1 Project 5

Due: Dec 4, by 11:59p. **No extensions or late submissions.**

This document first provides the aims of this project. It then lists the requirements as explicitly as possible. It then hints at how these requirements can be met. Finally, it describes exactly what needs to be submitted.

1.1 Aims

The aims of this project are as follows:

- To allow you to simulate a read-only cache.
- To expose you to typical random number generation.

1.2 Requirements

Implement a module which implements a simulator for a read-only cache. Specifically, implement the following ADT:

```
/** Opaque implementation */
typedef struct CacheSimImpl CacheSim;

/** Replacement strategy */
typedef enum {
  LRU_R,          /** Least Recently Used */
  MRU_R,          /** Most Recently Used */
  RANDOM_R        /** Random replacement */
} Replacement;

/** A primary memory address */
typedef unsigned long MemAddr;

/** Parameters which specify a cache.
 *  Must have nMemAddrBits > nLineBits >= 2.
 */
typedef struct {
  unsigned nSetBits;        /** Text notation: s; # of sets is 2**this */
  unsigned nLinesPerSet;    /** Text notation: E; # of cache lines/set */
  unsigned nLineBits;       /** Text notation: b; # of bytes/line is 2**this
*/
  unsigned nMemAddrBits;    /** Text notation: m; # of bits in primary
```

```
mem addr;
                                    total primary addr space is 2**this */
  Replacement replacement; /** replacement strategy */
} CacheParams;


/** Create and return a new cache-simulation structure for a
 *  cache for main memory with the specified cache parameters params.
 *  No guarantee that *params is valid after this call.
 */
CacheSim *new_cache_sim(const CacheParams *params);

/** Free all resources used by cache-simulation structure *cache */
void free_cache_sim(CacheSim *cache);

typedef enum {
  CACHE_HIT,                    /** address found in cache */
  CACHE_MISS_WITHOUT_REPLACE,   /** address not found, no line replaced */
  CACHE_MISS_WITH_REPLACE,      /** address not found in cache, line replaced
*/
  CACHE_N_STATUS                /** # of status values possible */
} CacheStatus;

typedef struct {
  CacheStatus status;  /** status of requested address */
  MemAddr replaceAddr; /** address of replaced line if status is
                        *  CACHE_MISS_WITH_REPLACE */
} CacheResult;

/** Return result for addr requested from cache */
CacheResult cache_sim_result(CacheSim *cache, MemAddr addr);
```

Submit a `submit/prj5-sol` directory in your github repository such that typing `make` within that directory will build a `cache-sim` executable program. The program must start execution at the main program provided (see below) which drives your implementation of the above API.

To further clarify what must be returned by the `cache_sim_result()` function: The `status` should be set to

`CACHE_HIT` The data specified by the address is present in the cache

`CACHE_MISS_WITHOUT_REPLACE` The specified data is not present in the cache but when the data was read from main memory into the cache it was not necessary to replace any cache line because a free (invalid) cache line was available.

`CACHE_MISS_WITH_REPLACE` The specified data is not present in the cache and there was no free cache line so that when the data was read from main memory into the cache it was necessary to replace valid data stored in the cache.

In the last case, the return value must include `replaceAddr` which should be set to the address of the **block** which was replaced; i.e. `replaceAddr` must include the tag and set bits of the replaced address with the block offset bits set to 0.

1.3 The main Program

You are being provided with a main program to exercise your implementation of the above simulation API. When this program is run, it expects a required argument of the form *s*-*E*-*b*-*m* where:

s The number of address bits used to specify a cache set; the cache will contain a total of 2^s sets.

E The number of cache lines stored in each cache set;

b The number of address bits used to specify an offset within a cache line (block). The size of a cache block will be 2^b bytes.

m The total number of address bits used to specify a primary memory address. The total size of the main memory will be 2^m bytes.

Optionally, the main program accepts one or more of the following options before the above required argument:

`-r` ***REPLACEMENT*** The replacement strategy to be used by the cache simulator. Possible values are `lru` (for **least recently used**), `mru` (for **most recently used**) or `rand` (for **random replacement**). The default is `lru`.

`-s` ***SEED*** *SEED* must be non-negative integer specifying a seed for the random number generator. The default value is `0`.

`-v` If this option is specified, then the program produces additional verbose output.

You should ensure that when the program is invoked, the `$HOME/cs220/lib` directory will be searched for necessary libraries.

When run, the main program is set up to read whitespace-separated hexadecimal addresses from standard input and repeatedly call your `cache_sim_result(¬ )` function with those addresses. If the `-v` verbose option is specified, then it prints out the result of each call. Otherwise, it will merely print out a summary of the cache statistics when end-of-file is encountered on standard input.

1.4 Random Number Generation

Most programming environments provide a pseudo-random number generation facility. The facility is *pseudo* in that it does not generate real random numbers but only **pseudo** random numbers in that the generated sequence meets some statistical tests for randomness. In fact, to facilitate testing, the random sequence is repeatable and is controlled by a *seed* for the random number generation.

In the C programming environment, the random number facilities are available using two library functions `rand()` and `srand()` which are specified in the `<stdlib.h>` header file; successive calls to the former function return random non-negative integers; the seed of the random number generator can be specified by calling `srand()`. You will need to use the `rand()` function to simulate the random cache replacement strategy specified by the `-r rand` option.

1.5 The lib Directory

You may continue to use the following library contained within the course lib directory:

`libcs220`:: A trivial library which provides help with memory allocation and error reporting:

- Provides checked versions `mallocChk()`, `reallocChk()`, and `callocChk¬ ()` of the memory allocation routines which wrap the standard memory allocation routines with the program exiting on failure. The specification file is in memalloc.h.
- Provides routines for reporting errors using `printf()`-style format strings with one modification: if the format string ends with `:`, then `strerror¬ (errno)` is appended to the error-message. The specification file is in errors.h.

You may assume that the environment in which your program will be compiled and run will have this library available.

1.6 Provided Files

The prj5-sol directory contains the following files:

main.c This provides the `main()` program described above. You should not modify this file.

Cache Simulator Files The cache-sim.h file contains the specification for the API you are required to implement. You should not modify this file.

The cache-sim.c file will be the file where you do your work. As provided, it contains a skeleton implementation sufficient to compile the program without errors. You should remove the placeholder return values and flesh out the function definitions given there (adding in any auxiliary declarations or definitions which may be needed).

Makefile A `Makefile` for the project. Note that this `Makefile` is set up to include a reference to the `$HOME/cs220/lib` directory in the executable so that it is possible to run the executable without needing to set up `LD_LIBRARY_PATH`. You should not need to modify this file.

README A `README` template.

You must submit all the files in the above prj5-sol directory whether you modify them or not.

The aux-files directory contains the following files:

Test files lru_2-2-2-8.test and mru_4-2-8-16.test These files provides addresses which can be used for testing your program. The files contain the expected output as comments following `#` characters. The options to your program should be those corresponding to the name of the test file. You will need to strip out the `#` comments before feeding the file to your program; this can be done simply using a Unix utility like `sed`. For example, assuming that you are in the directory containing your executable:

```
$ sed -e 's/#.*//' $HOME/cs220/projects/prj5/aux-files/lru_2-
2-2-8.test \
        | ./cache-sim -r lru -v 2-2-2-8
```

If your simulator is working correctly, your output should match that shown within the test file.

A ruby script address-trace.rb This script can be used to generate address traces which can be piped into your program. It generates the following kind of traces:

`matrix` Addresses used to access a square matrix. The size of each matrix element is hardwired to be 8, but you can specify the size of the matrix as well as the access pattern (by row or by column).

`program` A typical program address trace. You can specify the probability of intra-procedure branches or inter-procedure calls/returns.

`random` A random address trace.

You can run the script with a `--help` option in general:

```
$ ./address-trace.rb --help
```

to get general usage information or on a specified subcommand:

```
$ ./address-trace.rb matrix --help
```

For example, to run your program with an address trace corresponding to row-access for a 1024x1024 matrix:

```
$ $HOME/cs220/projects/prj5/aux-files/address-trace.rb matrix \
      --access=row \
    | ./cache-sim -r lru -v 4-4-8-32 >t.log
```

To repeat, for column-access:

```
$ $HOME/cs220/projects/prj5/aux-files/address-trace.rb matrix \
      --access=col \
    | ./cache-sim -r lru -v 4-4-8-32 >t.log
```

You should not submit files from the `aux-files` directory.

1.7 Hints

Review the material covered in class and from the text pertaining to memory caches. Also, review online or other material to understand how to use the `rand()` function.

The following steps are merely suggestions:

1. Look at how you can implement the `CacheSimImpl` structure for your cache simulator. Note that your cache simulator is not required to track the cached data, merely to simulate the hit/miss behavior of a real cache. Hence it seems sufficient to merely:

   - For each cached line, merely track the address of the block it contains.
   - When the cache is empty, the contents of all cached lines will be invalid. As the cache fills up, these contents will become valid. Hence tracking the valid/invalid status of a cached line will be necessary.
   - For implementing the different replacement strategies, you will need to figure out how you will track the age of a cached line and/or organize them in a manner which facilitates the replacement strategy.
   - Since the cache parameters are only provided to the `new_cache_¬ sim()` function and you will need those parameters to simulate the cache in `cache_sim_result()`, the `CacheSimImpl` structure will also need to track the cache parameters provided to `new_cache_sim()`.

2. Since the cache parameters are available only at runtime (via the `CacheP-arams` argument), you will need to use dynamic memory allocation for the cache simulation.

   Abstractly, a cache consists of a collection of cache sets with each cache set consisting of a collection of cache lines; i.e. a cache is a collection of collections. Alternatives for representing collections in C include arrays, linked lists, etc.

If you go with using arrays for representing collections, you might wish to use multi-dimensional arrays. OTOH, if you set up ragged arrays (as described in the transparencies), you can still use multi-dimensional array notation with dynamic memory (at the cost of extra memory for pointers).

Specifically, you can always use dynamic allocation to get a pointer to a vector of elements and then use array indexing notation with the return'd pointer. For example, to emulate `int a[m][n]` using dynamically allocated ragged arrays, one can do:

```
int **a = malloc(m * sizeof(int *));
for (int i = 0; i < n; i++) {
  a[i] = malloc(n * sizeof(int));
}
//can now use array notation like a[i][j].
```

3. Start out your implementation without bothering about the replacement strategy; i.e. simply use any ad-hoc replacement strategy which is convenient for your code.
4. Once you have verified that your code works for accessing the correct set and cache line, implement the different replacement strategies. For each cache line you could track when it was last accessed (where time can be tracked using a simulation clock which is advanced for each step of the simulation). Alternately, you could track relative access age using ordering within the tracked information.

Iterate until you meet all requirements.

1.8 Submission

When you are happy with your project, move it over from your `work` directory to your `submit` directory:

```
$ cd ~/i220X #X is either a or b
$ git mv work/prj5-sol submit
$ git commit -a -m 'suitable comment'
$ git push
```

This should submit your project as `submit/prj5-sol` via github. Your submission should not include any object files or executables; this will be prevented by the provided *.gitignore* file.

If submitting late, please drop an email to the TA for your section:

Section A yli241@binghamton.edu

Section B rrausha1@binghamton.edu