# East West University

## Department of CSE

## PROJECT REPORT

**Course Code and Name:** CSE325 (Operating System)

**Section :** 05

**Semester :** Fall 2023


**Submitted by**

Name : Sami Al Zabid

ID : 2022-1-60-103

**Submitted to**

Md. Mahir Ashhab

Lecturer, Department of Computer Science and Engineering


**Date of Submission :** December 23, 2023

# Introduction:

**WSClock Page Replacement Algorithm:** An improved algorithm, which is based on the clock algorithm but also uses the working set information is called WSClock page replacement algorithm. In WSClock, when a page fault occurs, the algorithm not only checks the reference bit but also considers the time since the page was last referenced. If a page's reference bit is 1 and it falls within the working set time window, the page is considered recently used and is not a candidate for replacement. Otherwise, if the page is not in the working set, it may be a candidate for replacement.

# Theoretical background:

Implementing the WSClock algorithm involves maintaining a circular list of page frames, tracking the reference bit for each page, and considering the temporal locality of page references through the working set information. Below is a theoretical step-by-step explanation of how we can implement the WSClock algorithm:

**Data Structures:**

Maintain a circular list of page frames to represent the available memory pages. Each page frame should have associated information:

- Reference bit: A flag indicating whether the page has been referenced recently (1 if referenced, 0 if not).
- Last referenced time: A timestamp indicating when the page was last referenced.
- Other necessary information, such as the page number.

**Working Set Parameters:**

- Working set time window (W): The time interval within which a page is considered part of the working set.
- Clock hand: A pointer to the current position in the circular list.

**Page Fault Handling:**

When a page fault occurs follow these steps:

i. Start at the page pointed to by the clock hand in the circular list.
ii. Check the reference bit of the page: If the reference bit is 0, indicating the page hasn't been referenced recently:
   - Check if the last referenced time is within the working set time window (W).
   - If within the working set, update the reference bit and last referenced time, and move the clock hand to the next page.
   - If not within the working set, replace the page with the new page, updating the necessary information.

If the reference bit is 1:

- Check if the last referenced time is within the working set time window (W).
- If within the working set, update the reference bit and last referenced time, and move the clock hand to the next page.
- If not within the working set, continue checking other pages until a suitable candidate is found.

**Reference Bit and Clock Hand Updates:**

- Update the reference bit and last referenced time for each page when it is referenced.
- Move the clock hand to the next page in the circular list after each page reference.

**Timekeeping:**

- Maintain a global clock or timestamp to record the current time.
- Update the last referenced time for each page whenever it is referenced.

**Working Set Dynamics:**

- The working set time window (W) determines the temporal locality of page references. Adjust this parameter based on the characteristics of the system and the workload.

**Initialization:**

- Initialize the circular list, reference bits, last referenced times, and other necessary data structures.

# Implementation Details:

The C++ language used for implementation in this algorithm.

- The code includes necessary C++ standard libraries (iostream, cstdlib, ctime) for input/output operations, random number generation, and time functions.
- Macros are defined for maximum pages (MAX_PAGE), maximum RAM size (MAX_RAM), simulation time (TIME), time interval of working set (TAU), and time to reset reference bits (T_MES).

**Data Structures:**

1. **Page Struct:**
    - **id**: Identifier for the page.
    - **Ref_bit**: Reference bit indicating whether the page has been accessed.
    - **time**: Timestamp indicating the last access time.
2. **Node Struct:**

- Represents a node in a doubly-linked list. Each node contains a **Page** and pointers to the next and previous nodes.

3. **LinkedList Struct:**
    - Keeps track of the size and the front node of a doubly-linked list.

**Functions:**

1. **initializeList:**
    - Initializes a linked list with a dummy node (head).
2. **getSize:**
    - Returns the size of the linked list.
3. **isEmpty:**
    - Checks if the linked list is empty.
4. **searchPage:**
    - Searches for a page with a given ID in the linked list.
5. **insertPage:**
    - Inserts a new page after a specified position in the linked list.
6. **removePage:**
    - Removes a page from the linked list.
7. **printList:**
    - Prints the contents of the linked list.
8. **randomPage:**
    - Generates a random page with a random ID, reference bit set to 1, and a timestamp.
9. **resetRef_bit:**
    - Resets the reference bits of all pages in the linked list to 0.
10. **replacePage:**
    - Finds a page to substitute based on the Working Set Clock algorithm. It looks for pages with a reference bit of 0 and a timestamp older than a threshold (**TAU**).
11. **workingSetClock:**
    - Simulates the working set clock algorithm over a specified time (**TIME**). It generates random pages and performs page replacement as necessary.

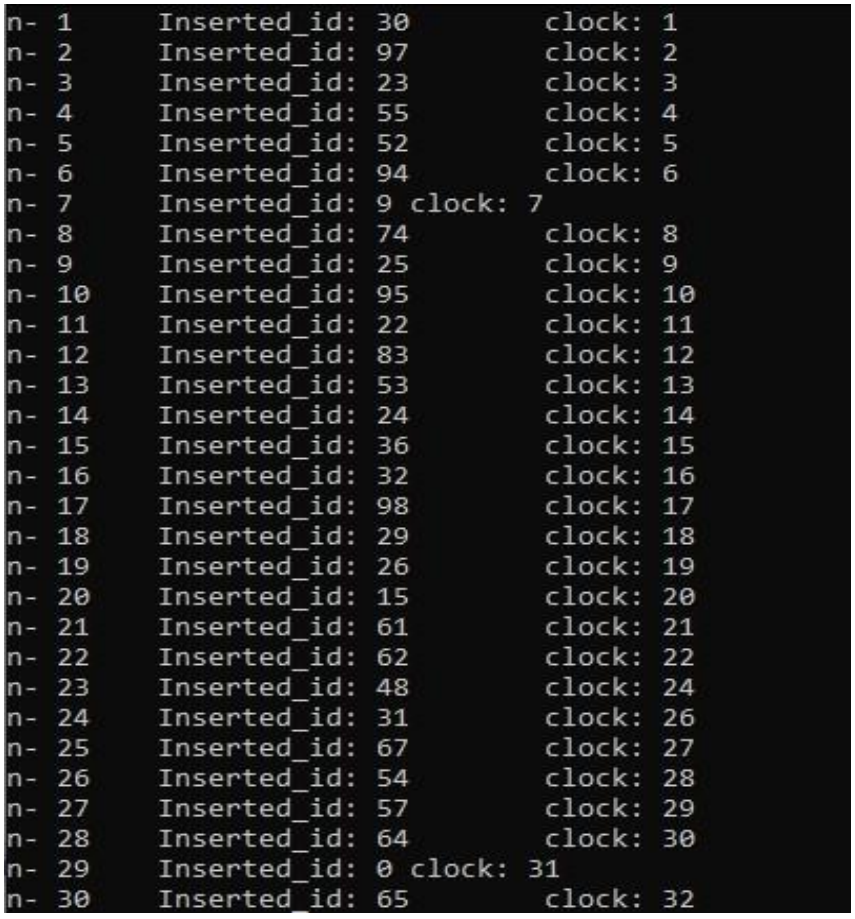
12. **main:**

- Initializes a linked list and runs the **workingSetClock** simulation.

**Simulation Details:**

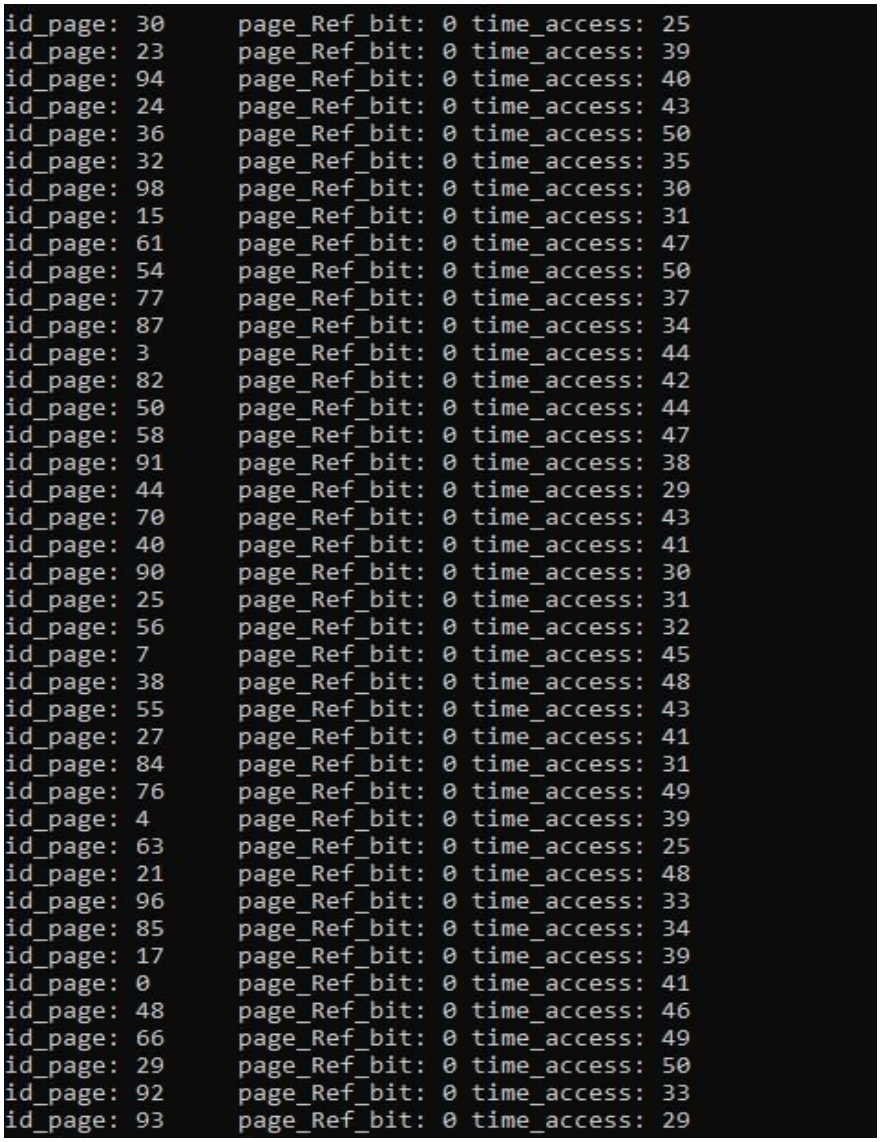
- The main function initializes a linked list and calls **workingSetClock** to simulate the Working Set Clock algorithm over a specified time.
- The algorithm randomly generates pages and simulates page hits.
- When a page is accessed:
  - If the page is in memory, its reference bit is set to 1, and the timestamp is updated.
  - If the page is not in memory: If there is space in memory, the page is inserted. If memory is full, a page is selected for replacement based on the Working Set Clock algorithm.

# Outputs:



```
n- 31    Inserted_id: 77        clock: 33
n- 32    Inserted_id: 87        clock: 34
n- 33    Inserted_id: 3 clock: 36
n- 34    Inserted_id: 34        clock: 38
n- 35    Inserted_id: 12        clock: 39
n- 36    Inserted_id: 16        clock: 40
n- 37    Inserted_id: 82        clock: 42
n- 38    Inserted_id: 50        clock: 44
n- 39    Inserted_id: 58        clock: 47
n- 40    Inserted_id: 8 clock: 48
                                    Reset CLOCK
n- 41    Inserted_id: 81        clock: 2
n- 42    Inserted_id: 39        clock: 7
n- 43    Inserted_id: 73        clock: 9
n- 44    Inserted_id: 59        clock: 10
n- 45    Inserted_id: 5 clock: 11
n- 46    Inserted_id: 85        clock: 12
n- 47    Inserted_id: 80        clock: 13
n- 48    Inserted_id: 17        clock: 14
n- 49    Inserted_id: 6 clock: 15
n- 50    Inserted_id: 35        clock: 16
Removed_id: 97 Inserted_id: 91  clock: 17
Removed_id: 81 Inserted_id: 1   clock: 18
Removed_id: 25 Inserted_id: 69  clock: 19
Removed_id: 55 Inserted_id: 44  clock: 22
Removed_id: 34 Inserted_id: 70  clock: 24
Removed_id: 64 Inserted_id: 60  clock: 25
Removed_id: 39 Inserted_id: 40  clock: 26
Removed_id: 74 Inserted_id: 79  clock: 27
Removed_id: 48 Inserted_id: 90  clock: 30
```

```
Removed_id: 8 Inserted_id: 48   clock: 42
Removed_id: 9 Inserted_id: 22   clock: 47
Removed_id: 45 Inserted_id: 66  clock: 49
Removed_id: 62 Inserted_id: 29  clock: 50
                                    Reset CLOCK
Removed_id: 74 Inserted_id: 33  clock: 1
Removed_id: 86 Inserted_id: 41  clock: 3
Removed_id: 33 Inserted_id: 34  clock: 5
Removed_id: 57 Inserted_id: 9   clock: 7
Removed_id: 41 Inserted_id: 8   clock: 8
Removed_id: 67 Inserted_id: 86  clock: 10
Removed_id: 34 Inserted_id: 11  clock: 15
Removed_id: 22 Inserted_id: 5   clock: 16
Removed_id: 9 Inserted_id: 6    clock: 18
Removed_id: 8 Inserted_id: 47   clock: 19
Removed_id: 81 Inserted_id: 92  clock: 21
Removed_id: 65 Inserted_id: 74  clock: 22
Removed_id: 60 Inserted_id: 53  clock: 24
Removed_id: 52 Inserted_id: 93  clock: 25
Removed_id: 16 Inserted_id: 31  clock: 26
Removed_id: 11 Inserted_id: 64  clock: 27
Removed_id: 5 Inserted_id: 51   clock: 28
Removed_id: 86 Inserted_id: 13  clock: 30
Removed_id: 6 Inserted_id: 89   clock: 34
Removed_id: 12 Inserted_id: 41  clock: 36
Removed_id: 74 Inserted_id: 43  clock: 38
Removed_id: 47 Inserted_id: 75  clock: 40
Removed_id: 53 Inserted_id: 14  clock: 44
                                    Reset CLOCK
count hit: 85
```




The code outputs information about inserted and replaced pages at each clock tick, as well as statistics on the number of page hits. Then finally print the list of pages remains. By tracing outputs we see it works according to theoretical process and implement the WSClock page replacement successfully.

## Conclusion:

In conclusion, the program provides a simple implementation of a page replacement algorithm using the working set clock policy. It randomly generates page accesses, manages a fixed-size RAM, and simulates the replacement of pages based on the Working Set Clock algorithm.