Lab #3. Cache Lab

Prof. Jaeseung Choi

Dept. of Computer Science and Engineering

Sogang University

Change of Plan: Labs

- **Labs will count for 30% of the total score in semester**
- **We have done three lab assignments so far**
  - **Lab #1:** Bit Lab (6%)
  - **Lab #2:** Reversing Lab (12%)
  - **Lab #3: Cache Lab (12% ⇒ 8%)**
    - I fixed the problem to make it easier than my original plan
    - And its weight in the total score has been slightly reduced
    - Remaining **4% point** will be given as free bonus point

About this lab

- **In this lab, you have to write a C program that simulates cache memory access**
  - The cache parameters (e.g., # of lines) will be given as input
  - For the provided memory access sequence, 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 implementing certain idea into concrete code

Remind: Cheating Policy

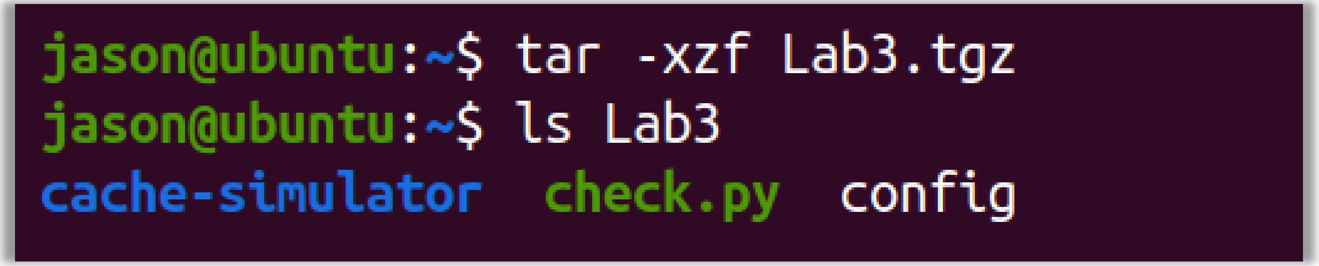
- **Cheating in assignment will give you a serious penalty**
  - Your final grade will be downgraded (e.g., from ***B+*** to ***C+***)
- **Scope of cheating in assignment**
  - Copying the code of other people
  - Sharing your solution with others
  - Asking ChatGPT to write your code
  - Discussing 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: 6/26 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

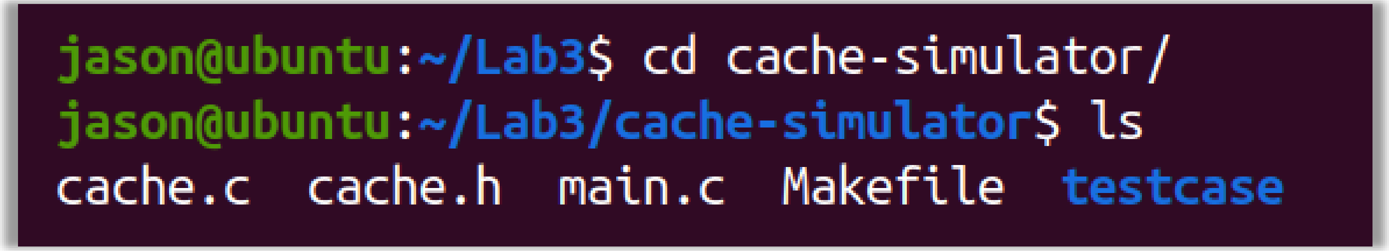
- **Copy `Lab3.tgz` into CSPRO server and decompress it**
  - Recommend to use cspro**2**.sogang.ac.kr (**Ubuntu 20.04**)
  - Don't decompress-and-copy; copy-and-decompress
- **`cache-simulator`: In Lab #3, this is the only problem directory that you have to care about**
- **`check.py`: Script for self-grading (explained later)**
- **`config`: Used by grading script (you may ignore)**



```
jason@ubuntu:~$ tar -xzf Lab3.tgz
jason@ubuntu:~$ ls Lab3
cache-simulator  check.py  config
```

Problem Directory

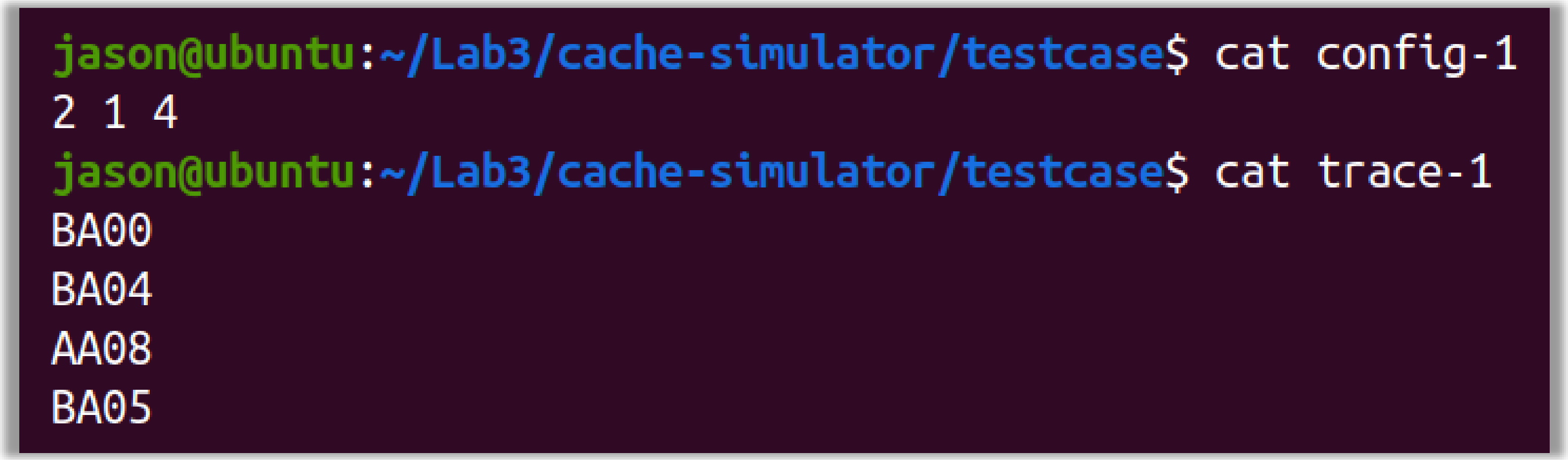
- `cache.h`: Provides type definitions and function declarations for the cache simulator
- `cache.c`: The actual cache simulator logic that you have to fill in (do NOT modify any other file)
- `main.c`: Program to run and test your simulator code
- `Makefile`: Already given for you (just type `make` to build)
- `testcase`: Contains test cases and expected outputs



```
jason@ubuntu:~/Lab3$ cd cache-simulator/
jason@ubuntu:~/Lab3/cache-simulator$ ls
cache.c  cache.h  main.c  Makefile  testcase
```

Input of Cache Simulator

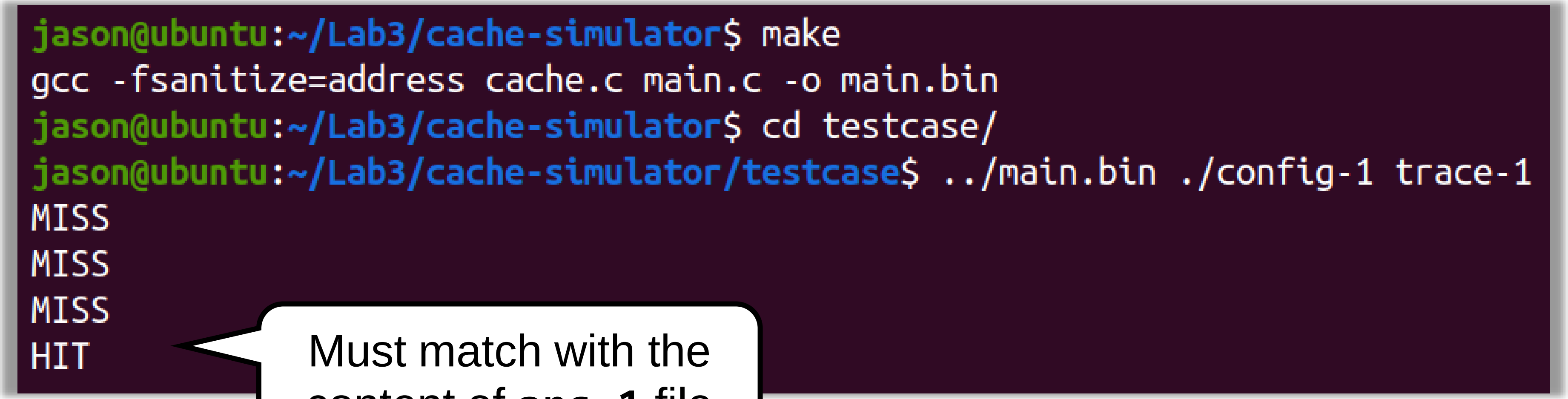
- **Let's first check the format of inputs and outputs**
- `config-N`, `trace-N` **are the inputs of the cache simulator**
  - **`config-*`** contains the cache parameters **b**, **E**, and **s** (be careful on the **lower/upper case**)
    - Assume that 0 < **b** <= 16, 0 < **E**, 0 < **s** <= 16
  - **`trace-*`** contains a sequence of memory addresses to access
    - Assume that all accesses are **1-byte access**



```
jason@ubuntu:~/Lab3/cache-simulator/testcase$ cat config-1
2 1 4
jason@ubuntu:~/Lab3/cache-simulator/testcase$ cat trace-1
BA00
BA04
AA08
BA05
```

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`**, **`AA08`**) are **miss** and the next one (**`BA05`**) is **hit**
- **`ans-N` is the expected output for `config-N` & `trace-N`**



```
jason@ubuntu:~/Lab3/cache-simulator$ make
gcc -fsanitize=address cache.c main.c -o main.bin
jason@ubuntu:~/Lab3/cache-simulator$ cd testcase/
jason@ubuntu:~/Lab3/cache-simulator/testcase$ ../main.bin ./config-1 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 reads in the cache parameters (**b**, **E**, and **s**) from `config-N` and initialize the cache structure by calling **`init_cache()`**
  - Then, it iteratively reads in each line of `trace-N` and calls **`access_memory()`** to decide whether it's cache hit or miss
  - Lastly, it calls **`free_cache()`** to clean up the heap memory
- **You must first take enough time to carefully read and understand the code in `main.c`**

Tasks to do

- **You have to implement the three functions below**
  - Read the type definitions and comments in `cache.h` carefully
- **Depending on the value of cache parameters, your cache can be direct-mapped or set/fully associative**
  - And your cache must implement **LRU policy** for eviction

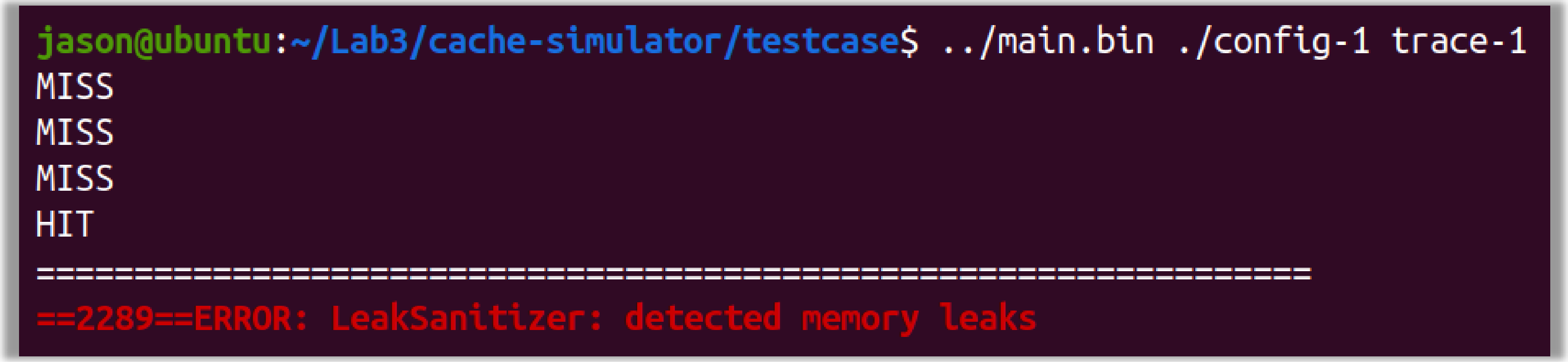
```
// Allocate and initialize cache_t structure. (...)
cache_t* init_cache(int b, int E, int s);

// Simulate memory access on address 'addr'. Return 1 if
// the access is a cache hit and 0 if it's a miss. (...)
int access_memory(cache_t* cache, addr_t addr);

// Free the dynamically allocated memory in 'cache'. (...)
void free_cache(cache_t* cache);
```

What is this error message?

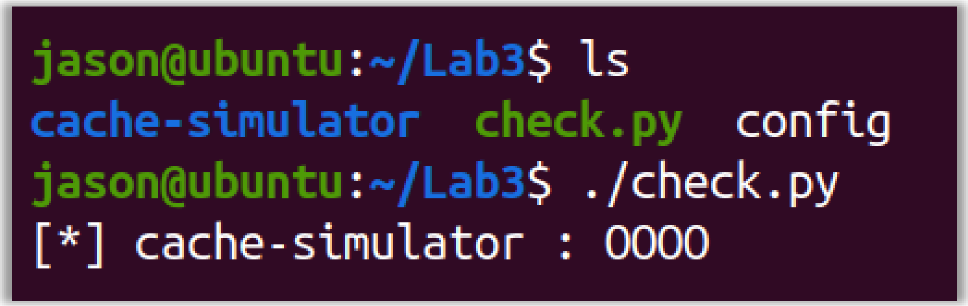
- **You may encounter this kind of error message when you run the compiled `main.bin` program**
- **In our `Makefile`, I used some special compiler flag when building the `main.bin` binary**
  - As shown below, it will report buffer overflows or memory leaks (unallocated memory when **`main()`** returns) during the runtime



```
jason@ubuntu:~/Lab3/cache-simulator/testcase$ ../main.bin ./config-1 trace-1
MISS
MISS
MISS
HIT
=================================================================
==2289==ERROR: LeakSanitizer: detected memory leaks
```

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, `'X'`: wrong, `'C'`: compile error,
    `'T'`: timeout, `'E'`: runtime error
- **You'll get the point based on the number of test cases that your code passes (100 points in total)**



```
jason@ubuntu:~/Lab3$ ls
cache-simulator  check.py  config
jason@ubuntu:~/Lab3$ ./check.py
[*] cache-simulator : OOOO
```

Submission Guideline

- **You should submit only one C source file**
  - `cache.c` (be careful not to submit `cache.h`)
- **If the submitted file does not compile by typing "`make`" command, cannot give you any point for that problem**
- **Submission format**
  - Upload this file directly to *Cyber Campus* (**do not zip it**)
  - **Do not change the file name** (e.g., adding any prefix or suffix)
  - If your submission format is wrong, you will get **-20% penalty**