Lab #3. Cache Lab

Prof. Jaeseung Choi

Dept. of Computer Science and Engineering

Sogang University

About this lab

- **In this lab, you have to write a C program that simulates cache memory access**
  - Configuration parameters of cache will be given as input
  - Sequence of memory addresses to be accessed will be also provided as input
  - Your program must decide whether each access will be a **cache hit** or **cache miss**
- **To do this assignment, you must clearly understand the operation of cache memory**
  - Also, it is a good chance to practice programming

Remind: Cheating Policy

- **Cheating in assignment will give you a serious penalty**
  - Your final grade will be downgraded (e.g., from ***B+*** to ***C+***)
- **Scope of cheating in assignment**
  - Copying the code of other people
  - Sharing your solution with others
  - Asking ChatGPT to write your code
  - Discussing with others how to solve the problem

General Information

- **Check the *Assignment* tab of *Cyber Campus***
  - Skeleton code (`Lab3.tgz`) is attached together with this slide
  - Submission will be accepted in the same post, too
- **Deadline: 6/25 Wednesday 23:59**
  - **No late submission for this Lab!**
- **Please read the instructions in this slide carefully**
  - This slide is a step-by-step tutorial for the lab
  - It also contains important submission guidelines
    - If you do not follow the guidelines, **you will get penalty**

Skeleton Code Structure

- **Copy `Lab3.tgz` into CSPRO server and decompress it**
  - Recommend to use cspro**2**.sogang.ac.kr
  - Don’t decompress-and-copy; copy-and-decompress
- **`3-1` & `3-2`: There are two sub-problems in this lab**
  - `3-1` : Implementing single-level cache
  - `3-2` : Extending implementation to multi-level cache
- **`check.py`: Script for self-grading (explained later)**
- **`config`: Used by grading script (you may ignore)**

```
jschoi@cspro2:~$ tar -xzf Lab3.tgz
jschoi@cspro2:~$ ls Lab3
3-1  3-2  check.py  config
```

3-1 Directory: Single-level Cache

- `Makefile`: **Already given for you (just type `make` to build)**
- `types.h`: **Definition of basic types for cache memory**
- `single_cache.h`: **Structure definition and function prototype for single-level cache simulator**
- `single_cache.c`: **The actual cache simulator logic that you must implement (the only file that you have to fix)**
- `main.c`: **Program to run and test your simulator code**

```
jschoi@cspro2:~/Lab3$ cd 3-1
jschoi@cspro2:~/Lab3/3-1$ ls
Makefile   single_cache.c   testcase
main.c     single_cache.h   types.h
```

Input of Cache Simulator

- **Let's first check the format of inputs and outputs**
- **`config-N`, `trace-N` are the inputs of the cache simulator**
  - **`config-*`** contains the values of **b**, **E**, and **s** (in this order)
    - Assume that 0 < **b** <= 16, 0 < **E**, 0 < **s** <= 16
  - **`trace-*`** contains the sequence of memory addresses to be accessed (in hexadecimal format)
    - Assume that all accesses are **1-byte access**

```
jschoi@cspro2:~/Lab3/3-1$ cat testcase/config-1
2 1 4
jschoi@cspro2:~/Lab3/3-1$ cat testcase/trace-1
BA00
BA04
...
```

Output of Cache Simulator

- **Your compiled program must decide whether each memory access will result in cache hit or cache miss**
  - Ex) The simulator below is saying that the first three accesses (**`BA00`**, **`BA04`**, **`...`**) are **misses** and the next one is a **hit**
- **`ans-N` is the expected output for `config-N` & `trace-N`**

```
jschoi@cspro2:~/Lab3/3-1$ make
gcc -c single_cache.c
gcc single_cache.o main.c -o main.bin
jschoi@cspro2:~/Lab3/3-1$ ./main.bin \
                            testcase/config-1 testcase/trace-1
 MISS
 MISS
 MISS
 HIT
 ...
```

Must match with the content of **`ans-1`** file

Driver Code (`main.c`)

- **The `main.c` file in skeleton code is already doing many things for the cache simulator**
  - It calls `init_single_cache_from_file()` to allocate and initialize `single_cache_t` structure based on the `config-N` file
  - Then, it iteratively reads in each line of `trace-N` and calls `access_with_single_cache()` to decide hit vs. miss
- **Before you proceed, take enough time to carefully read and understand the overall flow of the code**

Tasks to do (for 3-1)

- **You have to implement two functions in 3-1**
  - Read the type definitions and comments in `single_cache.h` and implement your code in `single_cache.c`
  - Note that `init_single_cache_from_file()` is already implemented to call `init_single_cache_with_param()`
- **You must implement LRU policy for the cache memory**

```
// Allocate and initialize a 'single_cache_t' structure with
// the provided cache parameters. (...)
single_cache_t* init_single_cache_with_param(int b, int E ...);

// Simulate memory access on address 'addr'. Return 0 if the
// access results in a hit and -1 if it raises a miss. (...)
int access_with_single_cache(single_cache_t* cache,
                             addr_t addr);
```

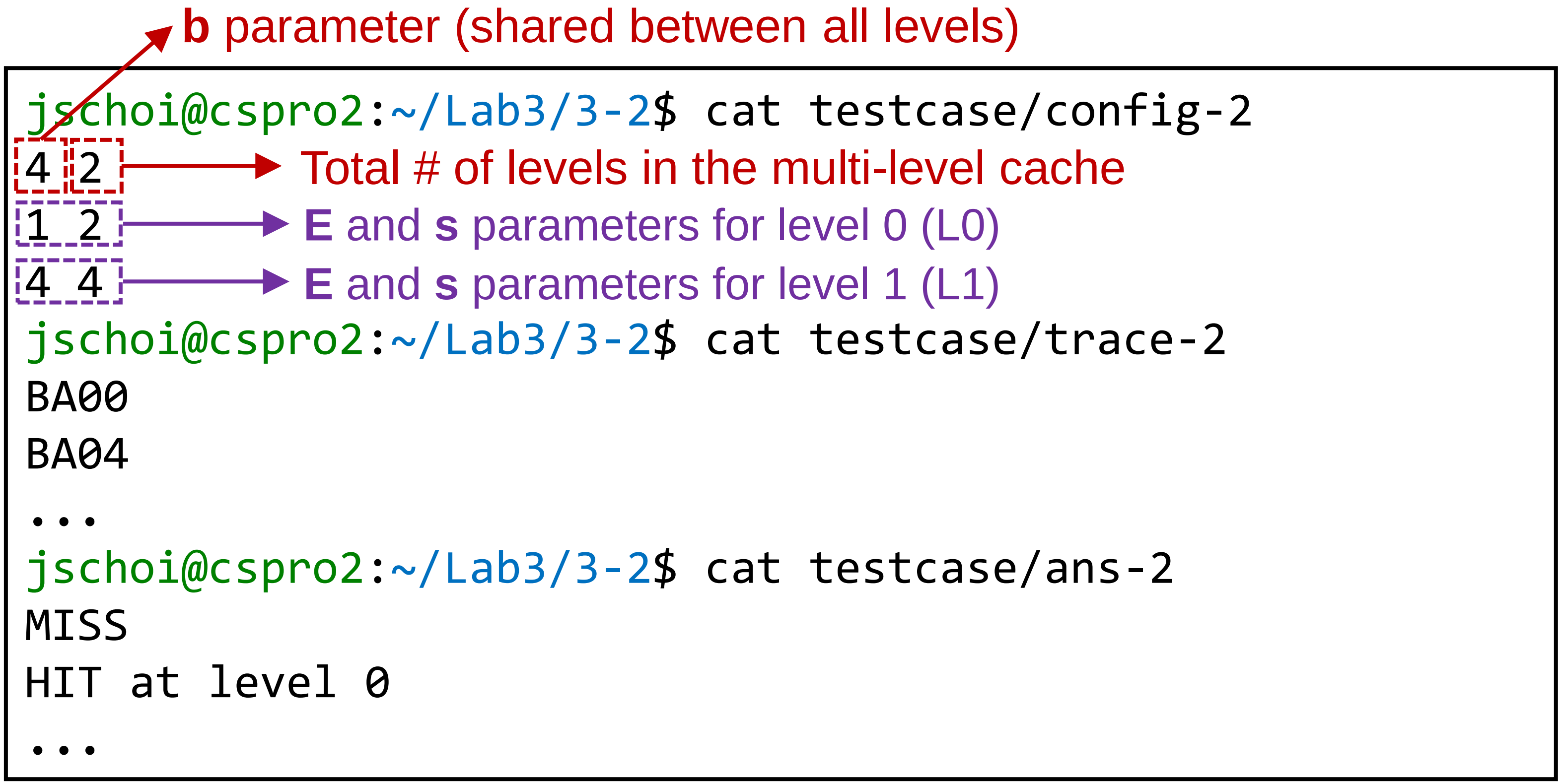
3-2 Directory: Multi-level Cache

- **Most files have the same role**
- **single_cache.c: You must *copy your completed code* from the 3-1 directory**
- **multi_cache.h: Structure definition and function prototype for *multi-level* cache simulator**
- **multi_cache.c: You must implement the logic here**
- **main.c: Program to run and test your simulator code (updated to handle the multi-level cache simulation)**

```
jschoi@cspro2:~/Lab3$ cd 3-2
jschoi@cspro2:~/Lab3/3-2$ ls
Makefile    multi_cache.c   single_cache.c   testcase
main.c      multi_cache.h   single_cache.h   types.h
```

Input and Output of 3-2

- **Note that the cache simulator now reports the level at which the cache hits have occurred**
  - Read **`main.c`** to check the changes



```
jschoi@cspro2:~/Lab3/3-2$ cat testcase/config-2
4 2
1 2
4 4
jschoi@cspro2:~/Lab3/3-2$ cat testcase/trace-2
BA00
BA04
...
jschoi@cspro2:~/Lab3/3-2$ cat testcase/ans-2
MISS
HIT at level 0
...
```

b parameter (shared between all levels)

Total # of levels in the multi-level cache

E and **s** parameters for level 0 (L0)

E and **s** parameters for level 1 (L1)

Tasks to do (for 3-2)

- **Again, you have to implement two functions in 3-2**
  - Read the type definitions and comments in **multi_cache.h** and implement your code in **multi_cache.c**
- **You can easily do this by *reusing the functions* for single-level cache memory you implemented in 3-1**
  - This is an important principle in software development - reusing small building blocks to construct a larger system

```
// Allocate and initialize a 'multi_cache_t' structure based
// on the provided cache configuration file. (...)
multi_cache_t* init_multi_cache_from_file(char *config_path);

// Simulate memory access on address 'addr'. Return -1 if the
// access eventually raised a cache miss. (...)
int access_with_multi_cache(multi_cache_t* cache, addr_t addr);
```

Self-Grading

- **Once you think everything is done, run `check.py` to confirm that you pass all the provided test cases**
  - Each character in the result has following meaning:

    'O': Correct output

    'X': Wrong output

    'C': Compile error

    'T': Timeout

    'E': Runtime error or non-zero exit code (e.g., **`exit(1);`**)
  - So you must make **`./check.py`** print '**O**' for all the cases

```
jschoi@cspro2:~/Lab3$ $ ./check.py
[*] 3-1 : OOOO
[*] 3-2 : OO
```

Problem Information

- **There are two problems in total**
  - Problem **`3-1`**: **80 pt.**
  - Problem **`3-2`**: **20 pt.**
  - As the score distribution indicates, correctly implementing **`3-1`** is more important and challenging
- **You will get the point for each problem based on the number of test cases that your code passes**
  - Recall that I will use a different test case set for actual grading
  - In this lab, it is especially important to **think of your own inputs** to test your code (you must practice doing this)
  - Passing the basic test cases provided in the skeleton code does not guarantee that your code is correct, so be careful

Submission Guideline

- **You should submit the following two C source files (be careful to not submit the header files)**
  - **`single_cache.c`**
  - **`multi_cache.c`**
  - **`3-1`** will be graded with **`single_cache.c`** and **`3-2`** will be graded by using **`single_cache.c`** + **`multi_cache.c`**
- **You must make sure that the submitted files correctly compile when I type "`make`" command**
- **Submission format**
  - Upload these files directly to *Cyber Campus* (**do not zip them**)
  - **Do not change the file name** (e.g., adding any prefix or suffix)
  - If your submission format is wrong, you will get **-20% penalty**