PSET 6

Jonathan Lam
Prof. Hakner
ECE 357
Operating Systems
12/9/19

errfat.h

```
#ifndef __ERRFAT_H
#define __ERRFAT_H
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
extern char *PROG;
#define ERR_FAT(op, ctx, msg) {\
    dprintf(2, "%s: ERROR: %s (%s): %s\n", PROG, op, ctx, msg);\
    exit(EXIT_FAILURE);\
  }
#define ERRNO_FAT(op, ctx)\
  ERR_FAT(op, ctx, strerror(errno))
#endif
```

tas.h

```
#ifndef __TAS_H
int tas(volatile char *lock);
#define __TAS_H
#endif
```

tas.S (x64)

```
  .text       #See tas.S (32-bit version) for
.globl tas        #other comments
  .type tas,@function
tas:
  pushq %rbp
  movq  %rsp, %rbp
  movq  $1, %rax
  lock;xchgb  %al,(%rdi)  #arg1 is in the rdi register
  movsbq  %al,%rax    #sign-extend result into rax
  pop %rbp
  ret       #rax contains the return value
.Lfe1:
  .size tas,.Lfe1-tas
```

spinlock.h

```
#ifndef __SPINLOCK_H
#define __SPINLOCK_H
// shadows linux's spinlock_t in /include/linux/spinlock_types.h
// but that's what we're imitating here anyways
typedef volatile char spinlock_t;
void spin_lock(spinlock_t *sl);
void spin_unlock(spinlock_t *sl);
#endif
```

spinlock.c

```
#include "tas.h"
#include "spinlock.h"

void spin_lock(spinlock_t *sl) {
  while(tas(sl));
}
void spin_unlock(spinlock_t *sl) {
  *sl = 0;
}
```

seqlock.h

```
#ifndef __SEQLOCK_H
#define __SEQLOCK_H
#include "spinlock.h"

struct seqlock {
  spinlock_t lock;
  int count;
};
void write_seqlock(struct seqlock *s);
void write_sequnlock(struct seqlock *s);
int read_seqbegin(struct seqlock *s);
int read_seqretry(struct seqlock *s, int orig);
#endif
```

seqlock.c

```
#include <sched.h>
#include "seqlock.h"

// This implementation closely follows the lecture notes. I didn't experience
// issues with non-atomic count incrementing, so didn't protect the field
// with a spinlock.

void write_seqlock(struct seqlock *s) {
  spin_lock(&s->lock);
  s->count++;
}
void write_sequnlock(struct seqlock *s) {
  s->count++;
  spin_unlock(&s->lock);
};
int read_seqbegin(struct seqlock *s) {
  int a;
  while((a=s->count)%2)
    sched_yield();
  return a;
}
int read_seqretry(struct seqlock *s, int orig) {
  return s->count!=orig;
}
```

slab.h

```
#ifndef __SLAB_H
#define __SLAB_H
#include "spinlock.h"
#include "dll.h"

/**
  * Simple slab implementation. For simplicity, the slab is fixed-size,
  * may only slab-allocate struct dlls, and the freemap is a slot. Slab ops
  * are protected with a simple spinlock.
  */

// NSLOTS should be small enough to fill up
#define NSLOTS 10000

struct slab {
  char freemap[NSLOTS];
  struct dll slots[NSLOTS];
  spinlock_t sl;

  // for optimization purposes on insert -- see slab_alloc for details
  int pos;
};

/**
  * Allocate memory for slab. Not thread-safe.
  * @return         pointer to newly-allocated slab
  */
struct slab *slab_create();

/**
  * Unallocate slab. Not thread-safe.
  * @param          pointer to slab to de-allocate
  */
void slab_destroy(struct slab *slab);

/**
  * Allocate object in slab. Thread-safe.
  * @param slab     pointer to slab
  * @return         pointer to newly-allocated object, or NULL if slab is full
  */
void *slab_alloc(struct slab *slab);

/**
  * Deallocate object in slab. Thread-safe.
  * @param slab     pointer to slab
  * @param object   pointer to object to deallocate
  * @return         1 on success, -1 on failure
  */
int slab_dealloc(struct slab *slab, void *object);

// Allocate stats struct. Lumped in here to keep all mmap-ing in this file.
```

```
void stats_alloc();
#endif
```

slab.c

```
#include <sys/mman.h>
#include "errfat.h"
#include "slab.h"
#include "stats.h"

// See slab.h for more details.

struct slab *slab_create() {
  struct slab *slab;
  if((slab = (struct slab *)
        mmap(NULL, sizeof(struct slab),PROT_READ|PROT_WRITE,
             MAP_SHARED|MAP_ANONYMOUS, -1, 0)) == MAP_FAILED)
    ERRNO_FAT("mmap", "slab_create");
  slab->pos = 0;
  return slab;
}
void slab_destroy(struct slab *slab) {
  if(munmap(slab, sizeof(struct slab)) < 0)
    ERRNO_FAT("munmap", "slab_destroy");
}

// to increase performance, allocation searches for a free slot beginning
// from where it last left off (as opposed to searching from start every time)
// i.e., slab->pos
void *slab_alloc(struct slab *slab) {
  unsigned int pos_start;
  void *res_pos;

  spin_lock(&slab->sl);
  for(pos_start=slab->pos?slab->pos-1:NSLOTS-1;
      slab->freemap[slab->pos] && slab->pos!=pos_start;
      slab->pos=(slab->pos+1)%NSLOTS);
  if(slab->pos==pos_start) {
    spin_unlock(&slab->sl);
    return NULL;
  }
  slab->freemap[slab->pos] = 1;
  res_pos = slab->slots+slab->pos;
  spin_unlock(&slab->sl);
  return res_pos;
}
int slab_dealloc(struct slab *slab, void *object) {
  int pos;

  spin_lock(&slab->sl);
  pos = ((struct dll *)object) - slab->slots;
  if(pos>=NSLOTS || !slab->freemap[pos]) {
    spin_unlock(&slab->sl);
    return -1;
  }
  slab->freemap[pos] = 0;
  spin_unlock(&slab->sl);
  return 1;
}

// see slab.h for details
void stats_alloc() {
  if((stats = (struct stats *)
        mmap(NULL, sizeof(struct stats),PROT_READ|PROT_WRITE,
             MAP_SHARED|MAP_ANONYMOUS, -1, 0)) == MAP_FAILED)
    ERRNO_FAT("mmap", "struct stat");
}
```

dll.h

```
#ifndef __DLL_H
#define __DLL_H
#include "spinlock.h"
#include "seqlock.h"

/**
  * This implementation of a sorted integer-valued circular doubly-linked list
  * doesn't do much error checking, assumes correct inputs. It also assumes
  * that all nodes in a struct dll are allocated from the same struct slab
  * as its anchor. Both spinlock- and seqlock-protected versions are described
  * in this header file. A reasonable space optimization would be to store
  * the lock in the value field of the anchor, but this simple implementation
  * doesn't worry about saving space for clarity/simplicity of the lock.
  */

struct slab; // forward declaration b/c of circ. dep. -- see below
struct dll {
  int value;
  struct dll *fwd, *rev;

  // locks only used on anchor(s); see note above about saving space
  spinlock_t sl;          // for use with dll.c
  struct seqlock seqlock; // for use with dll2.c
};

#include "slab.h" // needs to be placed here b/c of circ. dep. b/t dll, slab

/**
  * Allocates and returns a DLL anchor. Not thread-safe (called from parent).
  * @param slab     slab to allocate from
  * @return         anchor node for DLL
  */
struct dll *dll_create(struct slab *slab);

/**
  * De-allocates all nodes in the DLL. Not thread-safe (called from parent).
  * @param anchor   anchor node for DLL
  * @param slab     slab where nodes are allocated
  */
void dll_destroy(struct dll *anchor, struct slab *slab);

/**
  * Insert an integer into the DLL. Thread-safe.
  * @param anchor   DLL anchor
  * @param value    integer value to add to DLL
  * @param slab     slab to allocate nodes from
  * @return         created node, or NULL on failure
  */
struct dll *spin_dll_insert(struct dll *anchor,int value,struct slab *slab);

/**
```

```
  * Delete a node from the DLL. Thread-safe.
  * @param anchor   DLL anchor
  * @param node     pointer to node to delete
  * @param slab     slab that node is allocated in
  * @return         1 on success, -1 on failure
  */
int spin_dll_delete(struct dll *anchor, struct dll *node,struct slab *slab);

/**
  * Find the first node with a given value in the DLL. Thread-safe.
  * @param anchor   DLL anchor
  * @param value    integer value to search for
  * @return         pointer to first node in DLL containing value, or NULL
  */
struct dll *spin_dll_find(struct dll *anchor,int value);

/**
  * Analogous functions protected using seqlock (extra credit). The insert
  * and delete functions are protected with write seqlocks, and the find is
  * protected with an optimistic read seqlock. Thread-safe. Keeps track of
  * statistics in stats (see stats.h);
  */
struct dll *seq_dll_insert(struct dll *anchor, int value, struct slab *slab);
int seq_dll_delete(struct dll *anchor, struct dll *node, struct slab *slab);
struct dll *seq_dll_find(struct dll *anchor, int value);

void print_dll(struct dll *anchor); // for debugging
#endif
```

dll.c

```
#include "errfat.h"
#include "dll.h"
#include "slab.h"

// This dll implementation uses spinlocking. See dll.h for more info.

struct dll *dll_create(struct slab *slab) {
  struct dll *anchor;
  if(!(anchor = (struct dll *) slab_alloc(slab)))
    return NULL;
  anchor->fwd = anchor->rev = anchor;
  return anchor;
}
void dll_destroy(struct dll *anchor, struct slab *slab) {
  // it_fwd to avoid accessing it->nxt after it has been deallocated (and
  // possibly already reallocated)
  struct dll *it, *it_fwd;
  it = anchor, it_fwd = it->fwd;
  do {
    it_fwd = it->fwd;
    slab_dealloc(slab, it);
  } while((it=it_fwd) != anchor);
}

struct dll *spin_dll_insert(struct dll *anchor, int value, struct slab *slab) {
  struct dll *new_node, *it;

  if(!(new_node = (struct dll *) slab_alloc(slab)))
    return NULL;
  new_node->value = value;

  spin_lock(&anchor->sl);
  for(it=anchor->fwd; it->value<value && it!=anchor; it=it->fwd);
  new_node->fwd = it;
  new_node->rev = it->rev;
  it->rev = it->rev->fwd = new_node;
  spin_unlock(&anchor->sl);
  return new_node;
}

int spin_dll_delete(struct dll *anchor, struct dll *node, struct slab *slab) {
  spin_lock(&anchor->sl);
  // this condition if multiple deletes on same dll in quick succession
  if(!node || node->fwd==node) {
    spin_unlock(&anchor->sl);
    return -1;
  }
  node->rev->fwd = node->fwd;
  node->fwd->rev = node->rev;
  node->fwd = node->rev = node;
  spin_unlock(&anchor->sl);

  if(slab_dealloc(slab, node)<0)
    ERR_FAT("slab_dealloc", "", "Deallocating node failed");
  return 1;
}

struct dll *spin_dll_find(struct dll *anchor, int value) {
  struct dll *it;

  spin_lock(&anchor->sl);
  // some preliminary checks/optimizations
  if(anchor->fwd==anchor || value<anchor->fwd->value
     || value>anchor->rev->value) {
    spin_unlock(&anchor->sl);
    return NULL;
  }

  for(it=anchor->fwd; it->value<value && it!=anchor; it=it->fwd);
  spin_unlock(&anchor->sl);
  return it->value==value && it!=anchor ? it : NULL;
}

// for debugging
```

```
void print_dll(struct dll *anchor) {
  struct dll *it;

  spin_lock(&anchor->sl);
  dprintf(1, "printing dll: ");
  for(it = anchor->fwd; it != anchor; it=it->fwd)
    dprintf(1, "%d ", it->value);
  dprintf(1, "\n");
  spin_unlock(&anchor->sl);
}
```

seqdll.c

```
#include "errfat.h"
#include "dll.h"
#include "slab.h"
#include "stats.h"

// This dll implementation uses seqlocking. See dll.h for more info.

struct dll *seq_dll_insert(struct dll *anchor, int value, struct slab *slab) {
  struct dll *new_node, *it;
  int seqlock_cnt;

  if(!(new_node = (struct dll *) slab_alloc(slab)))
    return NULL;
  new_node->value = value;

  write_seqlock(&anchor->seqlock);
  for(it=anchor->fwd; it->value<value&&it!=anchor&&it!=it->fwd; it=it->fwd);
  new_node->fwd = it;
  new_node->rev = it->rev;
  it->rev = it->rev->fwd = new_node;
  write_sequnlock(&anchor->seqlock);
  return new_node;
}

int seq_dll_delete(struct dll *anchor, struct dll *node, struct slab *slab) {
  write_seqlock(&anchor->seqlock);
  // this condition if multiple deletes on same dll in quick succession
  if(!node || node->fwd == node) {
    write_sequnlock(&anchor->seqlock);
    return -1;
  }
  node->rev->fwd = node->fwd;
  node->fwd->rev = node->rev;
  node->fwd = node->rev = node;
  write_sequnlock(&anchor->seqlock);

  if(slab_dealloc(slab, node)<0)
    ERR_FAT("slab_dealloc", "", "Deallocating node failed");
  return 1;
}
```

```
struct dll *seq_dll_find(struct dll *anchor, int value) {
  struct dll *it;
  int seqlock_cnt, cnt=0;

  do {
    seqlock_cnt = read_seqbegin(&anchor->seqlock);
    if(anchor->fwd==anchor || value<anchor->fwd->value
       || value>anchor->rev->value)
      return NULL;

    for(it=anchor->fwd; it->value<value&&it!=anchor&&it!=it->fwd; it=it->fwd);
  } while(++cnt, read_seqretry(&anchor->seqlock, seqlock_cnt));

  spin_lock(&stats->lock);
  stats->att_seqlock_read += cnt;
  stats->suc_seqlock_read++;
  spin_unlock(&stats->lock);
  return it->value==value && it!=anchor ? it : NULL;
}
```

spinlocktest.c

```
// usage: spinlocktest [thread_count] [sample_count]

#include <stdio.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <unistd.h>
#include "errfat.h"
#include "spinlock.h"

#define THREAD_CNT_DFL 8
#define SAMPLE_CNT_DFL 100000
#define PROG "spinlocktest"

// generate spinlock test
void transact_nospinlock(int *p, int sample_cnt) {
  for(int i = 0; i < sample_cnt; i++)
    (*p)++;
}
void transact_spinlock(int *p, spinlock_t *sl, int sample_cnt) {
  for(int i = 0; i < sample_cnt; i++) {
    spin_lock(sl);
    (*p)++;
    spin_unlock(sl);
  }
}
void generate_spinlock_test(int use_spinlock, int thread_cnt, int sample_cnt) {
  int *p, i, wstatus, pid;
  spinlock_t *sl;

  // create shared mmap region; first (sizeof(int)) bytes for data to rapidly
  // change, and last (sizeof(char)) bytes for spinlock
  if((p = (int *)mmap(NULL,sizeof(int)+sizeof(spinlock_t),PROT_READ|PROT_WRITE,
                       MAP_SHARED|MAP_ANONYMOUS, -1, 0)) == MAP_FAILED)
    ERRNO_FAT("mmap", "shared region");
  sl = (spinlock_t *)(p+1);

  // create thread_cnt processes; do sample_cnt transactions in each child,
  // don't do anything in parent
  for(i=0; i<thread_cnt; i++)
    switch(fork()) {
    case -1:
      ERRNO_FAT("fork", i);
    case 0:
      if(use_spinlock)
        transact_spinlock(p, sl, sample_cnt);
      else
        transact_nospinlock(p, sample_cnt);
      exit(EXIT_SUCCESS);
    }

  // aggregate results and cleanup; ignore wstatus
  for(i=0; i<thread_cnt; i++)
    if(pid=wait(&wstatus)<0)
      ERRNO_FAT("wait", pid);
  dprintf(2, "%d\n", *p);
  if(munmap(p, sizeof(int)+sizeof(spinlock_t))<0)
    ERRNO_FAT("munmap", "shared region");
}
int main(int argc, char **argv) {
  int thread_cnt, sample_cnt;
  if(argc<3) {
    thread_cnt = THREAD_CNT_DFL;
    sample_cnt = SAMPLE_CNT_DFL;
  } else {
    thread_cnt = atoi(argv[1]);
    sample_cnt = atoi(argv[2]);
  }

  dprintf(1, "Processes:\t%d\nSamples/proc:\t%d\nExpected total:\t%d\n---\n",
          thread_cnt, sample_cnt, thread_cnt*sample_cnt);
  dprintf(1, "w/o spinlock:\t");
  generate_spinlock_test(0, thread_cnt, sample_cnt);
  dprintf(1, "w/ spinlock:\t");
  generate_spinlock_test(1, thread_cnt, sample_cnt);
}
```

Sample output (spinlocktest.c)

```
Processes:  8
Samples/proc:     100000
Expected total:   800000
---
w/o spinlock:     292355
w/ spinlock:      800000
```

stats.h

```
#ifndef __STATS_H
#define __STATS_H
#include "spinlock.h"

extern struct stats *stats;
struct stats {
  spinlock_t lock;
  unsigned int att_seqlock_read, suc_seqlock_read, net_dll_len_chg;
};
#endif
```

slabtest.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
#include "errfat.h"
#include "dll.h"
#include "slab.h"
#include "spinlock.h"
#include "stats.h"

#define THREAD_CNT 16
#define SAMPLE_CNT 10000
#define SAMPLE_MAX 1000
char *PROG = "slabtest";

// check if dll is sorted and of correct length
void check_dll(struct dll *anchor, int exp_len, int is_seqlock) {
  struct dll *it, *it_fwd;
  int tot_cnt, err_cnt;

  dprintf(2, "=====\nSTRESS CHECK COMPLETE. CHECKING DLL.\n");
  for(it=anchor->fwd, it_fwd=it->fwd, tot_cnt=err_cnt=0; it!=anchor;
      it_fwd=(it=it_fwd)->fwd, tot_cnt++)
    if(it_fwd->value<it->value && it_fwd!=anchor) {
      printf("%d\n", it->value);
      err_cnt++;
    }
  dprintf(2, "=====\nOVERALL CHECK:\nSORTING ERRORS:\t%d\nDLL LENGTH:\t%d\n"
             "EXP DLL LENGTH:\t%d\nLENGTH ERROR:\t%d\n",
          err_cnt, tot_cnt, exp_len, (exp_len-tot_cnt)*(exp_len>tot_cnt?1:-1));
  if(is_seqlock)
    dprintf(2, "OPTIMISTIC SEQLOCK SUCCESS RATE: %d/%d (%f\%)\n",
            stats->suc_seqlock_read,
            stats->att_seqlock_read,
            ((float)stats->suc_seqlock_read)/stats->att_seqlock_read);
}
```

```
// driver for slab testing
struct stats *stats;
void generate_slab_test(int is_seqlock) {
  // att/suc_op_cnt: attempted and successful operation counts
  struct slab *slab;
  struct dll *dll, *p;
  struct timeval proc_start, proc_end;
  int i, wstatus, att_op_cnt[3], suc_op_cnt[3], net_len, pid;
  long elap_usec;

  struct dll *(*dll_insert)(struct dll *,int,struct slab *);
  int (*dll_delete)(struct dll *,struct dll *,struct slab *);
  struct dll *(*dll_find)(struct dll *,int);

  if(!(slab = slab_create()))
    ERR_FAT("slab_create", "main slab", "Error creating slab");
  if(!(dll = dll_create(slab)))
    ERR_FAT("dll_create", "main slab", "Error creating dll");

  // set up shared statistics memory region -- see stats.h
  stats_alloc();

  // get correct functions
  dprintf(2, "=====\n%s TEST\n", is_seqlock?"SEQLOCK":"SPINLOCK");
  dll_insert = is_seqlock?seq_dll_insert:spin_dll_insert;
  dll_delete = is_seqlock?seq_dll_delete:spin_dll_delete;
  dll_find = is_seqlock?seq_dll_find:spin_dll_find;

  memset(att_op_cnt, 0, 3*sizeof(int));
  memset(suc_op_cnt, 0, 3*sizeof(int));
  dprintf(2, "PROC\tATT INS\tATT DEL\tATT FND\tSUC INS\tSUC DEL\tSUC FND\t"
             "NET CHG\tELP TME\n");
  for(i = 0; i < THREAD_CNT; i++) {
    switch(fork()) {
    case -1:
      ERRNO_FAT("fork", i);
    case 0:
      srand(time(NULL)+i); // should generate unique seed for each proc
      gettimeofday(&proc_start, NULL);
      for(int j=0; j<SAMPLE_CNT; j++) {
        switch(rand()%3) {
        case 0:
          att_op_cnt[0]++;
          if((*dll_insert)(dll, rand()%SAMPLE_MAX, slab))
            suc_op_cnt[0]++;
          break;
        case 1:
          att_op_cnt[1]++;
          if((*dll_find)(dll, rand()%SAMPLE_MAX))
            suc_op_cnt[1]++;
          break;
        case 2:
          att_op_cnt[2]++;
          if((*dll_delete)(dll, dll_find(dll, rand()%SAMPLE_MAX), slab)>0)
            suc_op_cnt[2]++;
          break;
        }
      }
      gettimeofday(&proc_end, NULL);
      elap_usec = 1000000*(proc_end.tv_sec-proc_start.tv_sec)
                  + (proc_end.tv_usec-proc_start.tv_usec);
      dprintf(2, "%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%6.03lfs\n",
              i, att_op_cnt[0], att_op_cnt[1], att_op_cnt[2],
              suc_op_cnt[0], suc_op_cnt[1], suc_op_cnt[2],
              suc_op_cnt[0]-suc_op_cnt[2],
              elap_usec/1e6);
      spin_lock(&stats->lock);
      stats->net_dll_len_chg += suc_op_cnt[0]-suc_op_cnt[2];
      spin_unlock(&stats->lock);
      exit(EXIT_SUCCESS);
    }
  }

  for(i=0; i<THREAD_CNT; i++) {
    if(pid=wait(&wstatus)<0) {
      ERRNO_FAT("wait", pid);
    } else if(wstatus)
      // non-fatal notice: child process died with non-zero exit code
      dprintf(2, "%s: wait: \"%d\": Process terminated with exit status %d\n",
              pid, wstatus);
  }

  check_dll(dll, stats->net_dll_len_chg, is_seqlock);

  dll_destroy(dll, slab);
  slab_destroy(slab);
}
int main(void) {
  dprintf(2, "=====\nslabtest.c\n"
             "=====\nTHREAD_CNT:\t%d\nSAMPLE_CNT:\t%d\n"
             "SAMPLE_MAX:\t%d\nNSLOTS:\t\t%d\n",
          THREAD_CNT, SAMPLE_CNT, SAMPLE_MAX, NSLOTS);

  // with spinlock
  generate_slab_test(0);

  // with seqlock
  generate_slab_test(1);
}
```

Test case: Medium parameter values

```
=====
slabtest.c
=====
THREAD_CNT:       16
SAMPLE_CNT:       10000
SAMPLE_MAX:       10000
NSLOTS:           10000
=====
SPINLOCK TEST
PROC  ATT INS     ATT DEL     ATT FND     SUC INS     SUC DEL     SUC FND     NET CHG     ELP TME
3     3336        3317        3347        2631        1394        1404        1227        18.876s
6     3366        3290        3344        2586        1331        1376        1210        19.970s
5     3352        3246        3402        2244        1608        1694        550         20.515s
1     3350        3275        3375        2467        1473        1477        990         20.713s
15    3339        3348        3313        2307        1641        1669        638         22.016s
13    3324        3375        3301        2199        1698        1687        512         22.486s
8     3350        3386        3264        2201        1712        1664        537         22.669s
10    3384        3276        3340        2260        1652        1698        562         22.882s
12    3287        3384        3329        2215        1705        1727        488         22.959s
7     3353        3364        3283        2095        1762        1691        404         23.126s
0     3343        3395        3262        2292        1518        1459        833         23.236s
4     3327        3365        3308        2125        1747        1672        453         23.272s
2     3349        3344        3307        2114        1707        1687        427         23.301s
11    3425        3299        3276        2171        1675        1694        477         23.300s
14    3300        3323        3377        2010        1683        1746        264         23.304s
9     3385        3342        3273        2121        1689        1695        426         23.339s
=====
STRESS CHECK COMPLETE. CHECKING DLL.
=====
OVERALL CHECK:
SORTING ERRORS:   0
DLL LENGTH:       9998
EXP DLL LENGTH:   9998
LENGTH ERROR:     0
=====
SEQLOCK TEST
PROC  ATT INS     ATT DEL     ATT FND     SUC INS     SUC DEL     SUC FND     NET CHG     ELP TME
2     3309        3308        3383        2531        1405        1462        1069        5.230s
0     3435        3242        3323        2570        1392        1423        1147        5.271s
4     3427        3269        3304        2474        1503        1506        968         5.561s
10    3341        3425        3234        2349        1572        1446        903         5.584s
15    3306        3300        3394        2337        1673        1690        647         5.984s
1     3270        3386        3344        2202        1594        1575        627         6.026s
8     3396        3305        3299        2155        1683        1668        487         6.050s
3     3281        3393        3326        2228        1702        1680        548         6.125s
11    3339        3307        3354        2151        1706        1732        419         6.169s
14    3296        3324        3380        2129        1625        1717        412         6.195s
6     3304        3311        3385        2100        1687        1698        402         6.241s
9     3395        3285        3320        2218        1652        1704        514         6.247s
7     3419        3297        3284        2275        1671        1680        595         6.271s
13    3276        3379        3345        2101        1757        1712        389         6.267s
```

```
5     3307        3390        3303        2155        1708        1628        527         6.279s
12    3302        3288        3410        2061        1661        1717        344         6.276s
=====
STRESS CHECK COMPLETE. CHECKING DLL.
=====
OVERALL CHECK:
SORTING ERRORS:   0
DLL LENGTH:       9998
EXP DLL LENGTH:   9998
LENGTH ERROR:     0
OPTIMISTIC SEQLOCK SUCCESS RATE: 106558/256690 (0.415123%)
```

Test case: More trials, higher chance of find/delete (lower SAMPLE_MAX)

```
=====
slabtest.c
=====
THREAD_CNT:       16
SAMPLE_CNT:       100000
SAMPLE_MAX:       1000
NSLOTS:           10000
=====
SPINLOCK TEST
PROC  ATT INS     ATT DEL     ATT FND     SUC INS     SUC DEL     SUC FND     NET CHG     ELP TME
2     33165       33485       33350       31070       30432       30302       768         265.390s
9     33330       33346       33324       30993       30467       30418       575         275.320s
1     33102       33369       33529       30927       30254       30485       442         277.109s
8     33297       33415       33288       31063       30596       30393       670         277.906s
13    33506       33296       33198       31142       30428       30300       842         279.324s
14    33376       33348       33276       31081       30459       30361       720         279.644s
3     33281       33360       33359       30961       30319       30170       791         280.501s
5     33520       33289       33191       31206       30134       29868       1338        282.112s
4     33329       33231       33440       31094       30357       30518       576         282.790s
12    33286       33210       33504       31037       30376       30583       454         282.915s
7     33351       33124       33525       30951       30339       30683       268         283.135s
15    33380       33438       33182       30848       30690       30376       472         283.849s
6     33178       33516       33306       30814       30557       30454       360         284.059s
10    33146       33455       33399       30874       30539       30519       355         284.186s
0     33252       33453       33295       30907       30545       30271       636         284.226s
11    33484       33164       33352       31228       30326       30503       725         284.216s
=====
STRESS CHECK COMPLETE. CHECKING DLL.
=====
OVERALL CHECK:
SORTING ERRORS:   0
DLL LENGTH: 9992
EXP DLL LENGTH:   9992
LENGTH ERROR:     0
=====
SEQLOCK TEST
PROC  ATT INS     ATT DEL     ATT FND     SUC INS     SUC DEL     SUC FND     NET CHG     ELP TME
4     33393       33287       33320       31007       29765       29761       1246        65.347s
0     33483       33355       33162       31162       29767       29589       1573        67.084s
2     33666       33139       33195       31143       30205       30188       955         69.178s
1     33308       33374       33318       30920       30233       30173       747         69.597s
15    33461       33104       33435       30881       29976       30383       498         69.763s
3     33304       33288       33408       30825       30184       30279       546         70.631s
6     33364       33336       33300       30869       30306       30332       537         70.666s
7     33421       33266       33313       30884       30369       30335       549         70.781s
10    33145       33598       33257       30665       30542       30162       503         70.850s
8     33341       33325       33334       30677       30257       30347       330         70.885s
13    33187       33504       33309       30680       30440       30306       374         71.035s
11    33387       33283       33330       30797       30309       30179       618         71.223s
14    33360       33350       33290       30823       30400       30259       564         71.212s
9     33338       33375       33287       30740       30405       30277       463         71.294s
```

```
12    33070       33428       33502       30572       30437       30500       72          71.297s
5     33145       33756       33099       30441       30772       30017       424         71.323s
=====
STRESS CHECK COMPLETE. CHECKING DLL.
=====
OVERALL CHECK:
SORTING ERRORS:   0
DLL LENGTH:       9999
EXP DLL LENGTH:   9999
LENGTH ERROR:     0
OPTIMISTIC SEQLOCK SUCCESS RATE: 1066512/2874594 (0.371013%)
```

Sample test case: Many operations on small dlls

```
=====
slabtest.c
=====
THREAD_CNT: 8
SAMPLE_CNT: 10000000
SAMPLE_MAX: 10
NSLOTS:           10
=====
SPINLOCK TEST
PROC  ATT INS     ATT DEL     ATT FND     SUC INS     SUC DEL     SUC FND     NET CHG     ELP TME
5     3330938     3333905     3335157     1343106     1412584     1365299     -22193      36.062s
0     3330540     3337773     3331687     1382338     1414653     1361455     20883       37.336s
7     3332757     3334019     3333224     1348168     1412597     1362722     -14554      37.329s
6     3333522     3333159     3333319     1353921     1417094     1369539     -15618      37.399s
2     3332200     3333983     3333817     1380033     1417873     1364507     15526       37.494s
1     3336070     3332361     3331569     1384245     1420747     1369157     15088       37.568s
3     3334897     3332747     3332356     1353696     1420963     1370791     -17095      37.585s
4     3333693     3332416     3333891     1396243     1427207     1378272     17971       37.595s
=====
STRESS CHECK COMPLETE. CHECKING DLL.
=====
OVERALL CHECK:
SORTING ERRORS:   0
DLL LENGTH: 8
EXP DLL LENGTH:   8
LENGTH ERROR:     0
=====
SEQLOCK TEST
PROC  ATT INS     ATT DEL     ATT FND     SUC INS     SUC DEL     SUC FND     NET CHG     ELP TME
0     3333601     3335776     3330623     1517381     1554593     1514760     2621        8.165s
5     3333186     3333231     3333583     1517783     1554074     1512693     5090        8.349s
6     3334786     3331539     3333675     1517505     1550790     1513189     4316        8.791s
7     3332195     3333908     3333897     1514807     1552548     1513864     943         8.866s
2     3332227     3333074     3334699     1506656     1549849     1513289     -6633       8.939s
4     3330786     3335409     3333805     1507838     1552305     1512145     -4307       8.988s
1     3333787     3330356     3335857     1512716     1549427     1510325     2391        8.992s
3     3335166     3330967     3333867     1507613     1550340     1512025     -4412       8.996s
=====
STRESS CHECK COMPLETE. CHECKING DLL.
=====
OVERALL CHECK:
SORTING ERRORS:   0
DLL LENGTH: 9
EXP DLL LENGTH:   9
LENGTH ERROR:     0
OPTIMISTIC SEQLOCK SUCCESS RATE: 42246863/54273870 (0.778401%)
```

Sample test case: Single-threaded performance

```
=====
slabtest.c
=====
THREAD_CNT:       1
SAMPLE_CNT:       10000000
SAMPLE_MAX:       1000
NSLOTS:           1000
=====
SPINLOCK TEST
PROC  ATT INS     ATT DEL     ATT FND     SUC INS     SUC DEL     SUC FND     NET CHG     ELP TME
0     3334101     3330914     3334985     1667039     1663457     1666040     999         25.803s
=====
STRESS CHECK COMPLETE. CHECKING DLL.
=====
OVERALL CHECK:
SORTING ERRORS:   0
DLL LENGTH: 999
EXP DLL LENGTH:   999
LENGTH ERROR:     0
=====
SEQLOCK TEST
PROC  ATT INS     ATT DEL     ATT FND     SUC INS     SUC DEL     SUC FND     NET CHG     ELP TME
0     3332923     3333276     3333801     1669508     1666818     1668511     997         25.442s
=====
STRESS CHECK COMPLETE. CHECKING DLL.
=====
OVERALL CHECK:
SORTING ERRORS:   0
DLL LENGTH:       997
EXP DLL LENGTH:   997
LENGTH ERROR:     0
OPTIMISTIC SEQLOCK SUCCESS RATE: 6653891/6653891 (1.000000%)
```

Sample test case: Many threads

```
=====
slabtest.c
=====
THREAD_CNT:       1000
SAMPLE_CNT:       1000
SAMPLE_MAX:       1000
NSLOTS:           1000
=====
SPINLOCK TEST
PROC  ATT INS     ATT DEL     ATT FND     SUC INS     SUC DEL     SUC FND     NET CHG     ELP TME
0     340         339         321         340         82          83          257         0.002s
1     313         346         341         313         87          78          235         0.002s
5     328         329         343         324         174         167         157         0.091s
210   354         325         321         139         159         149         -10         4.670s
796   332         326         342         181         159         168         13          4.632s
16    354         316         330         132         150         136         -4          8.046s
539   366         310         324         119         114         126         -7          16.551s
742   328         370         302         139         149         118         21          19.252s
551   341         299         360         101         115         142         -41         23.219s
3     319         345         336         272         131         118         154         25.562s
723   346         314         340         133         121         133         0           25.315s
636   360         318         322         111         114         137         -26         32.399s
[...truncated…]
431   341         324         335         143         129         146         -3          341.603s
34    329         322         349         149         150         169         -20         342.821s
680   347         304         349         160         138         154         6           340.448s
671   324         332         344         165         134         164         1           340.425s
357   319         351         330         157         158         155         2           342.040s
493   331         351         318         154         175         151         3           341.646s
102   355         314         331         185         152         154         31          342.796s
830   340         343         317         163         155         152         11          339.345s
623   321         370         309         144         168         148         -4          340.491s
=====
STRESS CHECK COMPLETE. CHECKING DLL.
=====
OVERALL CHECK:
SORTING ERRORS:   0
DLL LENGTH:       999
EXP DLL LENGTH:   999
LENGTH ERROR:     0
=====
SEQLOCK TEST
PROC  ATT INS     ATT DEL     ATT FND     SUC INS     SUC DEL     SUC FND     NET CHG     ELP TME
0     308         340         352         308         90          91          217         0.002s
3     316         334         350         316         106         112         204         0.002s
1     325         321         354         264         92          113         151         4.840s
113   306         338         356         130         136         149         -19         5.297s
258   320         365         315         134         167         124         10          6.604s
153   317         340         343         133         142         138         -5          6.661s
55    324         328         348         182         153         181         1           7.058s
364   311         357         332         135         166         127         8           6.829s
```

```
[...truncated…]
994   349         332         319         149         142         131         18          12.768s
117   353         320         327         152         142         139         13          17.253s
561   333         342         325         147         149         163         -16         16.277s
921   326         315         359         144         137         151         -7          14.014s
660   307         310         383         120         126         177         -57         15.507s
953   328         354         318         144         166         142         2           12.804s
977   326         316         358         140         131         143         -3          12.781s
727   325         321         354         161         137         165         -4          15.381s
944   338         314         348         155         143         162         -7          13.582s
754   321         350         329         148         159         146         2           14.692s
=====
STRESS CHECK COMPLETE. CHECKING DLL.
=====
OVERALL CHECK:
SORTING ERRORS:   0
DLL LENGTH:       997
EXP DLL LENGTH:   997
LENGTH ERROR:     0
OPTIMISTIC SEQLOCK SUCCESS RATE: 664770/1022575 (0.650094%)
```

Summary of results from test cases

- Both these spinlock and seqlock implementations seem to be successful in maintaining a DLL correctly (at least, the length of the DLL after the stress test is equal to the net changes in length to the DLL from each process, and it remains sorted.)
- It seems that with random results, we can expect at least a 30% success rate.
- For the first three test cases, with a medium number of samples and threads, seqlock seems to be 4-4.5x faster than seqlock. In the single-threaded test case, both appear to have almost the same performance (as expected). For highly-threaded and small number of trials per thread in the last test case, there was an enormous 20x speed improvement.
- The DLL/slab operations are much faster on small DLLs/slabs (expected because both only use linear operations), but it’s unclear whether this affects spinlocks or seqlocks more.