d", rel, page);
        pool->bufs[slot].pin = 0;
        pool->bufs[slot].dirty = 0;
        pool->bufs[slot].pop = 0;
    }
    // have a slot
    pool->bufs[slot].pin++;
    if (pool->bufs[slot].pop < MAX_USAGE)
        pool->bufs[slot].pop++;
#if 0
    showPoolState(pool);  // for debugging
#endif
    return slot;
}

void release_page(BufPool pool, char rel, int page)
{
#if 0
    printf("\nRelease %c%02d\n", rel, page);  // for debugging
#endif
    pool->nreleases++;
    int slot;
    slot = pageInPool(pool, rel, page);
    assert(slot != NONE);
    pool->bufs[slot].pin--;
    if (pool->bufs[slot].pop < MAX_USAGE)
        pool->bufs[slot].pop++;
```

```
#if 0
    showPoolState(pool);  // for debugging
#endif
}

// showPoolUsage(pool)
// - prints statistics counters for buffer pool

void showPoolUsage(BufPool pool)
{
    assert(pool != NULL);
    printf("#requests: %d\n",pool->nrequests);
    printf("#releases: %d\n",pool->nreleases);
    printf("#hits    : %d\n",pool->nhits);
    printf("#reads   : %d\n",pool->nreads);
    // not needed for this simulation
    // printf("#writes  : %d\n",pool->nwrites);
}

// showPoolState(pool)
// - display printable representation of buffer pool on stdout

void showPoolState(BufPool pool)
{
    int i; char *p;

    printf("\nFrames:    ");
    for (i = 0; i < pool->nbufs; i++)
        printf(" [%02d]", i);
    printf("\nContents:  ");
    for (i = 0; i < pool->nbufs; i++) {
        p = pageInBuf(pool,i);
        printf(" %4s", p);
    }
    printf("\nPinCount:  ");
    for (i = 0; i < pool->nbufs; i++) {
        printf(" %4d", pool->bufs[i].pin);
    }
    printf("\nPopularity:");
    for (i = 0; i < pool->nbufs; i++) {
        printf(" %4d", pool->bufs[i].pop);
    }
    printf("\nClock: %d\n", pool->clock);
}
```