TÖL309G Tölvutækni og forritun

Verkefni 2

Hjörvar Sigurðsson

1.

Kóði:

```
/*
 * csim.c - A cache simulator that can replay traces from Valgrind
 *     and output statistics such as number of hits, misses, and
 *     evictions.  The replacement policy is LRU.
 *
 * Implementation and assumptions:
 *  1. Each load/store can cause at most one cache miss. (I examined the
trace,
 *     the largest request I saw was for 8 bytes).
 *  2. Instruction loads (I) are ignored, since we are only interested in
evaluating
 *     data cache performance.
 *  3. Data modify (M) is treated as a load followed by a store to the same
 *     address. Hence, an M operation can result in two cache hits, or a miss
and a
 *     hit plus an possible eviction.
 *
 */
#include <getopt.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <assert.h>
#include <math.h>
#include <limits.h>
#include <string.h>
#include <errno.h>

/* Type: Memory address */
typedef unsigned long long int mem_addr_t;


/* Hér fyrir neðan eru ýmsar víðværar breytur skilgreindar.  Þið munuð nota
þessar
   en þurfið líka að bæta við nokkrum í viðbót (t.d. skilgreiningu á línu og
   skilgreiningu á skyndiminninu sjálfu)
 */
```

```c
/* Globals set by command line args */
int s = 0; /* set index bits */
int b = 0; /* block offset bits */
int E = 0; /* associativity */
char* trace_file = NULL;

/* Derived from command line args */
int S; /* number of sets */
int B; /* block size (bytes) */

/* Counters used to record cache statistics */
int miss_count = 0;
int hit_count = 0;
int eviction_count = 0;
unsigned long long int lru_counter = 1;

/* Gagnagrindur */
typedef struct skyndiminnislina {
    int valid;
    mem_addr_t tag;
    unsigned long long int lru;
} skyndiminnislina;

typedef skyndiminnislina* skyndiminnissett;
typedef skyndiminnissett* skyndiminni;
skyndiminni skm;

// Held utan um hæsta lru gildi svo ég viti hvar ég byrja að kanna niður.
unsigned long long int haestaLruGildi = 0;


/*
 * initCache - Allocate memory, write 0's for valid and tag and LRU
 */
void initCache()
{
    /* Þið þurfið að útfæra þetta fall sem upphafsstillir gagnagrindurnar
     */

    skm = malloc(sizeof(struct skyndiminnislina) * S);
    for (int i = 0; i < S; i++) {
        skm[i] = malloc(sizeof(struct skyndiminnislina) * E);
        for (int j = 0; j < E; j++) {
            skm[i][j].valid = 0;
            skm[i][j].tag = 0;
            skm[i][j].lru = 0;
        }
    }
}
```

```c
/*
 * freeCache - free allocated memory
 */
void freeCache()
{

    /* Þið þurfið að útfæra þetta fall sem skilar til baka úthlutuðu minni
     */

    for (int i = 0; i < S; i++) {
        free(skm[i]);
    }
    free(skm);

}


/*
 * accessData - Access data at memory address addr.
 *   If it is already in cache, increast hit_count
 *   If it is not in cache, bring it in cache, increase miss count.
 *   Also increase eviction_count if a line is evicted.
 */
void accessData(mem_addr_t addr)
{

    /* Þið þurfið að útfæra þetta fall sem útfærir minnisaðgang
     */

    unsigned long long int laegstiLru = haestaLruGildi;
    unsigned long long int laegstai;
    unsigned long long int laegstaj;
    int varISkm = 0;

    // Er addr í skyndiminni?
    for (int i = 0; i < S; i++) {
        for (int j = 0; j < E; j++) {
            if (skm[i][j].tag == addr && skm[i][j].valid == 1) {
                skm[i][j].lru++;
                hit_count++;
                varISkm = 1;
            }
            if (skm[i][j].lru > haestaLruGildi) {
                haestaLruGildi = skm[i][j].lru;
            }
            if (skm[i][j].lru < laegstiLru) {
                laegstiLru = skm[i][j].lru;
```

```
                laegstai = i;
                laegstaj = j;
            }
        }
    }

    // addr var ekki í skyndiminni.
    if (varISkm == 0) {
        if (skm[laegstai][laegstaj].valid == 1) {
            eviction_count++;
        }
        skm[laegstai][laegstaj].tag = addr;
        skm[laegstai][laegstaj].valid = 1;
        miss_count++;
    }


}


/*
 * replayTrace - replays the given trace file against the cache
 */
void replayTrace(char* trace_fn)
{
    char buf[1000];
    mem_addr_t addr=0;
    unsigned int len=0;
    FILE* trace_fp = fopen(trace_fn, "r");

    if(!trace_fp){
        fprintf(stderr, "%s: %s\n", trace_fn, strerror(errno));
        exit(1);
    }

    while( fgets(buf, 1000, trace_fp) != NULL) {
        if(buf[1]=='S' || buf[1]=='L' || buf[1]=='M') {
            sscanf(buf+3, "%llx,%u", &addr, &len);

            accessData(addr);

            /* If the instruction is R/W then access again */
            if(buf[1]=='M')
                accessData(addr);
        }
    }

    fclose(trace_fp);
}
```

```c
/*
 * printSummary - Summarize the cache simulation statistics
 */
void printSummary(int hits, int misses, int evictions)
{
    printf("hits: %d  misses: %d  evictions: %d\n", hits, misses, evictions);
    printf("miss ratio: %.2f%%\n", 100.0*misses/(hits+misses));
}

/*
 * printUsage - Print usage info
 */
void printUsage(char* argv[])
{
    printf("Usage: %s [-h] -s <num> -E <num> -b <num> -t <file>\n", argv[0]);
    printf("Options:\n");
    printf("  -h         Print this help message.\n");
    printf("  -s <num>   Number of set index bits.\n");
    printf("  -E <num>   Number of lines per set.\n");
    printf("  -b <num>   Number of block offset bits.\n");
    printf("  -t <file>  Trace file.\n");
    printf("\nExamples:\n");
    printf("  linux>  %s -s 4 -E 1 -b 4 -t traces/yi.trace\n", argv[0]);
    printf("  linux>  %s -s 8 -E 2 -b 4 -t traces/yi.trace\n", argv[0]);
    exit(0);
}

/*
 * main - Main routine
 */
int main(int argc, char* argv[])
{
    char c;

    while( (c=getopt(argc,argv,"s:E:b:t:h")) != -1){
        switch(c){
        case 's':
            s = atoi(optarg);
            break;
        case 'E':
            E = atoi(optarg);
            break;
        case 'b':
            b = atoi(optarg);
            break;
        case 't':
            trace_file = optarg;
            break;
```

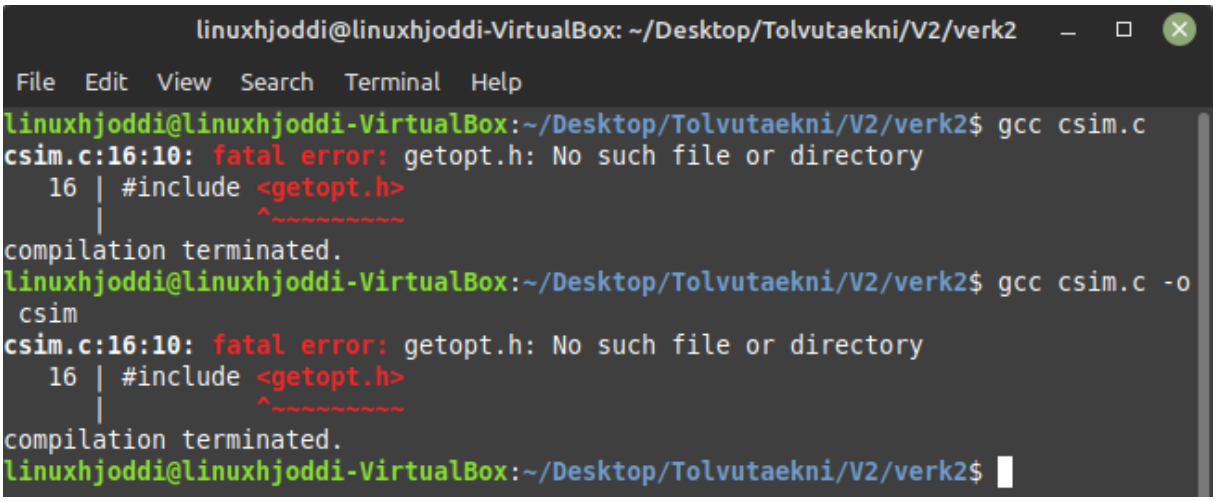
```
        case 'h':
            printUsage(argv);
            exit(0);
        default:
            printUsage(argv);
            exit(1);
        }
    }

    /* Make sure that all required command line args were specified */
    if (s == 0 || E == 0 || b == 0 || trace_file == NULL) {
        printf("%s: Missing required command line argument\n", argv[0]);
        printUsage(argv);
        exit(1);
    }

    /* Compute S, E and B from command line args */
    S = (unsigned int) (1 << s);
    B = (unsigned int) (1 << b);

    /* Initialize cache */
    initCache();

    /* Run the simulation */
    replayTrace(trace_file);

    /* Free allocated memory */
    freeCache();

    /* Output the hit and miss statistics */
    printSummary(hit_count, miss_count, eviction_count);
    return 0;
}
```

Keyrsla:

Ég gat ekki fengið forritið til að keyrast. Ég fékk stöðugar villur varðandi línur 16 og 18 í forritinu:

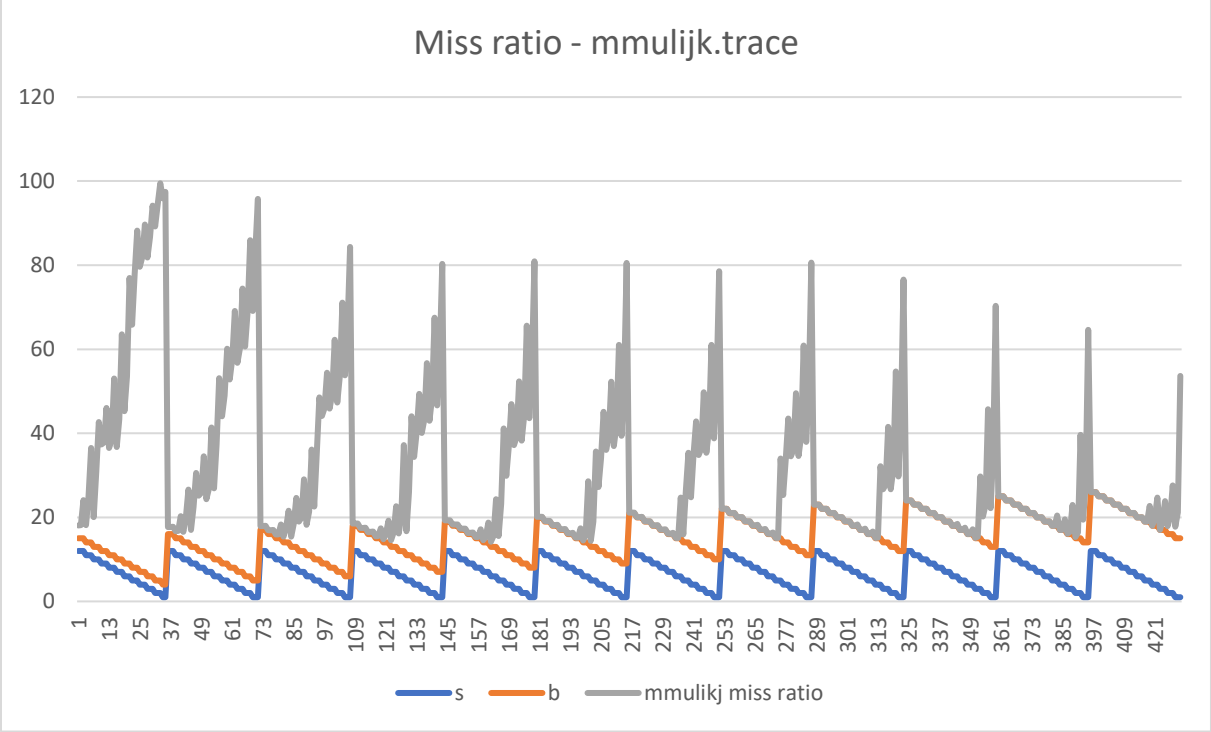
```
#include <getopt.h>
#include <unistd.h>
```

Keyrsla virkaði ekki í Linux mint Oracle VM VirtualBoxinu sem ég hef notað hingað til:

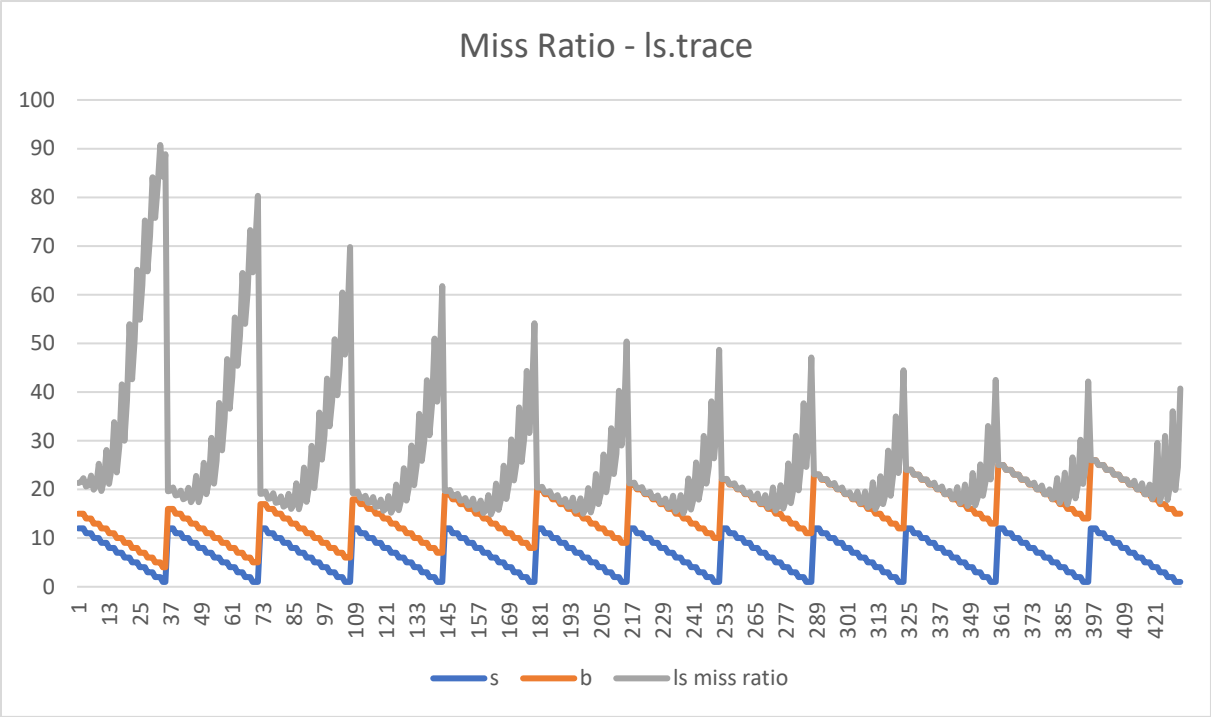

linuxhjoddi@linuxhjoddi-VirtualBox: ~/Desktop/Tolvutaekni/V2/verk2
File Edit View Search Terminal Help
linuxhjoddi@linuxhjoddi-VirtualBox:~/Desktop/Tolvutaekni/V2/verk2$ gcc csim.c
csim.c:16:10: fatal error: getopt.h: No such file or directory
16 | #include <getopt.h>
| ^~~~~~~~~~
compilation terminated.
linuxhjoddi@linuxhjoddi-VirtualBox:~/Desktop/Tolvutaekni/V2/verk2$ gcc csim.c -o
csim
csim.c:16:10: fatal error: getopt.h: No such file or directory
16 | #include <getopt.h>
| ^~~~~~~~~~
compilation terminated.
linuxhjoddi@linuxhjoddi-VirtualBox:~/Desktop/Tolvutaekni/V2/verk2$

2.

Því hærri sem fjöldi mengja og hliðrunarbita, því lægri er skellahlutfallið (sjá myndir 2.1 og 2.2). Þetta gildir um bæði mmulijk.trace sem og ls.trace. Þó virðist áhrif fjölda hliðrunarbita vera meiri en áhrif fjölda hliðrunarbita. Lægsta skellahlutfallið fyrir mmulijk.trace er 0.01, en það fæst þegar b er á bilinu 13-14, og s er á bilinu 4-12. Lægsta skellahlutfallið fyrir ls.trace er 0.02, en það fæst þegar b er 14, og s er á bilinu 5-12.

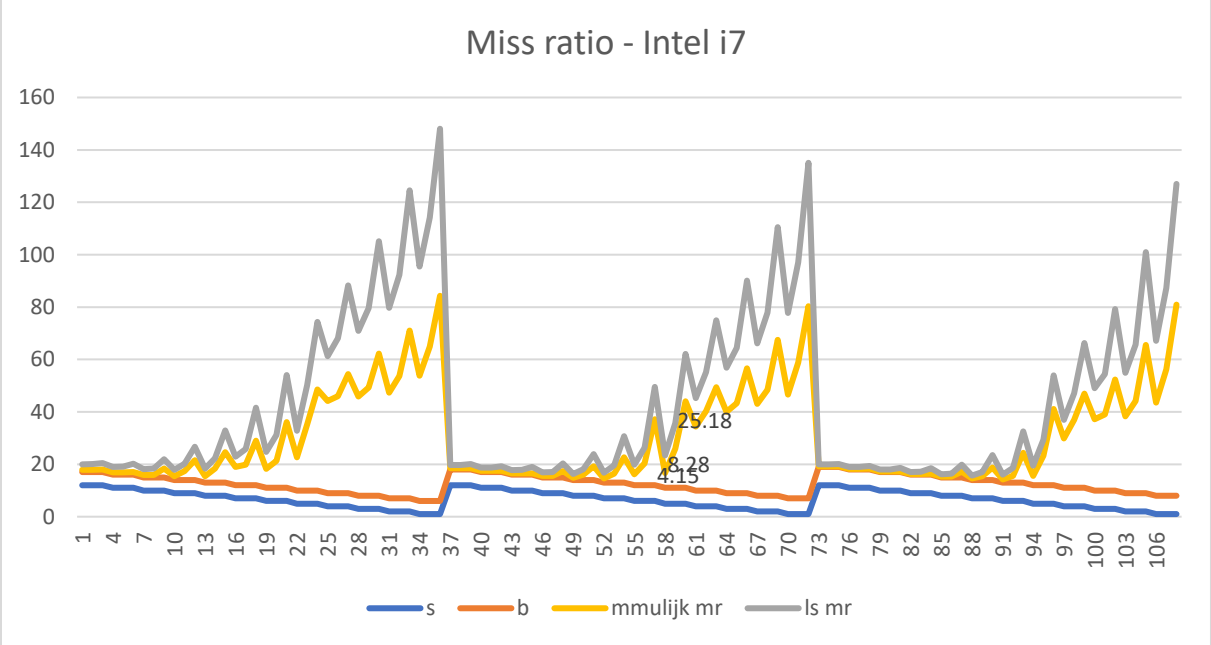


Mynd 2.1. Samband mengjafjölda (s), hliðrunarbita (b) og skellahlutfalls fyrir mmulijk.trace.



Mynd 2.2. Samband mengjafjölda (s), hliðrunarbita (b) og skellahlutfalls fyrir ls.trace.

Intel Core i7 hefur 64 bæta línur og 64 mengi. Skellahlutfall slíkarra uppsetningar er merkt inná mynd 2.3.



Mynd 2.3. Samband mengjafjölda (s), hliðrunarbita (b) og skellahlutfalls fyrir mmulijk.trace. Uppsetning Intel Core i7 er merkt inná myndina með þremur punktum í svörtum lit.

Eins og sjá má á mynd 2.3, þá mætti minnka skellahlutfall Intel Core i7 örgjörvans með því að hækka fjölda mengja og hliðrunarbita.

Mögulega valdi Intel þessa uppsetningu frekar en hærri fjölda mengja og hliðrunarbita vegna þess að tekur styttri tíma að hljóta aðgang að minnislínu ef fjöldi mengja og hliðrunarbita er lægri en ef hann væri hærri. Þeir gætu því hafa ákveðið að þessi uppsetning er góð málamiðlun milli hraða í að fá aðgang að og ferðast um minnið annars vegar, og skellahlutfalls hins vegar.

Á mynd 2.3 má einnig sjá muninn á skellahlutfalli forritana tveggja, mmulijik.trace og ls.trace. Þegar fjöldi mengja og hliðrunarbita er hár, þá er skellahlutfallið svipað í báðum forritum. Hinsvegar, ef að fjöldi mengja og hliðrunarbita er lár, þá er skellahlutfallið hærra í ls.trace heldur en í mmulijik.trace. Líklega er það vegna þess að ls.trace fer oftar í innri gagnagrindur í stýrikerfinu þar sem gögnin eru ekki á eins reglulegu formi, en því er sjaldnar hægt að setja stórann bút af gögnum í skyndiminni sem svo er notaður. Í mmulijk.trace er farið í gegnum fylki á reglubundinn máta, en því er hægt að setja stórann bút af gögnum í skyndiminni þar sem nokkuð öruggt er að gögnin verða notuð í fylkjaaðgerðunum.