CSE 4733/6733 - Operating System 1

S. Torri, T. Ritter

Mississippi State University

CSE 4733/6733 - Operating System 1

S. Torri, T. Ritter

Mississippi State University

Introduction to Memory Management

- Fact: Given that main memory holds a subset of all the pages in the system like a cache.
- Problem: Picking a replacement policy minimizes the number of cache misses.

Introduction to Memory Management
- Fact: Given that main memory holds a subset of all the pages in the system like a cache.
- Problem: Picking a replacement policy minimizes the number of cache misses.

1. A cache miss occurs when a page is not in main system memory, thus requiring a fetching from long-term storage (e.g. SSD).
2. The purpose of a memory management replacement policy, often referred to as a page replacement policy, is to determine which memory pages to evict (replace) when a page fault occurs and the required page is not present in the main memory (RAM). These policies are crucial in systems that implement virtual memory, where the main memory acts as a cache for the secondary storage (like a hard disk or SSD), which holds the entire address space of a process.

Main Objectives

The main objectives of a memory management replacement policy are:

- Maximize CPU Utilization
- Minimize Page Fault Rate
- Efficient Use of Memory
- Fairness
- Predictability
- Low Overhead

Main Objectives
The main objectives of a memory management replacement policy are:
- Maximize CPU Utilization
- Minimize Page Fault Rate
- Efficient Use of Memory
- Fairness
- Predictability
- Low Overhead

1. Maximize CPU Utilization: By ensuring that the necessary pages are available in memory when needed, the CPU can continue executing processes without waiting for frequent page swaps between main memory and secondary storage.
2. Minimize Page Fault Rate: A lower page fault rate means that the required pages are already in memory when needed, leading to faster execution times.
3. Efficient Use of Memory: The policy should ensure that the limited memory resources are used judiciously, keeping the most frequently or recently accessed memory pages.

Main Objectives

The main objectives of a memory management replacement policy are:

- Maximize CPU Utilization
- Minimize Page Fault Rate
- Efficient Use of Memory
- Fairness
- Predictability
- Low Overhead

Main Objectives
The main objectives of a memory management replacement policy are:
- Maximize CPU Utilization
- Minimize Page Fault Rate
- Efficient Use of Memory
- Fairness
- Predictability
- Low Overhead

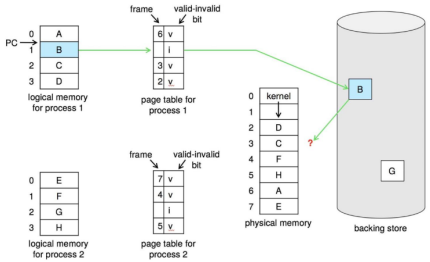
1. Fairness: Ensure that all processes get a fair share of the memory and one process doesn't monopolize the memory resources at the expense of others.
2. Predictability: The policy should provide consistent performance, allowing system designers and users to anticipate how the system will behave under different workloads.
3. Low Overhead: The algorithm used for the replacement policy should be efficient in terms of time and space complexity. High overheads can negate the benefits provided by the policy.

Table of Contents

Table of Contents
1. What Happens if there are no free frames?
2. Frame allocation
3. Page Fault
4. Replacement Algorithms
5. Conclusion

What to do when you run out of memory - Part 1

Silberschatz, Galvin, and Gagne ©2018



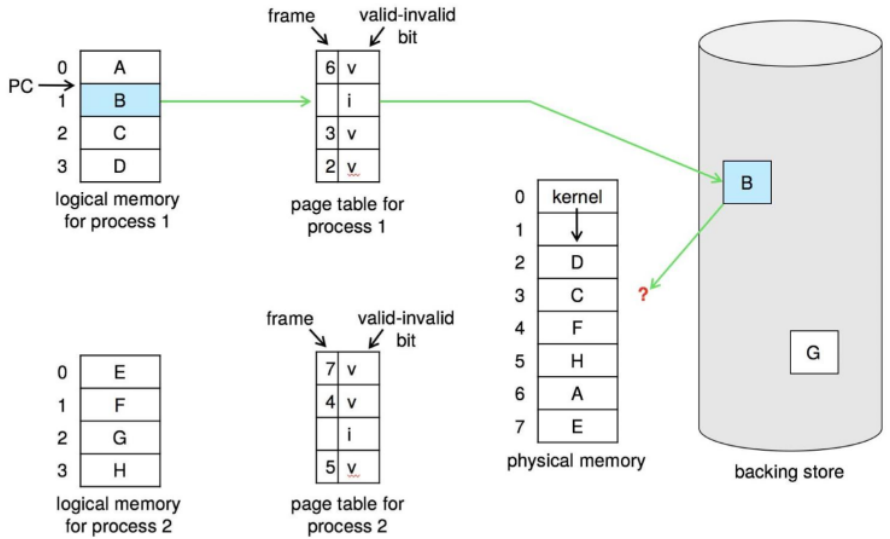
- Process pages have been used up by running applications.
- Operating system requires memory for I/O buffers, etc.
- How much to allocate to each?
- Page replacement criteria:
  - Algorithm options (FIFO, LRU, Optimal, Clock)
  - Performance and trade-offs

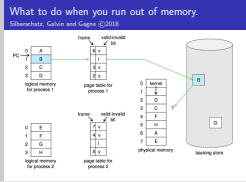
What to do when you run out of memory - Part 1
Silberschatz, Galvin, and Gagne ©2018
Process pages have been used up by running applications.
Operating system requires memory for I/O buffers, etc.
How much to allocate to each?
Page replacement criteria:
Algorithm options (FIFO, LRU, Optimal, Clock)
Performance and trade-offs

1. A strategy is required when no free frames are available.
2. We want an algorithm that will result in a minimum number of page faults.

What to do when you run out of memory.

Silberschatz, Galvin and Gagne ©2018





What to do when you run out of memory - Part 2

Strategies to manage memory shortages

Key Strategies:

- **Increase Physical Memory:** Add more RAM if possible.
- **Optimize Virtual Memory:** Ensure swap space is adequate.
- **Reduce Memory Footprint:** Free unused memory, fix leaks.
- **Use Efficient Page Replacement:** LRU, Clock, or Working Set Model.
- **Prevent Thrashing:** Adjust multiprogramming level.
- **Terminate or Suspend Processes:** Kill low-priority tasks.

What to do when you run out of memory - Part 2
Strategies to manage memory shortages
Key Strategies:
- Increase Physical Memory: Add more RAM if possible.
- Optimize Virtual Memory: Ensure swap space is adequate.
- Reduce Memory Footprint: Free unused memory, fix leaks.
- Use Efficient Page Replacement: LRU, Clock, or Working Set Model.
- Prevent Thrashing: Adjust multiprogramming level.
- Terminate or Suspend Processes: Kill low-priority tasks.

1. Thrashing can be reduced by adjusting the working set size or the level of multiprogramming.
2. Consider using memory compression techniques like Zswap or Zram to optimize RAM usage.

Table of Contents

Table of Contents
What Happens if there are no free frames?
Frame allocation
Page Fault
Replacement Algorithms
Conclusion

Allocation of Frames

Silberschatz, Galvin and Gagne ©2018

- Each process needs a minimum number of frames.
- Maximum, of course, is the total frames in the system.
- Two major allocation schemes:
  - Equal Allocation
  - Proportional allocation
  - Priority allocation
  - GlobalReplacement
  - Local Replacement

Allocation of Frames
Silberschatz, Galvin and Gagne ©2018
- Each process needs a minimum number of frames.
- Maximum, of course, is the total frames in the system.
- Two major allocation schemes:
  - Equal Allocation
  - Proportional allocation
  - Priority allocation
  - GlobalReplacement
  - Local Replacement

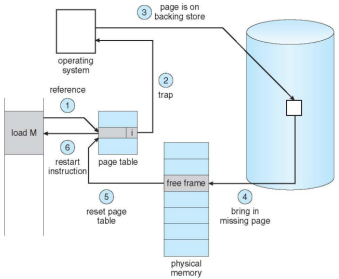
1. **Equal allocation**: For example, if there are 100 frames and five processes, give each process 20 frames.
2. **Proportional allocation**: Allocate according to the size of the process. Any changes in the number of processes affect the limit on process size changes
3. **Global Replacement**: process selects a replacement frame from the set of all frames.
   - One process can take a frame from another
   - Process execution time can vary greatly.
   - Greater throughput, so more common.
4. **Local Replacement**: each process selects from only its own set of allocated frames.
   - More consistent per-process performance.
   - But possibly underutilized memory.

Table of Contents

Table of Contents
What Happens if there are no free frames?
Frame allocation
Page Fault
Replacement Algorithms
Conclusion

Steps in Handling a Page Fault - Part 1

Silberschatz, Galvin and Gagne ©2018



1. **Process requests a memory page**
2. **Page fault trap**
3. **Check backing store**
4. **Identify a free frame**

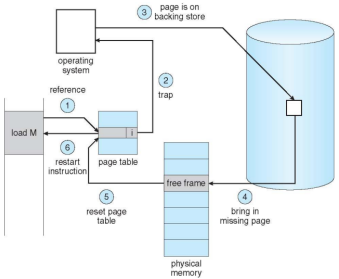
Steps in Handling a Page Fault - Part 1
Silberschatz, Galvin and Gagne ©2018

1. Process requests a memory page
2. Page fault trap
3. Check backing store
4. Identify a free frame

1. Process requests a memory page: A reference is made to a page that is not in memory.
2. Page fault trap: The operating system detects the missing page and generates a page fault.
3. Check backing store: The system verifies whether the required page is in secondary storage (swap space or disk).
4. Identify a free frame: If a free frame is available, use it; otherwise, select a victim page using a replacement algorithm.

Steps in Handling a Page Fault - Part 2

Silberschatz, Galvin and Gagne ©2018



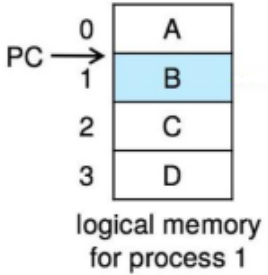
5. **Write back if necessary**
6. **Load the required page**
7. **Update page table**
8. **Restart the instruction**

Steps in Handling a Page Fault - Part 2
Silberschatz, Galvin and Gagne ©2018
5. Write back if necessary
6. Load the required page
7. Update page table
8. Restart the instruction

1. Write back if necessary: If the victim page has been modified, write it back to disk before replacing it.
2. Load the required page: Fetch the missing page from the disk into the allocated frame in physical memory.
3. Update page table: Modify the page table to reflect the new mapping between virtual and physical memory.
4. Restart the instruction: Resume the process execution at the point where the page fault occurred.

Goals of Page Replacement

Silberschatz, Galvin and Gagne ©2018



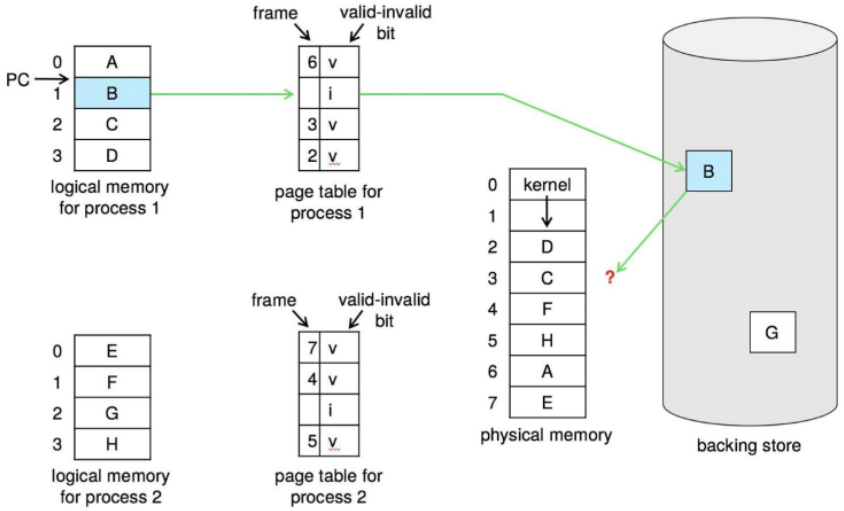
- Prevent **over-allocation** of memory by modifying page-fault service routine to include page replacement.
- Use **modify** (**dirty**) **bit** to reduce the overhead of page transfers - only modified pages are written to disk.
- Page replacement completes separation between logical and physical memory – large virtual memory can be provided on a smaller physical memory.

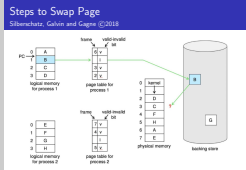
Goals of Page Replacement
Silberschatz, Galvin and Gagne ©2018
PC 0 A 1 B 2 C 3 D
logical memory for process 1
- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use modify (dirty) bit to reduce the overhead of page transfers - only modified pages are written to disk.
- Page replacement completes separation between logical and physical memory – large virtual memory can be provided on a smaller physical memory.

1. Notes:
   - **Dirty page**: a page has been modified and is thus dirty.
   - A dirty page must be written back to disk to evict it, which is expensive.
   - **Clean page**: a page that has not been modified (and is thus clean).
   - An eviction of a page is free. The physical page/frame can be reused for other purposes without additional I/O. Thus, some virtual memory systems prefer to evict clean pages over dirty pages.

Steps to Swap Page

Silberschatz, Galvin and Gagne ©2018

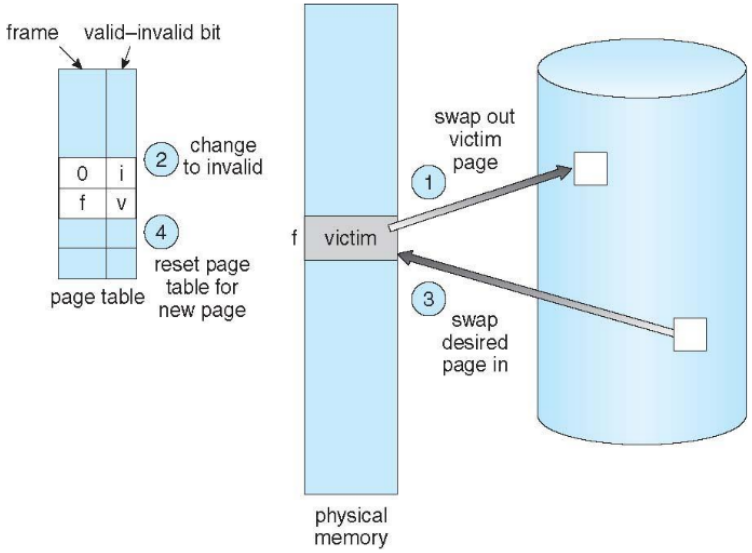


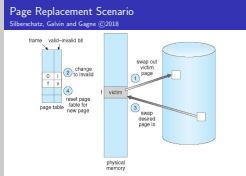


1. Two processes are attempting to use memory. The kernel requires memory to run as well.
2. Steps:
   - Program Counter (PC) is ready to execute the next instruction.
   - During a page table, look-up shows that the frame is not in physical memory.
   - An I/O request goes out to the backing store (e.g., SSD, HDD)
   - Unfortunately, there is no room in physical memory for frame "B."

Page Replacement Scenario

Silberschatz, Galvin and Gagne ©2018





1. A victim page is selected and marked to be written to file location or swap file.
2. The old page table frame entry has its valid-invalid bit set to invalid. The location is not in physical memory.
3. The new page is retrieved from the file location or swap partition and paced into physical memory.
4. The new page table for the frame is marked as valid.

Table of Contents

Table of Contents
What Happens if there are no free frames?
Frame allocation
Page Fault
Replacement Algorithms
Conclusion

Page and Frame Replacement Algorithms

Silberschatz, Galvin and Gagne ©2018

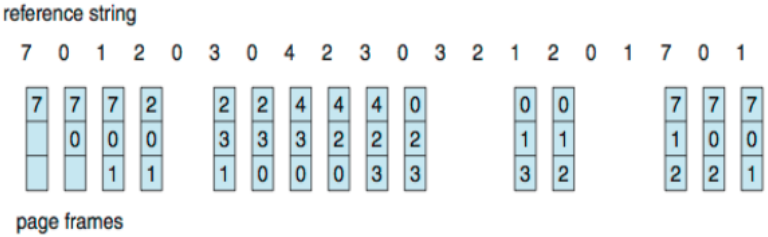
- Frame-allocation algorithm determines:
  - How many frames to give each process
  - Which frames to replace
- Page-replacement algorithm
  - Want the lowest page-fault rate on both first access and re-access.
- Evaluate the algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
  - String is just page numbers, not full addresses.
  - Repeated access to the same page does not cause a page fault.
  - Results depend on the number of frames available.

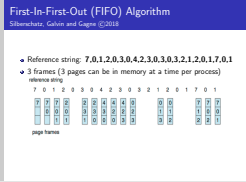
Page and Frame Replacement Algorithms
Silberschatz, Galvin and Gagne ©2018
- Frame-allocation algorithm determines:
  - How many frames to give each process
  - Which frames to replace
- Page-replacement algorithm
  - Want the lowest page-fault rate on both first access and re-access.
- Evaluate the algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
  - String is just page numbers, not full addresses.
  - Repeated access to the same page does not cause a page fault.
  - Results depend on the number of frames available.

First-In-First-Out (FIFO) Algorithm

Silberschatz, Galvin and Gagne ©2018

- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)





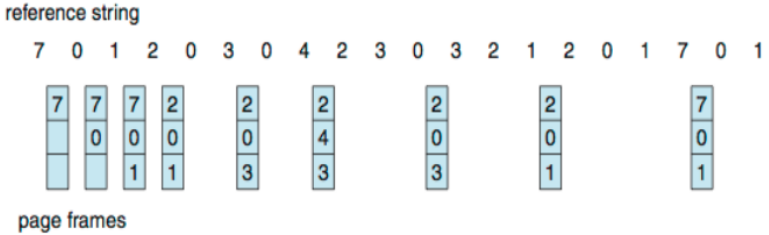
1. FIFO is simple but can suffer from Belady's anomaly, where increasing the number of frames can increase page faults.
2. One way to improve FIFO is by implementing a second-chance algorithm (also called the clock algorithm), which gives pages a second chance if they have been referenced recently.
3. Tracking additional metadata, such as reference bits, can help refine which page to replace, reducing unnecessary replacements.
4. FIFO is not always the best strategy; in practice, algorithms like LRU (Least Recently Used) or LFU (Least Frequently Used) often perform better in real workloads.

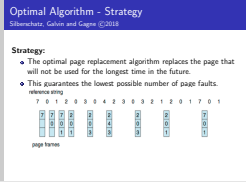
Optimal Algorithm - Strategy

Silberschatz, Galvin and Gagne ©2018

Strategy:

- The optimal page replacement algorithm replaces the page that will not be used for the longest time in the future.
- This guarantees the lowest possible number of page faults.





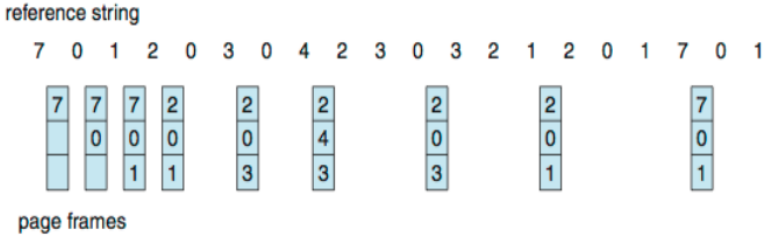
1. The first four frames remain the same, meaning the initial page replacements follow a normal sequence.
2. The optimal algorithm looks ahead to determine which page will not be needed for the longest period, minimizing page faults.
3. Unlike FIFO or LRU, this algorithm achieves the best-case scenario in terms of page faults.

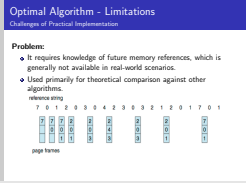
Optimal Algorithm - Limitations

Challenges of Practical Implementation

Problem:

- It requires knowledge of future memory references, which is generally not available in real-world scenarios.
- Used primarily for theoretical comparison against other algorithms.





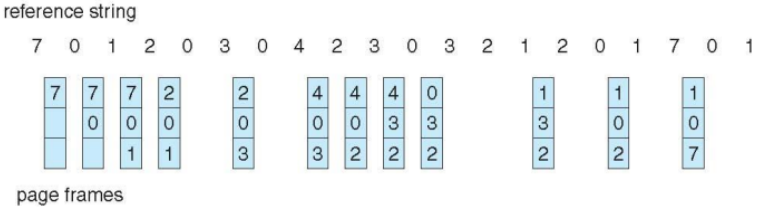
1. The biggest limitation of the optimal algorithm is that it requires knowledge of future memory accesses, which is impossible in real-world applications.
2. This algorithm is primarily used as a benchmark to compare real-world page replacement strategies.
3. While theoretical, studying the optimal algorithm helps understand the limits of page replacement efficiency.

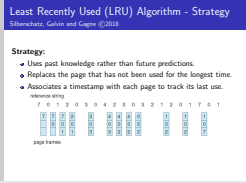
Least Recently Used (LRU) Algorithm - Strategy

Silberschatz, Galvin and Gagne ©2018

Strategy:

- Uses past knowledge rather than future predictions.
- Replaces the page that has not been used for the longest time.
- Associates a timestamp with each page to track its last use.





1. The LRU algorithm selects the page that was least recently used, assuming that pages used recently will be used again soon.
2. This approach typically results in fewer page faults than FIFO, as it adapts to program locality better.

Least Recently Used (LRU) Algorithm - Implementations

How to Implement LRU Efficiently

Counter Implementation:

- Each page table entry maintains a counter that records the last time the page was accessed.
- When a page is referenced, update its counter with the current clock value.
- When replacement is needed, search for the page with the smallest counter value.
- Requires a full search of the page table, making it expensive in practice.

Least Recently Used (LRU) Algorithm - Implementations
How to Implement LRU Efficiently
Counter Implementation:
- Each page table entry maintains a counter that records the last time the page was accessed.
- When a page is referenced, update its counter with the current clock value.
- When replacement is needed, search for the page with the smallest counter value.
- Requires a full search of the page table, making it expensive in practice.

1. The counter method provides accurate tracking of the least recently used page but requires a costly search operation for replacement.

Least Recently Used (LRU) Algorithm - Stack Implementation

Alternative Efficient Implementation

Stack Implementation:

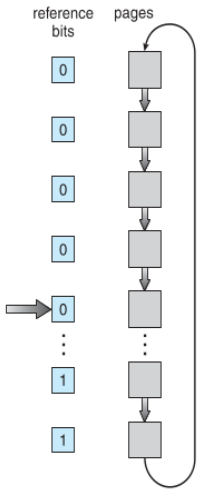
- Uses a stack (often a doubly linked list) to track page usage order.
- When a page is accessed, it is moved to the top of the stack.
- When replacement is needed, the page at the bottom of the stack is removed.
- Requires updating multiple pointers but avoids searching for the least recently used page.

Least Recently Used (LRU) Algorithm - Stack Implementation
Alternative Efficient Implementation
Stack Implementation:
- Uses a stack (often a doubly linked list) to track page usage order.
- When a page is accessed, it is moved to the top of the stack.
- When replacement is needed, the page at the bottom of the stack is removed.
- Requires updating multiple pointers but avoids searching for the least recently used page.

1. Stack implementation avoids the need to search for the least recently used page, making it more efficient than the counter method.
2. However, each memory access requires multiple pointer updates, increasing overhead.

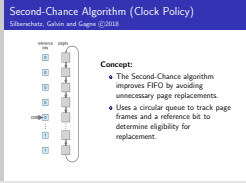
Second-Chance Algorithm (Clock Policy)

Silberschatz, Galvin and Gagne ©2018



Concept:

- The Second-Chance algorithm improves FIFO by avoiding unnecessary page replacements.
- Uses a circular queue to track page frames and a reference bit to determine eligibility for replacement.



1. The Second-Chance algorithm is an enhancement of FIFO, providing better page replacement efficiency.
2. It gives pages a 'second chance' before replacing them by checking their reference bit.

Second-Chance Algorithm - Steps

How the Algorithm Works

Steps:

1. Initialize a circular queue for page frames and an array to track reference bits.
2. When a page is accessed, set its reference bit to 1.
3. If a page needs replacement:
   - Traverse the circular queue.
   - If a page has a reference bit of 0, replace it with the new incoming page that caused the page fault.
   - If a page has a reference bit of 1, set it to 0 and move to the next page.
   - Repeat the process until a page with a reference bit of 0 is found and replaced by the new incoming page.

Second-Chance Algorithm - Steps
How the Algorithm Works
Steps:
1. Initialize a circular queue for page frames and an array to track reference bits.
2. When a page is accessed, set its reference bit to 1.
3. If a page needs replacement:
   - Traverse the circular queue.
   - If a page has a reference bit of 0, replace it with the new incoming page that caused the page fault.
   - If a page has a reference bit of 1, set it to 0 and move to the next page.
   - Repeat the process until a page with a reference bit of 0 is found and replaced by the new incoming page.

1. The Second-Chance algorithm operates as an enhanced FIFO, preventing unnecessary replacements by using reference bits.
2. A circular queue ensures that all pages get equal consideration before replacement.
3. When a page with a reference bit of 0 is found, it is replaced with the new page that triggered the page fault. This ensures efficient memory utilization.
4. If all pages have their reference bit set to 1, the algorithm cycles through once, resetting all bits to 0 before replacing the first available page.

Second-Chance Algorithm - Advantages and Limitations

Evaluating the Performance of Second-Chance

Advantages:

- Reduces unnecessary page replacements compared to FIFO.
- Simple to implement using a circular queue.

Limitations:

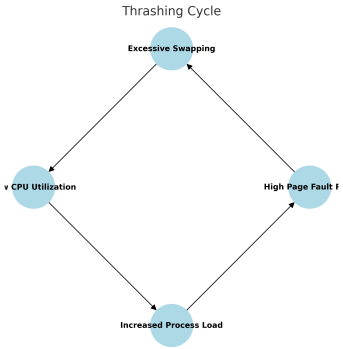
- Does not guarantee optimal page replacement.
- Can degrade to FIFO performance if all pages frequently have their reference bits set.

Second-Chance Algorithm - Advantages and Limitations
Evaluating the Performance of Second-Chance
Advantages:
- Reduces unnecessary page replacements compared to FIFO.
- Simple to implement using a circular queue.
Limitations:
- Does not guarantee optimal page replacement.
- Can degrade to FIFO performance if all pages frequently have their reference bits set.

1. While more efficient than FIFO, Second Chance is not as effective as LRU or Optimal replacement algorithms.
2. It works best when there is some level of page reuse but can struggle in highly dynamic workloads.

Thrashing - Definition

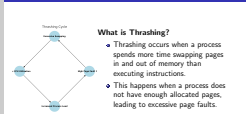
Silberschatz, Galvin and Gagne ©2018



What is Thrashing?

- Thrashing occurs when a process spends more time swapping pages in and out of memory than executing instructions.
- This happens when a process does not have enough allocated pages, leading to excessive page faults.

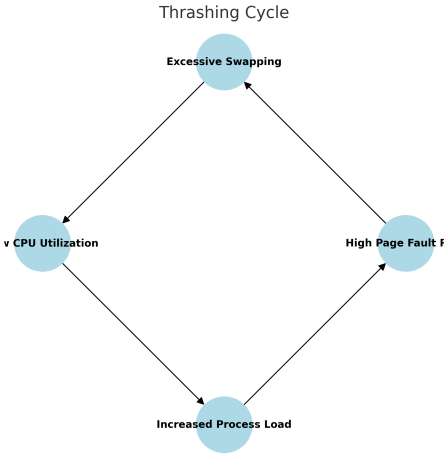
Thrashing - Definition
Silberschatz, Galvin and Gagne ©2018



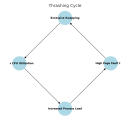
1. Thrashing happens when a process continuously generates page faults, preventing it from making real progress.
2. It is caused by a lack of sufficient frames for a process's working set, forcing it to reload pages constantly.

Thrashing - Definition

Silberschatz, Galvin and Gagne ©2018

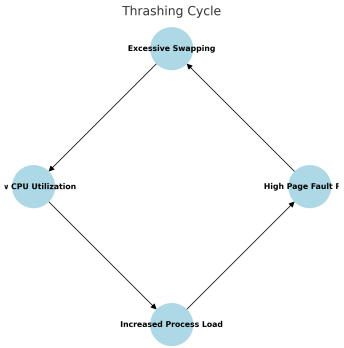


Thrashing - Definition
Silberschatz, Galvin and Gagne ©2018



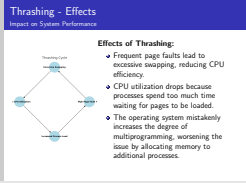
Thrashing - Effects

Impact on System Performance



Effects of Thrashing:

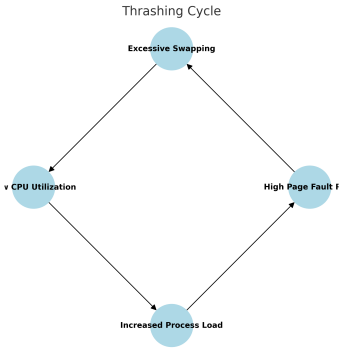
- Frequent page faults lead to excessive swapping, reducing CPU efficiency.
- CPU utilization drops because processes spend too much time waiting for pages to be loaded.
- The operating system mistakenly increases the degree of multiprogramming, worsening the issue by allocating memory to additional processes.



1. Thrashing severely degrades system performance by overwhelming memory resources with excessive paging operations.
2. The system mistakenly interprets low CPU utilization as a sign to increase the number of processes, further exacerbating the issue.

Thrashing - Causes

Why Does Thrashing Occur?



Causes of Thrashing:

- High multiprogramming level without sufficient memory allocation.
- Poor page replacement strategies causing frequent evictions of required pages.
- Programs with high working set variability leading to sudden memory demands.

Thrashing - Causes
Why Does Thrashing Occur?
Causes of Thrashing:
- High multiprogramming level without sufficient memory allocation.
- Poor page replacement strategies causing frequent evictions of required pages.
- Programs with high working set variability leading to sudden memory demands.

1. Thrashing is often a result of an overloaded system with too many active processes competing for memory.
2. Using an appropriate page replacement strategy and managing the working set of processes can help mitigate thrashing.

Table of Contents

Table of Contents
What Happens if there are no free frames?
Frame allocation
Page Fault
Replacement Algorithms
Conclusion

Conclusion

- When memory utilization increases, the operating system is required to reclaim frames.
- The Operating System determines the number of frames allocated to a process. The change in the number of frames may increase or reduce the number of page faults.
- A page fault occurs when a process attempts to access memory not located within physical memory.
- The various page reclamation algorithms show the challenge of determining the correct number of frames per process and how to reduce the number of page faults that occur.

Conclusion
- When memory utilization increases, the operating system is required to reclaim frames.
- The Operating System determines the number of frames allocated to a process. The change in the number of frames may increase or reduce the number of page faults.
- A page fault occurs when a process attempts to access memory not located within physical memory.
- The various page reclamation algorithms show the challenge of determining the correct number of frames per process and how to reduce the number of page faults that occur.

References I

References I