CSCI 3150 Introduction to Operating Systems: Assignment Three

1. Basic Information

- Topic: Address Translation, Swapping
- Total Marks: 100 + 10 bonus
- **Deadline: 18:00:00 p.m., Mon, Nov 18th**
- Submission: Github classroom
- High-level aim:
    - Gaining a deeper understanding of multi-level paging and virtual-to-physical address translation
    - Implementing three common swapping policies (LFU, LRU, and CLOCK algorithms) to simulate cache behavior, improving understanding of how different algorithms manage memory in constrained environments.

2. Task Overview

In assignment three, you have two separate questions.

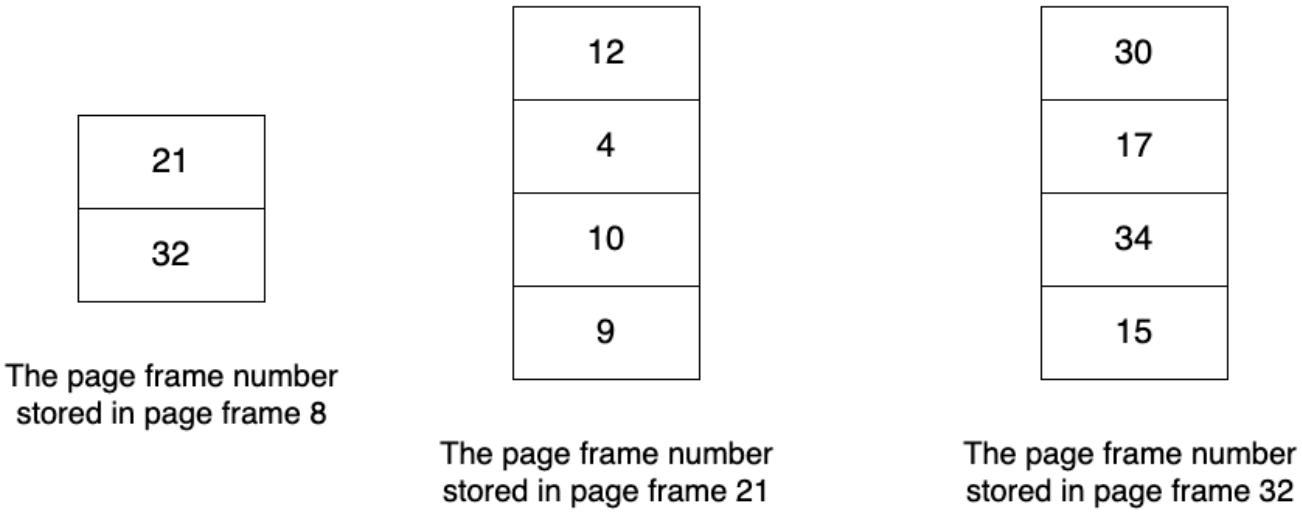
2.1. Question 1: Address Translation (40 marks)

(a)A certain computer has a 32-bit virtual address space, and the page size is 1024 bytes. Each page table entry is 4 bytes long. Since each page table must fit within a page, multi-level page tables are needed. Then:

1. How many levels of page tables are required?
2. When mapping addresses, how many parts is needed to divide logical address, and how many bits are in each part?

(b)In a system where the main memory is byte-addressable and uses paging, the physical page frame size is 8 bytes. Each page table entry occupies 2 bytes. Process P has a logical address space of 64 bytes. The PCB of process P stores the physical page frame number of the outermost page table, which is 8. The diagram below shows the physical page frame numbers where each level of page tables for process P is stored (other contents of the page table entries are omitted). Please answer:

1. Into how many levels is process P’s page table divided?
2. Calculate the physical address corresponding to logical address 20 generated during the execution of process P.



2.2. Question 2: Swapping (60 marks + 10 bonus)

In this question, you would be guided to implement three commonly used swapping policies, **LFU, LRU and CLOCK algorithm**. You will get 60 marks if you implement LFU and LRU algorithm correctly and 10 bonus marks if you implement Clock algorithm correctly. We provide a framework for you to implement these algorithms. The followings are some descriptions about the framework.

We simply simulate a cache that contains a reference to the cache line, a global time counter, and a Pointer of the Clock algorithm.

Assume that the cache can only hold 5 cache lines.

```
#define CACHE_SIZE 5

typedef struct {
    CacheLine lines[CACHE_SIZE];
} CacheLinePool;

typedef struct {
    CacheLinePool *line_pool;
    unsigned int time;            // Global time counter
    int clock_pointer;            // Pointer of the Clock algorithm
} Cache;
```

You need to implement the cache line yourself, here are some tips: you need to store the corresponding memory addresses and variables for the LFU, LRU, and CLOCK algorithms.

You are also required to implement functions:

create_cache, cache_access_lru, cache_access_lfu and cache_access_clock in cache.c

For create_cache, we have already completed part of the initialization of cache. You only need to add the initialization of cacheline.

For cache_access_lru, cache_access_lfu and cache_access_clock, they are the specific implementations of the three swapping algorithms. You need to check whether the cache hits, replace it if it does not hit, and print relevant information.

One testcases is provided for you to validate your implementation. We show this test case for your better understandings of this problem.

```
0x1A2B
0x3C4D
0x1A2B
0x5E6F
0x7A8B
0x9A8B
0x7A9F
0x7A8B
0x7CAB
0x1A2B
0x3C4D
```

Each row represents an access request for that address.

The output of the program involves whether cache hits or not and the replacement status if missed like the followings for the case(If you don't implement the bonus part):

```
Using LRU replacement strategy：
LRU: Cache miss, replaced line 0
LRU: Cache miss, replaced line 1
LRU: Cache hit at line 0
LRU: Cache miss, replaced line 2
LRU: Cache miss, replaced line 3
LRU: Cache miss, replaced line 4
LRU: Cache miss, replaced line 1
LRU: Cache hit at line 3
LRU: Cache miss, replaced line 0
LRU: Cache miss, replaced line 2
LRU: Cache miss, replaced line 4

Use LFU replacement strategy:
LFU: Cache miss, replaced line 0
LFU: Cache miss, replaced line 1
LFU: Cache hit at line 0
LFU: Cache miss, replaced line 2
LFU: Cache miss, replaced line 3
LFU: Cache miss, replaced line 4
LFU: Cache miss, replaced line 1
LFU: Cache hit at line 3
LFU: Cache miss, replaced line 1
LFU: Cache hit at line 0
LFU: Cache miss, replaced line 1

Use Clock replacement strategy:
```

(If you implement the bonus part correctly):

```
Using LRU replacement strategy：
LRU: Cache miss, replaced line 0
LRU: Cache miss, replaced line 1
LRU: Cache hit at line 0
LRU: Cache miss, replaced line 2
LRU: Cache miss, replaced line 3
LRU: Cache miss, replaced line 4
LRU: Cache miss, replaced line 1
LRU: Cache hit at line 3
LRU: Cache miss, replaced line 0
LRU: Cache miss, replaced line 2
LRU: Cache miss, replaced line 4

Use LFU replacement strategy:
LFU: Cache miss, replaced line 0
LFU: Cache miss, replaced line 1
LFU: Cache hit at line 0
LFU: Cache miss, replaced line 2
LFU: Cache miss, replaced line 3
LFU: Cache miss, replaced line 4
LFU: Cache miss, replaced line 1
LFU: Cache hit at line 3
LFU: Cache miss, replaced line 1
LFU: Cache hit at line 0
LFU: Cache miss, replaced line 1

Use Clock replacement strategy:
Clock: Cache miss, replaced line 0
Clock: Cache miss, replaced line 1
Clock: Cache hit at line 0
Clock: Cache miss, replaced line 2
Clock: Cache miss, replaced line 3
Clock: Cache miss, replaced line 4
Clock: Cache miss, replaced line 0
Clock: Cache hit at line 3
Clock: Cache miss, replaced line 1
Clock: Cache miss, replaced line 2
Clock: Cache miss, replaced line 4
```

Meanwhile, your implementation would be verified by some hidden testcases. You could refer to the tutorial slides to get some hints about how to implement this.

Please strictly follow this format for output, because the automated test cases will be scored based on the output

3. Submission Content

For Question 1, you need to submit a single pdf file containing your answer. Please rename your answer pdf to xxxxx.pdf, where xxxxx is your student ID. Please put the pdf in the same directory as Question2 folder

For Question 2, you are required to modify `cache.h` and `cache.c`. **Do not modify other files**

Before submission, please ensure you have completed (1) the declaration of originality at `declaration_en.doc`, and (2) acknowledgment of AI tools at `ai_declaration.txt`.

4. Generative AI Policy

In assignment three, we don't allow usage of generative AI tools.

- Using AI tools without properly understanding of the generated code and answers can put you at risk in quizzes/exams.

5. Other Notes

- For coding questions, we will compile, run, and grade your program with Ubuntu latest, which is the same as the autograding environment. Please make sure your program sources are compatible with the corresponding version of Ubuntu. Otherwise, 0 marks will be given.
- Note that we can only grade what you submit in the github repo. You will lose write access to the github repo after deadline. Please find related policies on the course website. To use grace token, we provide a separate assignment in github classroom named "assignment_3_grace_token" for your submission. The entrance to this assignment will be released immediately after the normal deadline. DO NOT accept grace token assignment unless you do want to use grace token.
- TA CHEN, kaiwen are responsible for this assignment. Questions about the assignment via Piazza are welcomed. Due to massive class size, no individual email will be replied. Requests including but not limited to asking TA to set up environment, write code and debug for you will be rejected according to regulations.