Tölvutækni og Forritun Verkefni 2

brj46

November 2024

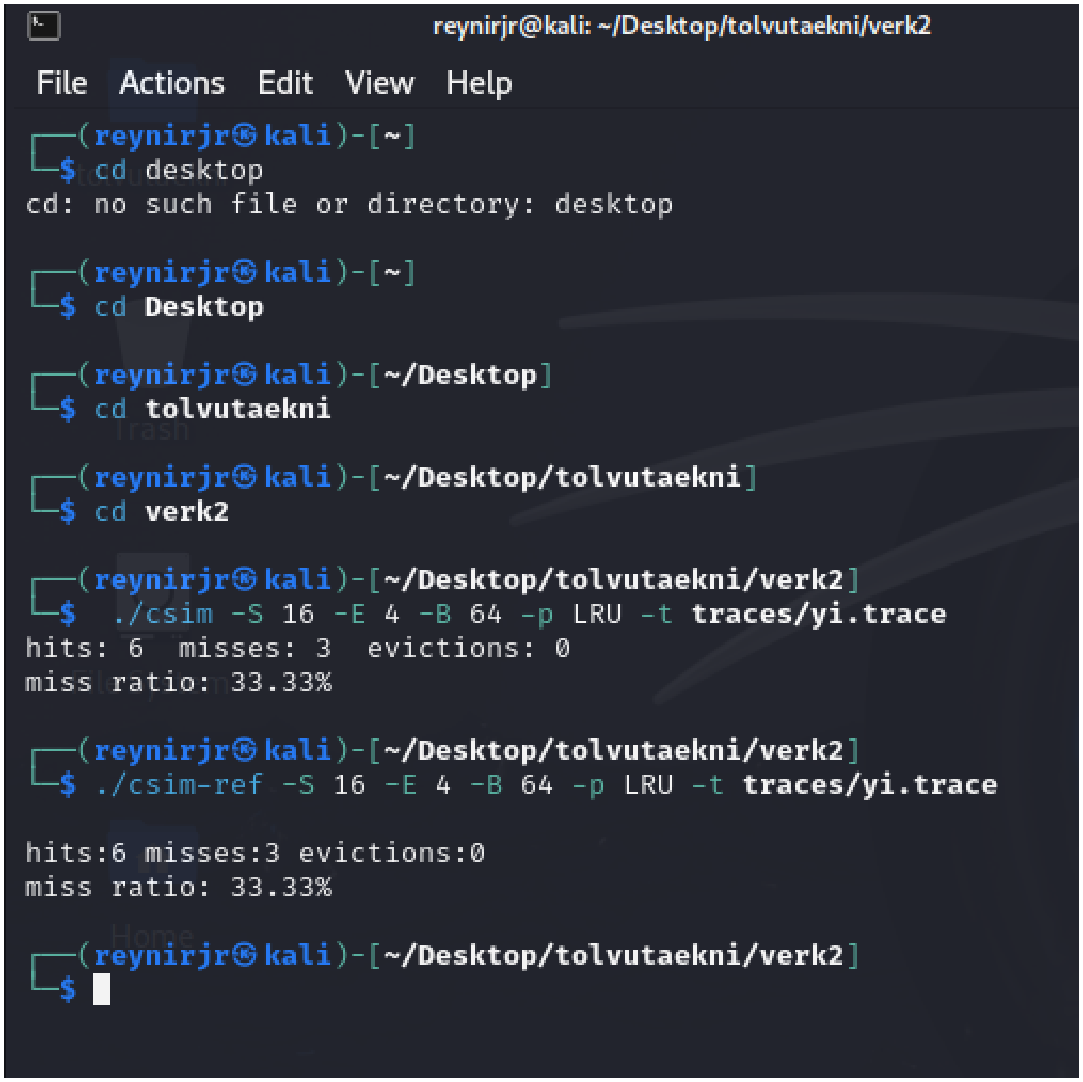
HÁSKÓLI
ÍSLANDS

Forritskóði

```
      GNU nano 8.2

         csim.c
/*
 * csim.c - A cache simulator that can replay traces from
    Valgrind
 *     and output statistics such as number of hits, misses,
    and
 *     evictions.  The default replacement policy is LRU.
 *
 * Implementation and assumptions:
 *  1. Each load/store can cause at most one cache miss. (the
    largest requests in the
 *     trace files are 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;

/* Type: Cache line */
typedef struct cache_line {
    char valid;
    mem_addr_t tag;
    unsigned long long int lru;    /* counter for LRU */
    unsigned long long int fifo;   /* counter for FIFO */
} cache_line_t;

typedef cache_line_t* cache_set_t;
typedef cache_set_t* cache_t;


/* Her fyrir ne an eru  msar  v   v  r a r  breytur skilgreindar
   .    i   munu  nota  essar
   breytur og   ttu   ekki  a    urfa   a   skilgreina fleiri
```

```
      v   v  r a r  breytur.
 */


/* Globals set by command line args */
int S = 0;     /* number of sets */
int B = 0;     /* block size (bytes) */
int E = 0;     /* associativity */
char* trace_file = NULL;   /* string with filename */
char* policy = NULL;       /* string with "LRU", "FIFO" or "
   RAND" */
int policy_code = 1;       /* code for replacement rule 1=LRU,
   2=FIFO, 3=RAND */

/* Derived from command line args */
int s;    /* set index bits */
int b;    /* block offset bits */

/* 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;
unsigned long long int fifo_counter = 1;

/* The cache we are simulating */
cache_t cache;
mem_addr_t set_index_mask;



/*
 * initCache - Allocate memory, write 0's for valid and tag and
     LRU
 */
void initCache()
{

    /*   i     urfi    a     tfra     etta   fall sem
       upphafsstillir gagnagrindurnar.
         a     arf   a   b  a til skyndiminni  sem fylki af
          mengjum og hvert mengi
       inniheldur mengi af l num (nota malloc).    a     arf
          a   upphafsstilla
        ll   s v i  in      llum   l  n u m .
     */
         S = 1 << s;
         cache = malloc(S * sizeof(cache_set_t));
        for (int i = 0; i< S; i++){
                cache[i] = malloc(E * sizeof(cache_line_t));
                for( int j = 0; j < E; j++){
                        cache[i][j].valid = 0;
                        cache[i][j].tag = 0;
                        cache[i][j].lru = 0;
                        cache[i][j].fifo = 0;
                }
        }
}


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

    /*   i      urfi    a      tfra     etta   fall sem skilar til
       baka    thlutuu    minni
     */
        for(int i = 0; i< S; i++){
                free(cache[i]);
        }
        free(cache);

}


/*
 * 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)
{

    /*   i      urfi    a      tfra     etta   fall sem    tfrir
       minnisa gang.
       Helstu skref:
        1. Finna hva a mengi l nan   tti   a   vera   .
        2. Athuga hvort l nan er       v   mengi (fara
           gegnum  ll   s tin og ath. valid og tag).
        3. Ef svo er,       smellur og uppf ra LRU
        4. Ef l nan finnst ekki    neinu s ti ,      skellur
        5.  Ef eitthva   s  ti      menginu er    lglegt
           fer n ja l nan    anga
        6.  Annars  arf   a   velja l nu til a   henda  t   me
            r ttri   tskiptireglu
        7.  Setja n ja  l nu inn og uppf ra svi      s tinu.
     */
        mem_addr_t set_index = (addr >> b) & ((1 << s) -1);
        mem_addr_t tag = addr >> (s + b);
        cache_set_t set = cache[set_index];
        int empty_line = -1;
        unsigned long long min_counter = ULLONG_MAX;
        int eviction_line= -1;
        for( int i = 0; i < E; i++){
                if(set[i].valid) {
                        if(set[i].tag == tag) {
                                hit_count++;
                                if(policy_code == 1) {
                                set[i].lru = lru_counter++;
                                }
                                return;
                        } else {
                                if (policy_code == 1 && set[i].
                                   lru < min_counter) {
                                        min_counter = set[i].
                                           lru;
                                        eviction_line = i;
                                }else if (policy_code == 2 &&
                                   set[i].fifo < min_counter)
                                   {
                                        min_counter = set[i].
                                           fifo;
                                        eviction_line = i;
                                }
                        }
                }else if (empty_line == -1) {
                        empty_line = i;
                }
        }
miss_count++;

int target_line = (empty_line != -1) ? empty_line :
   eviction_line;

if( empty_line == -1){
        eviction_count++;
}
if(policy_code == 3 && empty_line == -1){
        target_line = rand() % E;
        eviction_count++;
}

set[target_line].valid = 1;
set[target_line].tag = tag;
if(policy_code == 1){
        set[target_line].lru = lru_counter++;
}else if(policy_code == 2){
        set[target_line].fifo = fifo_counter++;
}else if(policy_code == 3){
        if(empty_line == -1){
                target_line = rand() % E;
                eviction_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);
}

/*
 * 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> -p <P> -t
        <file>\n", argv[0]);
    printf("Options:\n");
    printf("  -h         Print this help message.\n");
    printf("  -S <num>   Number of sets (s = log_2(S) is the
       number of bits for set index).\n");
    printf("  -E <num>   Number of lines per set (the
       associativity of the cache).\n");
    printf("  -B <num>   Number of bytes per line (b = log_2(B)
        is the number of bits for line offset).\n");
    printf("  -p <P>     Selects line replacement policy, LRU,
       FIFO or RAND.\n");
    printf("  -t <file>  Trace file.\n");
    printf("\nExamples:\n");
    printf("  linux>  %s -S 16 -E 1 -B 16 -p LRU -t traces/yi.
       trace\n", argv[0]);
    printf("  linux>  %s -v -S 256 -E 2 -b 16 -p FIFO -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:p:t:vh")) != -1){
        switch(c){
        case 'S':
            S = atoi(optarg);
            break;
        case 'E':
            E = atoi(optarg);
            break;
        case 'B':
            B = atoi(optarg);
            break;
        case 'p':
            policy = optarg;
        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 ||
       policy == NULL) {
        printf("%s: Missing required command line argument\n",
           argv[0]);
        printUsage(argv);
        exit(1);
    }
    /* Check replacement policy string and set code */
    if ( strcmp(policy, "LRU") == 0 ) policy_code = 1;
    else if (strcmp(policy, "FIFO") == 0 ) policy_code = 2;
    else if (strcmp(policy, "RAND") == 0 ) policy_code = 3;
    else {
        printf("%s: Line replacement policy invalid, use LRU,
           FIFO or RAND\n", argv[0]);
        printUsage(argv);
        exit(1);
    }

    /* Compute s and b from command line args */
    s = (unsigned int) log2(S);
    b = (unsigned int) log2(B);

    /* Initialize cache */
    initCache();

    replayTrace(trace_file);

    /* Free allocated memory */
    freeCache();

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



```
┌──(reynirjr㉿kali)-[~]
└─$ cd desktop
cd: no such file or directory: desktop

┌──(reynirjr㉿kali)-[~]
└─$ cd Desktop

┌──(reynirjr㉿kali)-[~/Desktop]
└─$ cd tolvutaekni

┌──(reynirjr㉿kali)-[~/Desktop/tolvutaekni]
└─$ cd verk2

┌──(reynirjr㉿kali)-[~/Desktop/tolvutaekni/verk2]
└─$  ./csim -S 16 -E 4 -B 64 -p LRU -t traces/yi.trace
hits: 6  misses: 3  evictions: 0
miss ratio: 33.33%

┌──(reynirjr㉿kali)-[~/Desktop/tolvutaekni/verk2]
└─$ ./csim-ref -S 16 -E 4 -B 64 -p LRU -t traces/yi.trace

hits:6 misses:3 evictions:0
miss ratio: 33.33%

┌──(reynirjr㉿kali)-[~/Desktop/tolvutaekni/verk2]
└─$ 
```

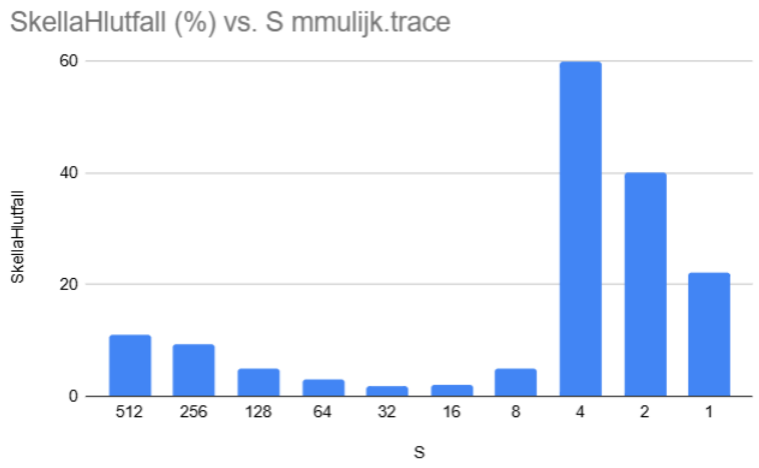
II

Besta uppsetning á skyndiminni

Við keyrslu á öllum þessum hermunum fékk ég fyrir mmulijk.trace:

Tafla 1: Gögn fyrir mmulijk.trace

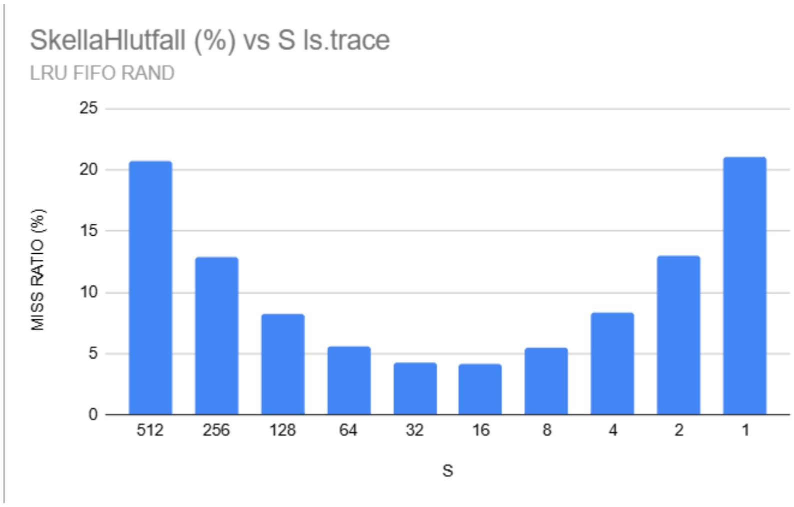
Trace File	Policy	B	S	Miss Ratio (%)	Hits	Misses	Evictions
traces/mmulijk.trace	LRU	8	512	4.41	567811	26209	22113
traces/mmulijk.trace	FIFO	8	512	5.50	561348	32672	28576
traces/mmulijk.trace	RAND	8	512	5.60	560777	33243	87441
traces/mmulijk.trace	LRU	16	256	2.72	577873	16147	14099
traces/mmulijk.trace	FIFO	16	256	3.28	574530	19490	17442
traces/mmulijk.trace	RAND	16	256	3.30	574416	19604	52668
traces/mmulijk.trace	LRU	32	128	1.47	585294	8726	7702
traces/mmulijk.trace	FIFO	32	128	1.80	583328	10692	9668
traces/mmulijk.trace	RAND	32	128	1.80	583307	10713	29067
traces/mmulijk.trace	LRU	64	64	0.83	589094	4926	4414
traces/mmulijk.trace	FIFO	64	64	1.03	587917	6103	5591
traces/mmulijk.trace	RAND	64	64	1.06	587723	6297	17355
traces/mmulijk.trace	LRU	128	32	0.48	591141	2879	2623
traces/mmulijk.trace	FIFO	128	32	0.63	590285	3735	3479
traces/mmulijk.trace	RAND	128	32	0.66	590079	3941	11055
traces/mmulijk.trace	LRU	256	16	0.58	590590	3430	3302
traces/mmulijk.trace	FIFO	256	16	0.81	589204	4816	4688
traces/mmulijk.trace	RAND	256	16	0.73	589659	4361	12699
traces/mmulijk.trace	LRU	512	8	1.36	585915	8105	8041
traces/mmulijk.trace	FIFO	512	8	1.53	584935	9085	9021
traces/mmulijk.trace	RAND	512	8	2.10	581533	12487	37269
traces/mmulijk.trace	LRU	1024	4	21.22	467957	126063	126031
traces/mmulijk.trace	FIFO	1024	4	22.67	459337	134683	134651
traces/mmulijk.trace	RAND	1024	4	16.05	498687	95333	285903
traces/mmulijk.trace	LRU	2048	2	12.90	517394	76626	76610
traces/mmulijk.trace	FIFO	2048	2	14.67	506901	87119	87103
traces/mmulijk.trace	RAND	2048	2	12.62	519047	74973	224871
traces/mmulijk.trace	LRU	4096	1	6.38	556107	37913	37905
traces/mmulijk.trace	FIFO	4096	1	8.20	545286	48734	48726
traces/mmulijk.trace	RAND	4096	1	7.52	549349	44671	133989



Við keyrslu á öllum þessum hermunum fékk ég fyrir ls.trace:

Tafla 2: ls.trace

Trace File	Policy	B	S	Miss Ratio (%)	Hits	Misses	Evictions
traces/ls.trace	LRU	8	512	6.66	306731	21879	17783
traces/ls.trace	FIFO	8	512	6.90	305935	22675	18579
traces/ls.trace	RAND	8	512	7.12	305205	23405	57927
traces/ls.trace	LRU	16	256	4.09	315184	13426	11378
traces/ls.trace	FIFO	16	256	4.30	314479	14131	12083
traces/ls.trace	RAND	16	256	4.50	313817	14793	38235
traces/ls.trace	LRU	32	128	2.54	320253	8357	7333
traces/ls.trace	FIFO	32	128	2.74	319619	8991	7967
traces/ls.trace	RAND	32	128	2.90	319092	9518	25482
traces/ls.trace	LRU	64	64	1.67	323106	5504	4992
traces/ls.trace	FIFO	64	64	1.88	322420	6190	5678
traces/ls.trace	RAND	64	64	2.04	321917	6693	18543
traces/ls.trace	LRU	128	32	1.21	324631	3979	3723
traces/ls.trace	FIFO	128	32	1.46	323808	4802	4546
traces/ls.trace	RAND	128	32	1.59	323377	5233	14931
traces/ls.trace	LRU	256	16	1.10	325008	3602	3474
traces/ls.trace	FIFO	256	16	1.55	323519	5091	4963
traces/ls.trace	RAND	256	16	1.54	323537	5073	14835
traces/ls.trace	LRU	512	8	1.39	324044	4566	4502
traces/ls.trace	FIFO	512	8	2.06	321853	6757	6693
traces/ls.trace	RAND	512	8	2.00	322036	6574	19530
traces/ls.trace	LRU	1024	4	2.32	320989	7621	7589
traces/ls.trace	FIFO	1024	4	3.11	318375	10235	10203
traces/ls.trace	RAND	1024	4	2.89	319097	9513	28443
traces/ls.trace	LRU	2048	2	3.60	316774	11836	11820
traces/ls.trace	FIFO	2048	2	4.72	313084	15526	15510
traces/ls.trace	RAND	2048	2	4.70	313155	15455	46317
traces/ls.trace	LRU	4096	1	5.74	309757	18853	18845
traces/ls.trace	FIFO	4096	1	7.34	304486	24124	24116
traces/ls.trace	RAND	4096	1	7.94	302505	26105	78291



i

Hver er munurinn á bestu uppsetningu á milli rakningaskráanna? Er hægt að skýra hann með því hvers konar forrit eru þar í keyrslu?

Skoðum gögnin og finnum smæstu skellaHlutföllin/ miss ratio

í "Best case scenario"

þá höfum við

- **mmulijk.trace** best í S = 32 með skellahlutfallið 0.48
- **ls.trace** best í S = 16 með skellahlutfallið 1.10

Við sjáum líka að mmulijk.trace er betra með fleirri mengi/S

Meðan ls.trace er betra í 16-32 S

mmulijk.trace er einnig með lægra skellahlutfall heldur en ls.trace

mmulijk.trace hefur háa spatial og minnisaðgangsmynsturið er reglulegt

ls.trace hefur minna af spatial locality og hefur óreglulegra minnisaðgangsmynstur

ii

Intel Core i7 hefur L1 skyndiminni sem er 8-vítt, 32KB að stærð með línustærð 64 bæti og 64 mengi. Hvernig kemur sú uppsetning út í hermununum? Ef hún er ekki best, hvers vegna skyldi Intel hafa valið hana?

úr Gögnunum kemur mmulijk.trace me' B= 64 og S = 64

POLICY	**SKELLAHLUTFAL**L (%)
LRU	0.83
FIFO	1.03
RAND	1.06

Úr ls.trace fengum við

POLICY	**SKELLAHLUTFALL** (%)
LRU	1.67
FIFO	1.88
RAND	2.04

samkvæmt mínum gögnum er intel uppsetningin ekki sú besta en hún er með frekar lága miss ratio í báðum tilfellum

Þetta er sennilega almenn málamiðlum milli þess að nýta spatial locality og halda latency lágri

Intel þarf að hanna skyndiminni sem skilar góðri frammistöðu fyrir mjög fjölbreytt urval af forritum Þannig intel valdi sennilega það sem hefur besta jafnvægið.