CSEN383: Project 4

Group-I
1) Simran Dhawan (07700011429)
2) Tishyaketu Atul Deshpande (07700000172)
2) Atharva Abhay Bal (07700005835)
3) Sowmya Baluvu (07700011659)
4) Ziang Chen (07700006146)

Introduction

The project involves developing a simulator for running processes and using algorithms to evict pages from memory. In a 60 second interval, 150 processes will be run. Each process has the following properties:

- The start time is randomly chosen between [0, 60) seconds.
- The size of the processes is randomly selected from {5, 11, 17, 31} MB.
- The service duration of processes is randomly chosen from {1, 2, 3, 4, 5} seconds.

There are 100 pages in memory, each page stores 1 MB of data.
A process is eligible to be started after its start time. The process is started if there are at least 4 pages free in memory. If there are less than 4 free pages, the process waits to be started.

When a process starts, its page 0 is loaded in memory. Each process runs for the service duration. Every 100 milliseconds, it performs requests for a new page. Assuming the process's current page is *i*, the new page is computed using the following procedure:

- Generate a random number r between [0, 10)
- If r is in [0, 6], the next page would be i-1, i or i+1 with equal probability.
    - Generate a new random number r2 between [0, 2].
    - Next page number is: i + r2 - 1 (i + [0,2]-1 = i + [-1, 1])
- If r is in [7, 9]:
    - Generate a random number r3 between [2, 9] using 2 + rand()%8
    - Generate a random number sign between {-1 or 1}
    - Next page number is: i + (sign x r3)
- If the next page number is beyond the max page of the process (for ex, current page = 29, max page = 30, next page = 33), the next page is chosen by wrapping around to 0 after the max page (33%31 = 2). Similarly, if the next page is -1 and the current page is 0, the next page becomes 30 (wraps around).

Simulation

The simulator performs the following steps:

- The user provides page replacement algorithm to use via a command line argument.
    - ./program {FIFO, LRU, LFU, MFU, RANDOM}
- 5 iterations are run. In each iteration, the steps listed below are performed.
    - The 150 processes are initialized using the arrival time, service duration and page limit described in the project problem statement.
    - The processes are sorted in increasing order of arrival timestamp.
    - A page list (memory) containing 100 empty pages is initialized.
    - The timestamp (in milliseconds) is started from 0. It is incremented by 100 ms in a loop.
    - In each run of the loop, the following steps are performed.
        - Iterate over all the processes. If a process is eligible to be started (arrival time <= current timestamp) and it has not started yet:
            - Check if number of free pages is at least 4.
            - If yes, the first page of the process is added to the page list, and the process is marked as started.
            - Logging: A record containing <timestamp_ms, process_id, Enter, process_size, service_duration, process_size> is logged.
            - The memory map is also logged.
        - For each of the running processes:
            - Get the next page it would access using the locality reference method described above.
            - If the next page already exists in memory, increment the hit counter.
                - Logging: A record containing <timestamp_ms, process_id, page_referenced, Found Page In Memory> is logged.
            - If not:
                - Increment the miss counter.
                - Check if there is at least one free page in memory.
                    - If yes, add the next page to the memory.
                    - If no, evict a page from memory using the user provided algorithm.
                    - Add the next page to the memory.
                    - Logging: A record containing <timestamp_ms, process_id, page_referenced, Not Found Page In Memory> is logged. The memory map before and after eviction is also logged.
            - Decrease the remaining service duration by 100 ms
            - If the remaining service duration becomes 0:
                - Release all the pages of the process from memory.
                - Logging: A record containing <timestamp_ms, process_id, Exit, process_size> is logged.

        ○ The memory map is logged before and after releasing the pages of this process from the memory.
    ○ Report the number of swapped in processes and the hit rate by averaging across the 5 iterations.

Results

The hit rate for each of the page replacement algorithms is described in the table below. For each of the strategies, the number of processes swapped in was 150, meaning all the processes were able to be scheduled in the 60 second interval.

Strategy	**Average Hit Rate (%)**	**Hit Rate In Each Iteration (%)**
FIFO	65.739853%	65.515694%, 66.519913%, 66.101692%, 66.145378%, 64.321266%
RANDOM	65.015068%	66.255608%, 64.064514%, 64.806038%, 64.756874%, 65.242104%
MFU	65.063293%	64.955154%, 65.911949%, 65.021187%, 65.286346%, 64.072395%
LFU	65.848976%	65.582962%, 66.561844%, 66.207626%, 66.475769%, 64.321266%
LRU	65.674377%	65.538116%, 66.373169%, 65.868645%, 66.189430%, 64.321266%

Example records from simulation

The records at various stages in the simulation are gathered and shared in this section.

Format of memory map:
|3 digit page number: process_id[page_number][f:<first time the page was loaded in memory>, l: <last time the page was accessed>, c: <number of times the page was accessed>].
For free pages, all the fields except page number are marked as *.

Process starts

[Starting a new process] timestamp_ms: 40000, process_id: 3, Enter, service_duration_ms: 1000, process_size: 11
Memory map:

1. |000: 003[00][f:40000, l:40000, c:001]|
2. |001: ***[**][f:*****, l:*****, c:***]|
3. |002: ***[**][f:*****, l:*****, c:***]|
4. |003: ***[**][f:*****, l:*****, c:***]|
5. |004: ***[**][f:*****, l:*****, c:***]|
6. …

[Starting a new process] timestamp_ms: 40000, process_id: 4, Enter, service_duration_ms: 1000, process_size: 17
Memory map:

1. |000: 003[00][f:40000, l:40000, c:001]|
2. |001: 004[00][f:40000, l:40000, c:001]|
3. |002: ***[**][f:*****, l:*****, c:***]|
4. |003: ***[**][f:*****, l:*****, c:***]|
5. |004: ***[**][f:*****, l:*****, c:***]|
6. …

Process completes

[Finishing a process] timestamp_ms: 55900, process_id: 5, Exit, service_duration_ms: 0
Memory map before cleanup:

1. |000: 005[16][f:55800, l:55800, c:000]|
2. |001: 005[00][f:55700, l:55700, c:000]|
3. |002: 005[05][f:55900, l:55900, c:000]|
4. |003: 001[04][f:55700, l:55700, c:000]|
5. |004: 001[03][f:55800, l:55900, c:001]|
6. …

Memory map after cleanup:

1. |000: ***[**][f:*****, l:*****, c:***]|
2. |001: ***[**][f:*****, l:*****, c:***]|
3. |002: ***[**][f:*****, l:*****, c:***]|
4. |003: 001[04][f:55700, l:55700, c:000]|
5. |004: 001[03][f:55800, l:55900, c:001]|
6. …

The first 3 pages belonged to process id 5. All those become free pages after the cleanup.

Memory access – Page found in memory

[Accessing a new page] timestamp_ms: 41100, process_id: 2, page_referenced: 16, page_in_memory: true

[Accessing a new page] timestamp_ms: 41200, process_id: 2, page_referenced: 16, page_in_memory: true

Memory access – Page not found in memory, free space available

Memory map before adding new page:

1. |000: 002[00][f:41000, l:41000, c:001]|
2. |001: ***[**][f:*****, l:*****, c:***]|
3. |002: ***[**][f:*****, l:*****, c:***]|
4. |003: ***[**][f:*****, l:*****, c:***]|
5. |004: ***[**][f:*****, l:*****, c:***]|
6. …

[Accessing a new page] timestamp_ms: 41000, process_id: 2, page_referenced: 16, page_in_memory: false

Memory map after adding page 16 for process_id 2:

1. |000: 002[00][f:41000, l:41000, c:001]|
2. |001: 002[16][f:41000, l:41000, c:000]|
3. |002: ***[**][f:*****, l:*****, c:***]|
4. |003: ***[**][f:*****, l:*****, c:***]|
5. |004: ***[**][f:*****, l:*****, c:***]|
6. …

Memory access – Page not found in memory, evict to make room

The records are taken from the **FIFO** algorithm simulation.

[Accessing a new page] timestamp_ms: 41700, process_id: 2, page_referenced: 13, page_in_memory: false

Memory is full, evicting a page. Memory map before eviction:

1. |000: 002[00][f:41000, l:41500, c:002]|
2. |001: 002[16][f:41000, l:41200, c:002]|
3. |002: 002[15][f:41300, l:41300, c:000]|
4. |003: 002[07][f:41400, l:41400, c:000]|
5. |004: 002[10][f:41600, l:41600, c:000]|

[Page Eviction] process_id: 002, page_number: 00, first_access_timestamp_ms:41000, last_access_timestamp_ms: 41500, access_count: 2

Memory map after eviction:

1. |000: ***[**][f:*****, l:*****, c:***]|
2. |001: 002[16][f:41000, l:41200, c:002]|
3. |002: 002[15][f:41300, l:41300, c:000]|
4. |003: 002[07][f:41400, l:41400, c:000]|
5. |004: 002[10][f:41600, l:41600, c:000]|
6. …

Memory map after adding page 13 for process_id 2:

1. |000: 002[13][f:41700, l:41700, c:000]|
2. |001: 002[16][f:41000, l:41200, c:002]|
3. |002: 002[15][f:41300, l:41300, c:000]|
4. |003: 002[07][f:41400, l:41400, c:000]|
5. |004: 002[10][f:41600, l:41600, c:000]|
6. …

Output generation instructions

Compile the program:

```
gcc driver.c algorithms.c page_utils.c  -o program
```

Run the program:

```
./program [FIFO/LRU/LFU/MFU/RANDOM]
```

Example:

```
./program FIFO
```

To store the output records to a file:

```
./program FIFO > FIFO
```

To run the program with all algorithms:

```
./run_all.sh
```