Programming Assignment #3

Operating Systems (SCE213)

Younghoon Kim
(yhoon@ajou.ac.kr)

PA3

- Implement and analyze a simplified virtual memory management system supporting multi-level paging(3-level) and TLB.

1. **Round-Robin-based Simulator (40 points)**

2. **LRU-based Simulator & Experimental Report (60 points)**
    - **This part includes the testcase generator and the graph generation code**

PA3: step-by-step

1. Implement a Round-Robin-based Simulator
2. Extend Round-Robin-based Simulator into LRU-based Simulator
3. Implement input testcase generator
4. Test LRU-based Simulator using generated input testcases
5. Analyze the experimental results.

Simulator-Common

- Your simulator must be implemented in C only.
  - You do **not** need to put all of your code into a single .c file — you may organize it into multiple .c and .h files as you prefer.
  - However, you **must include a** Makefile for compilation, and the **compilation procedure** should be clearly described in your report.

- Your simulator must accept the following command-line options:
  - **-p**: specifies the replacement **policy** (RR or LRU)
  - **-f**: specifies the **input test case file**
  - **-l**: specifies the **output log file** to save the results.

- It must work correctly when executed exactly as shown in the following example.
  - "./simulator -p RR -f ./input_1 -l ./output_1" *or* "./simulator -p LRU -f ./input_1 -l ./output_1"

1. Round-Robin-based Simulator

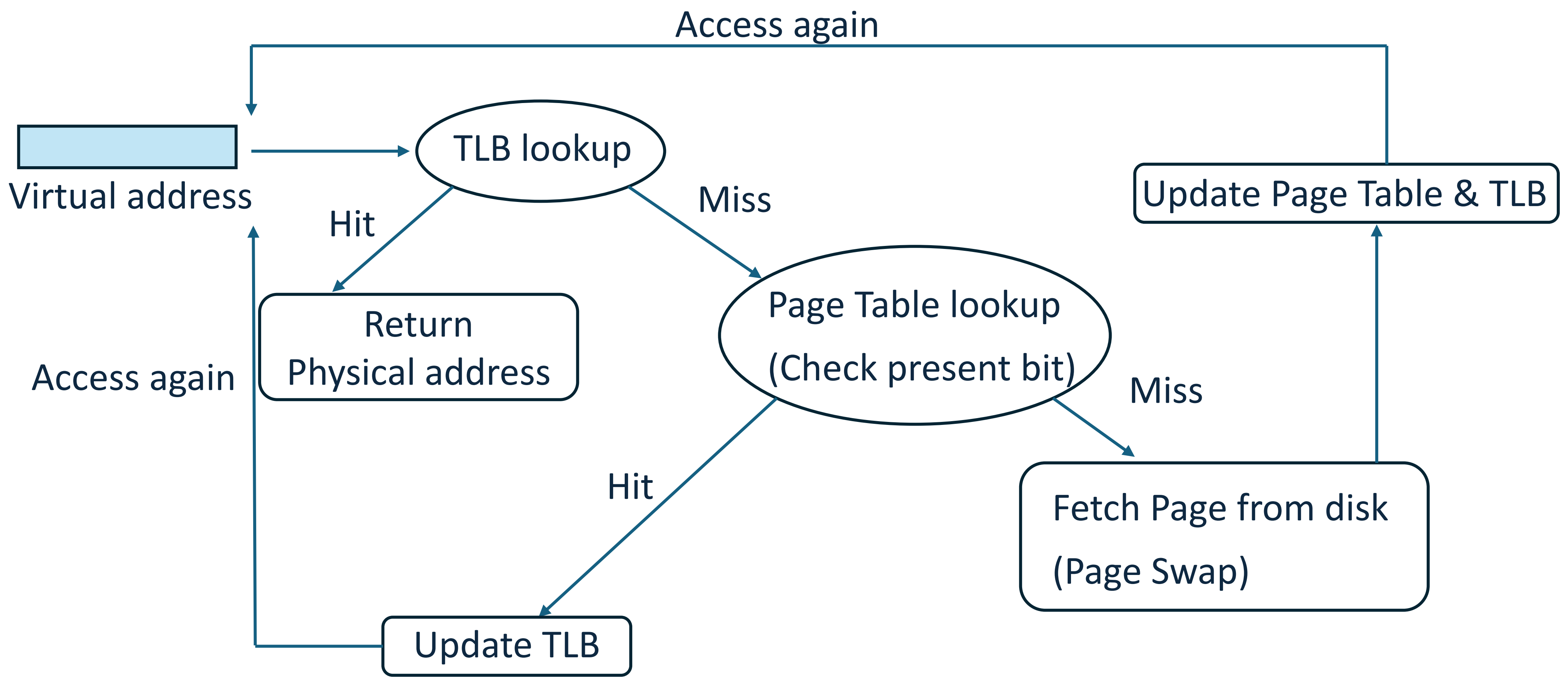
- Implement a memory simulator where both TLB replacement and page swap (eviction) follow the Round-Robin policy.
  - This simulator should support 3-level paging **(slides 6)**

- The simulator should produce output logs by utilizing the provided logging functions.

- Your implementation will be automatically graded using the input–output pairs.

1. Round-Robin-based Simulator

- Here is the basic flow of the simulator.



1. Round-Robin-based Simulator

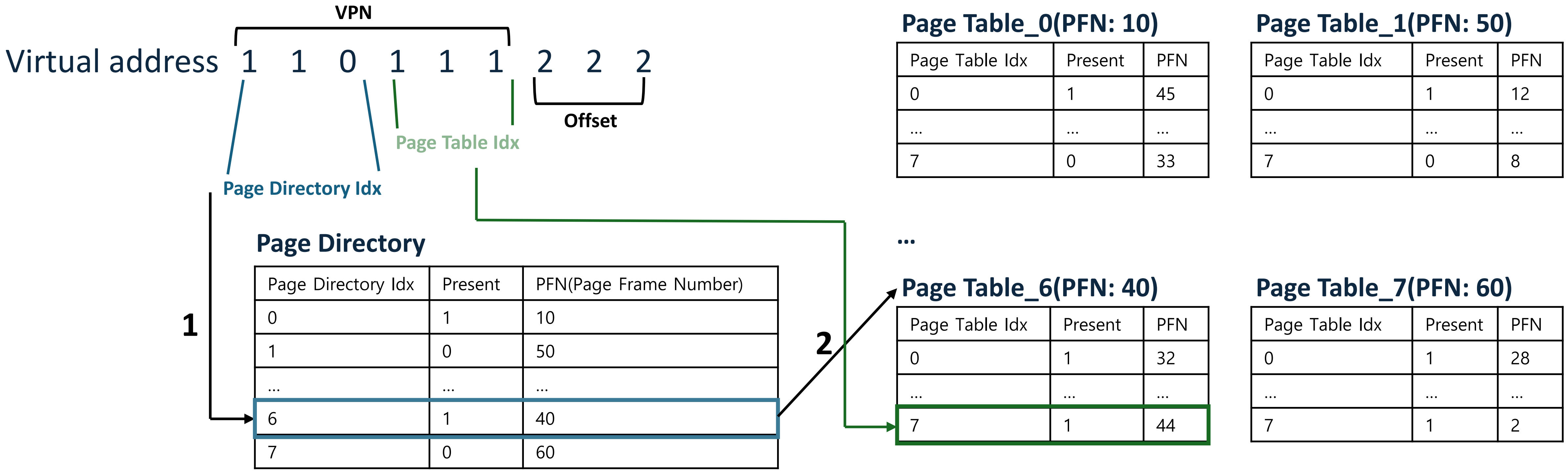
- A multi-level page table is a hierarchical structure used in virtual memory systems to reduce the memory overhead of storing page tables.

- Instead of keeping a single large page table, it breaks the page table into multiple page-sized units.
  - In this simulator, the size of the page table and page are equal.

- To track the page table and get PFN, it use “page directory”
  - Page directory is also a page table.
  - Using page directory, you can get the PFN of page table.

1. Round-Robin-based Simulator

- Page directory is also page table which indicate the PFN of another page table.
- Here is an example of how multi-level page table works.(2-level)
  - This is just a simple example and may not reflect a logically accurate design.



Page Directory

Page Directory Idx	Present	PFN(Page Frame Number)
0	1	10
1	0	50
…	…	…
6	1	40
7	0	60

Page Table_0(PFN: 10)

Page Table Idx	Present	PFN
0	1	45
…	…	…
7	0	33

Page Table_1(PFN: 50)

Page Table Idx	Present	PFN
0	1	12
…	…	…
7	0	8

Page Table_6(PFN: 40)

Page Table Idx	Present	PFN
0	1	32
…	…	…
7	1	44

Page Table_7(PFN: 60)

Page Table Idx	Present	PFN
0	1	28
…	…	…
7	1	2

1. Round-Robin-based Simulator

- This simulator use a 12-bit address space.
- TLB has total 16 entries.
- The size of physical memory(main memory) is 1024Bytes
- The size of a page(page-size) and frame is 8Bytes.
- The size of a page table entry(PTE) is 1Byte.
- The simulator use 3-level page table.
- The size of page table?
- The number of PTE per page table?
- The number of bit of offset and VPN?
- The number of bit of page directory index?

Calculate yourself using above information!

1. Round-Robin-based Simulator

- Page Table Entry(PTE) is 1Byte.
  - Both consists of a 1-bit “present” field and a 7-bit “page frame number(PFN)”
  - “present” bit indicates whether the page is in main memory or not

- In this simulator, suppose there is only single process.
  - Do not consider context switch.

1. Round-Robin-based Simulator

- "Update TLB" means
  - Overwrite exist entry with new entry.
  - The new entry position is determined by Round-Robin (Start from 0).
- "Swap page" means
  - If main memory is full, swap out one page from main memory and swap in new page from disk to main memory.
  - If main memory is not full, swap in new page from disk to main memory.
  - The swapped-out page is determined in Round-Robin manner (Start from 0).
  - If a page is swapped out, its "present" bit must be set to 0 in both the TLB and the Page Table.

1. Round-Robin-based Simulator

- "Update Page Table" means
  - All of the page table(page directory) is resided in main memory.
    - So, if the main memory is full, page swap should be occurred.
    - Pages the contain page tables are never swapped out.
      - ✓ If the eviction policy select such a page, the simulator skips it and continues searching until it finds a page that can be evicted.
  - Finally, insert new entry to page table
    - Page table entries are never evicted from the page table.

1. Round-Robin-based Simulator

- All of addresses should be result in TLB hit.
- If TLB hit, nothing to do.
- If TLB miss but hit in Page Table, the update TLB and the address is accessed again.
  - » So, when the same address is accessed again, a TLB hit occurs
- If both the TLB and page table miss, swap the page and update Page Table and TLB. Then the address is accessed again.
  - » So, when the same address is accessed again, a TLB hit occurs

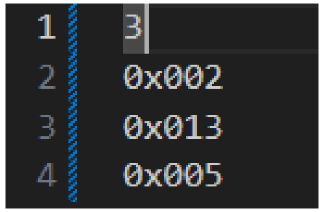
1. Round-Robin-based Simulator

- In a 3-level page table, not all page tables and page directories exist from the beginning.
- They are dynamically allocated as memory accesses occur.
- Until this, the simulator closely resembles a real system.

  However, there is one key difference:
  - In a real system, a page table might be deallocated when all of its entries are invalid.
  - But in this simulator, it keeps in main memory.(Deallocation is not happened)

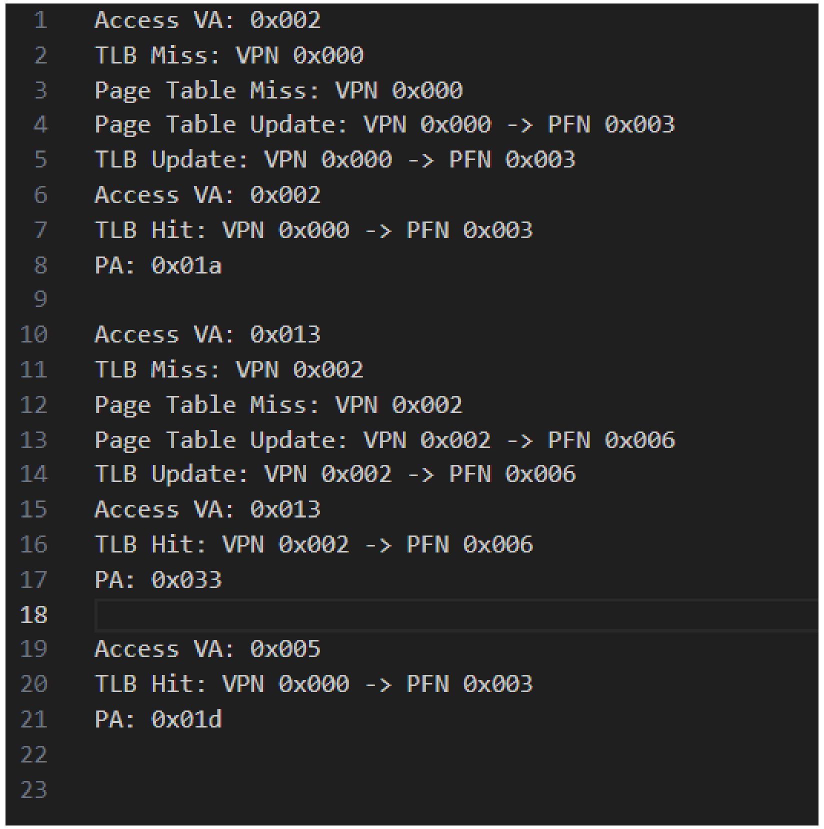
Example input/output

- Input



```
3
0x002
0x013
0x005
```

- Output



```
Access VA: 0x002
TLB Miss: VPN 0x000
Page Table Miss: VPN 0x000
Page Table Update: VPN 0x000 -> PFN 0x003
TLB Update: VPN 0x000 -> PFN 0x003
Access VA: 0x002
TLB Hit: VPN 0x000 -> PFN 0x003
PA: 0x01a

Access VA: 0x013
TLB Miss: VPN 0x002
Page Table Miss: VPN 0x002
Page Table Update: VPN 0x002 -> PFN 0x006
TLB Update: VPN 0x002 -> PFN 0x006
Access VA: 0x013
TLB Hit: VPN 0x002 -> PFN 0x006
PA: 0x033

Access VA: 0x005
TLB Hit: VPN 0x000 -> PFN 0x003
PA: 0x01d
```

- Explanation

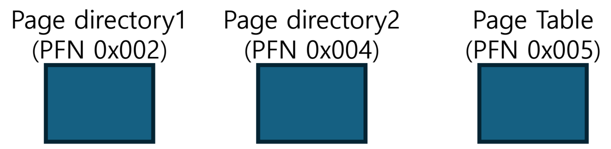
1. Try to access 0x002
2. Both TLB and page table miss occur
3. Get free frame

   - 0x003 because 0x000 and 0x001 are used for bitmask and 0x002 for Page directory1(Root page table)

4. Update page table entry and TLB entry
5. When try to access 0x002 again, then TLB hit

6. Access 0x013 is handled in the same way as 0x002.
7. But here, free frame is 0x006

   - Because of 3-level page table



8. Regarding 0x005, it has same VPN with 0x002. So, TLB hit occur.

2. LRU-based Simulator

- Based on the simulator implemented in Part 1, extend it to support the LRU (Least Recently Used) replacement policy for both TLB replacement and page swapping (eviction).

- No reference output is provided; this version will be used for further analysis.

3. Testcase Generator

- Implement a testcase generator that produces memory access traces to be used as input for the simulator:
  - Uniform distribution
  - Zipfian (skewed) distribution

- You can implement generator in any programming language.

3. Testcase Generator

- **A uniform distribution** assigns **equal probability** to each item: every page is accessed with the same probability.
- For items (pages) $k = 1, \dots, N$, the probability is:
  - $P(X = k) = \frac{1}{N}$

- **Parameters you must consider**
  - N: number of distinct pages (Calculate yourself).

3. Testcase Generator

- A **zipfian** distribution models **heavy-tailed popularity**: a few items are accessed very frequently, while many are rarely accessed.
- For ranks $k = 1, \dots, N$ (1 = most popular), the probability is:
  - $P(X = k) = \frac{1/k^{s}}{H_{N,s}}, \quad H_{N,s} = \sum_{i=1}^{N} \frac{1}{i^{s}}$
  - https://en.wikipedia.org/wiki/Zipf%27s_law

- **Parameters you must consider**
  - N: number of distinct pages (Calculate yourself).
  - s (also called **α**, **a**, **exponent**, or **skew**): controls how skewed the popularity is.
    - Larger $s \Rightarrow$ **more skew** (top pages dominate).

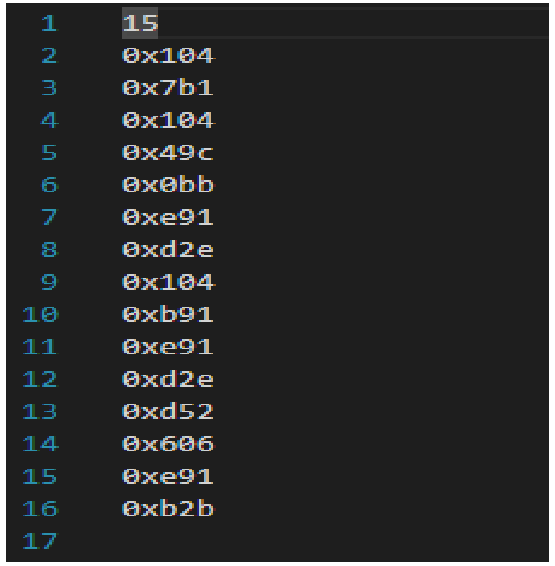
3. Testcase Generator

- The **testcase generator** must accept two command line options:
  - **--dist** or **-d**: specifies the distribution type — either "**uniform**" or "**zipfian**".
  - **--s** or **-s**: specifies the **skew parameter** when using the **zipfian** distribution.
    - $ ./testcase_generator -d uniform
    - $ python3 testcase_generator.py --dist zipfian --s 1.5

- A clear and detailed **user guide** explaining how to execute the **testcase generator** must be **included in the report**.

3. Testcase Generator

- The format of the input test case is as follows:
  - The **first line** specifies the total number of memory accesses.
  - Starting from the **second line**, each line contains a single accessed address in **hexadecimal format**.
  - The **final line** is an **empty newline**.



```
15
0x104
0x7b1
0x104
0x49c
0x0bb
0xe91
0xd2e
0x104
0xb91
0xe91
0xd2e
0xd52
0x606
0xe91
0xb2b

```

4. Test LRU-based Simulator using generated testcases.

- Run experiments using the **LRU-based simulator** with each of the following input distributions
  - **Uniform distribution**
  - **Zipfian** with $s = 0.5, 1.0,$ and 1.5
  - Each input contains a total of 10,000 memory accesses.

- The execution results must follow the same logging format and method used in the previous FIFO-based simulator.

5. Analyze the experimental results

- You are required to write a **summary report** of your experimental results.
  - You must implement a **tool** that generates **meaningful graphs** based on the **log files produced by the simulator**, and include the resulting graphs in your **report**.

- The **required contents** of the report will be explained in the **following slides**.

Report

- There is **no fixed template** for the report; however, it must include the following components:

1. **Personal information** (student ID, name) and a description of your **experimental environment** (e.g., operating system, compiler version, etc.).

2. **Instructions** on how to compile and run:
   - **simulator, testcase generator** and **graph generation tool**.

Report

3. **Graphs** illustrating the **page access frequency** for each generated test case (four graphs in total).
   - Explanation of how the **testcase generator** works.

4. **Graphs** comparing the **LRU-based simulator results** across test cases, including metrics such as **TLB miss rate** and **page fault rate**.
   - A brief description of **which log information** was used and **how** it was utilized to generate the graphs.
   - In these calculations, re-accessed addresses should be excluded.

5. **Your analysis and discussion** of the results — what they imply and **why** you think these outcomes occurred.

Report

- There is no fixed format for the **graphs**, but they should be designed to ensure high visibility and allow readers to easily grasp the intended message.
  - Consider which type of graph would most effectively convey your findings.

- The report must be written in English or Korean and should be two pages in length (Do not over).

- You should submit your report in PDF format.

Grading

1. **Round-Robin-based Simulator (40 points)**
   - We will evaluate your simulator using **10 hidden testcases**, and assign scores based on correctness (**4 points each**).
2. **LRU-based Simulator & Experimental Report (60 points)**
   - **Testcase Generation & Visualization (20 pts)**
     - A clear explanation is provided on **how the testcase generator works** and how the testcases were produced.
     - Page access frequency is clearly visualized for all four testcases with proper labeling and scales.
   - **LRU Simulation Results & Visualization (20 pts)**
     - A brief description is included on **which log information was used** and **how it was processed to generate the graphs**.
     - Results such as TLB miss ratio and page hit/miss ratio are accurately plotted and clearly presented.
   - **Analysis & Discussion (20 pts)**
     - Explanations effectively describe **why** certain distributions yield different outcomes.

Assignment Policy

- Your simulator must be implemented in C only.
  - You do **not** need to put all of your code into a single .c file — you may organize it into multiple .c and .h files as you prefer.
- You may use any programming language to implement the testcase generator and generate graphs for the report.
  - However, simply using tools such as Excel or Google Sheets is not allowed.
- Grading of the **Round-Robin-based simulator** will be performed **solely by comparing the generated log files**.
  - You must produce logs **exactly** according to the **provided logging functions and examples**.
  - In this evaluation, the number of input addresses will **not exceed 150**.

Assignment Policy

- To ensure that your simulator runs correctly in the TA’s environment, you must include a Makefile and provide a detailed description of the compilation process in your report.
  - Failure to include the Makefile or to clearly explain the compilation procedure **may result in no credit being awarded**.

Submission

- You should submit a single **[studentID].zip** file that includes:

1. All source code of the **simulator**
2. A **Makefile** for compiling the simulator
3. The **testcase generator** source code
4. The **graph generator** source code
5. A **report** named **[studentID].pdf**

QA

- QA
  - For general question: Google QA Spreadsheet(PA3)
    - https://docs.google.com/spreadsheets/d/1sFQfJBmQmL8oFcTwNPbJefb3Janst9uMZIDW_hM_nHI/edit?gid=312764764#gid=312764764
  - For personal question: Subject e-mail
    - os.at.ajou@gmail.com
    - When sending an email, please follow the below policy

Entitle your emails with your subsection info and a concise description of your inquiries.

Include your name and Student ID in the email

Ex) Title: [OS] Question regarding assignment/lecture

Notice

- Duration
  - Start: **11/18** 23:59 PM
  - End: **12/2** 23:59 PM
  - Late Submission: **12/4** 23:59AM
    - 25% penalty of final score of “this” assignment
    - After late submission, all of submissions will not be accepted