Memory Management and Process Synchronization Shell

https://github.com/baralsamrat/MSCS630_Project

Deliverable 3

Samrat Baral

February 15, 2025

University of the Cumberlands

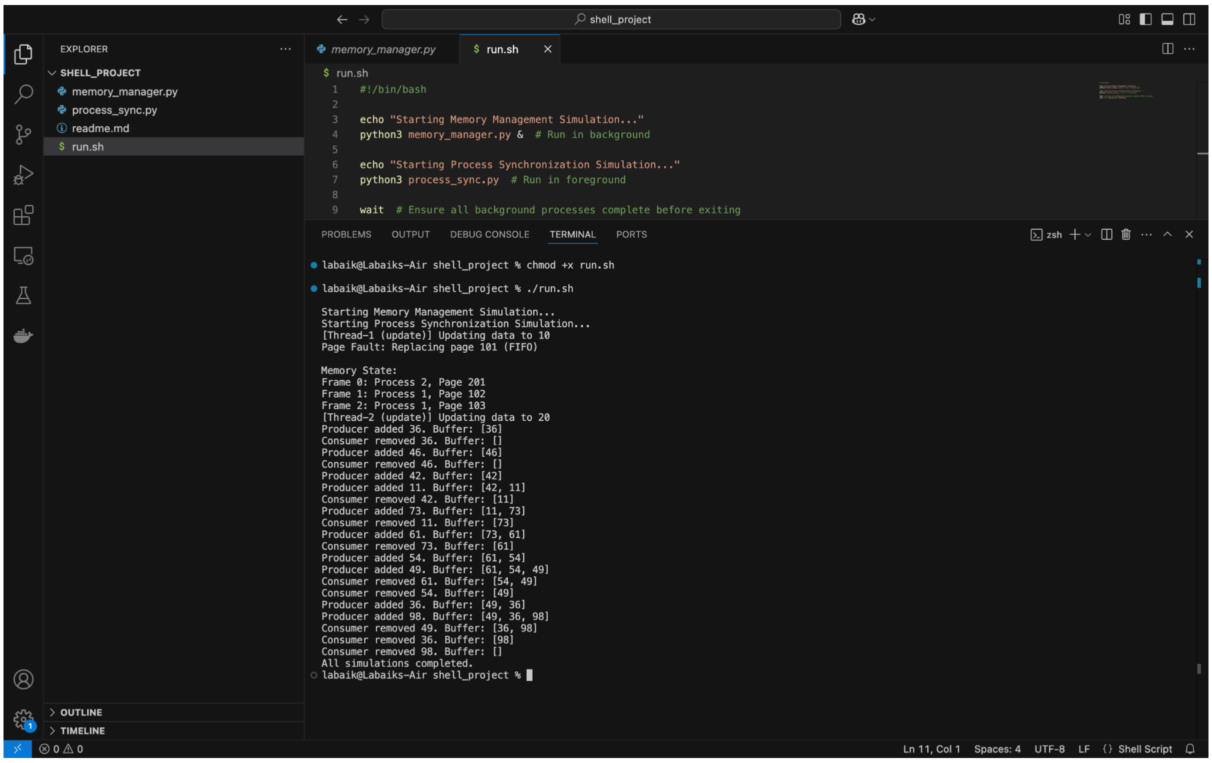
Advanced Operating Systems (MSCS-630-A01)

Primus Vekuh

Code Submission

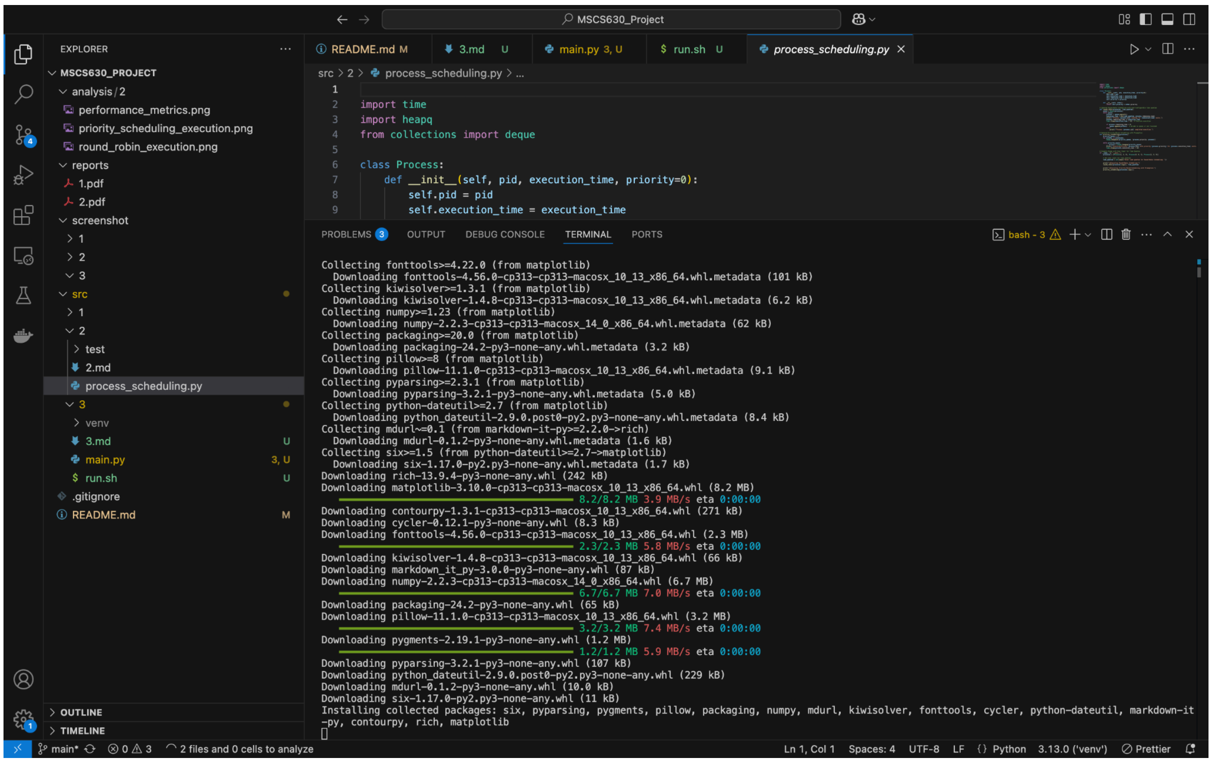
https://github.com/baralsamrat/MSCS630_Project

Screenshots:



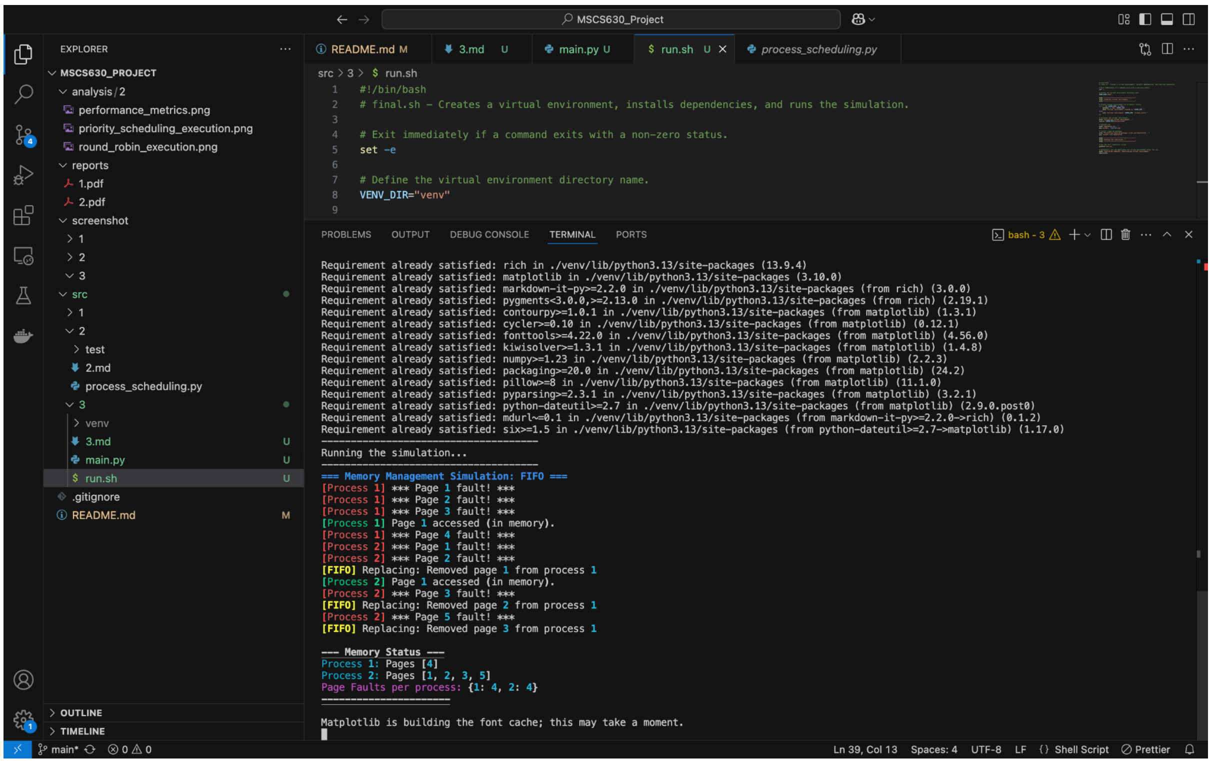
```
#!/bin/bash

echo "Starting Memory Management Simulation..."
python3 memory_manager.py &  # Run in background

echo "Starting Process Synchronization Simulation..."
python3 process_sync.py  # Run in foreground

wait  # Ensure all background processes complete before exiting
```

```
labaik@Labaiks-Air shell_project % chmod +x run.sh

labaik@Labaiks-Air shell_project % ./run.sh

Starting Memory Management Simulation...
Starting Process Synchronization Simulation...
[Thread-1 (update)] Updating data to 10
Page Fault: Replacing page 101 (FIFO)

Memory State:
Frame 0: Process 2, Page 201
Frame 1: Process 1, Page 102
Frame 2: Process 1, Page 103
[Thread-2 (update)] Updating data to 20
Producer added 36. Buffer: [36]
Consumer removed 36. Buffer: []
Producer added 46. Buffer: [46]
Consumer removed 46. Buffer: []
Producer added 42. Buffer: [42]
Producer added 11. Buffer: [42, 11]
Consumer removed 42. Buffer: [11]
Producer added 73. Buffer: [11, 73]
Consumer removed 11. Buffer: [73]
Producer added 61. Buffer: [73, 61]
Consumer removed 73. Buffer: [61]
Producer added 54. Buffer: [61, 54]
Producer added 49. Buffer: [61, 54, 49]
Consumer removed 61. Buffer: [54, 49]
Consumer removed 54. Buffer: [49]
Producer added 36. Buffer: [49, 36]
Producer added 98. Buffer: [49, 36, 98]
Consumer removed 49. Buffer: [36, 98]
Consumer removed 36. Buffer: [98]
Consumer removed 98. Buffer: []
All simulations completed.
labaik@Labaiks-Air shell_project %
```



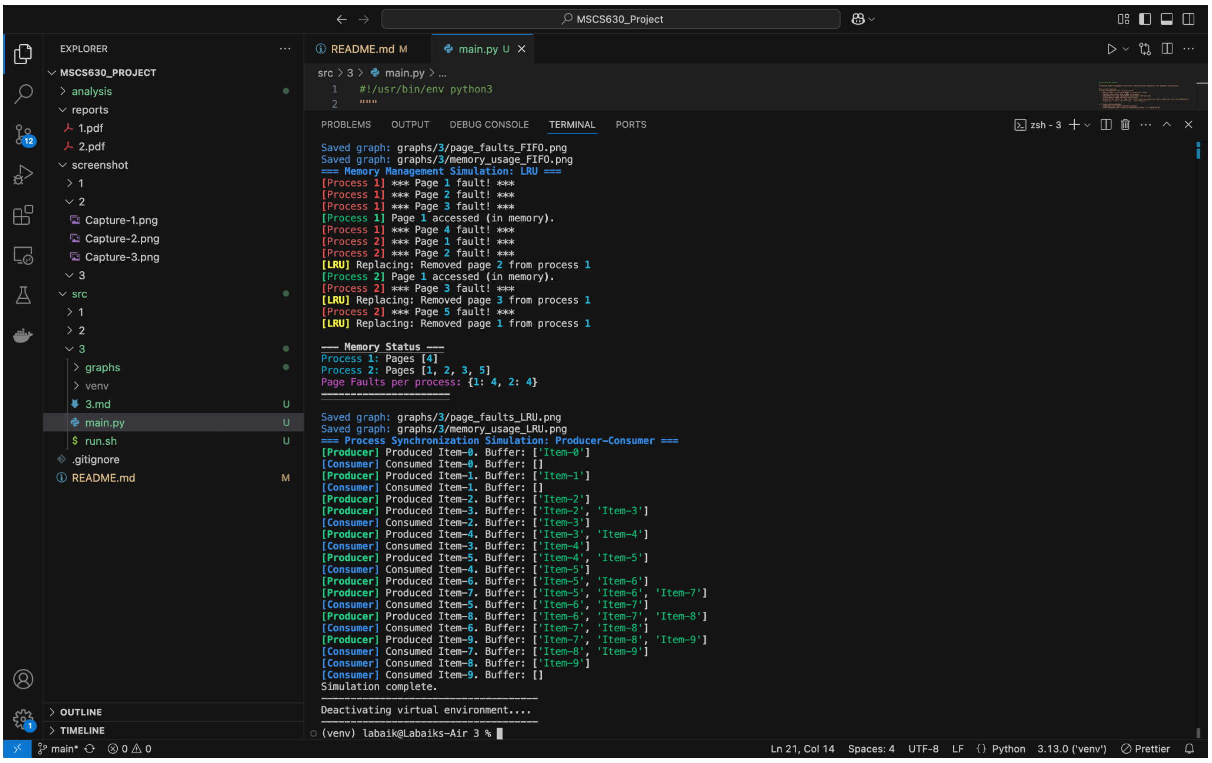
```

import time
import heapq
from collections import deque

class Process:
    def __init__(self, pid, execution_time, priority=0):
        self.pid = pid
        self.execution_time = execution_time
```

```
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.56.0-cp313-cp313-macosx_10_13_x86_64.whl.metadata (101 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.8-cp313-cp313-macosx_10_13_x86_64.whl.metadata (6.2 kB)
Collecting numpy>=1.23 (from matplotlib)
  Downloading numpy-2.2.3-cp313-cp313-macosx_14_0_x86_64.whl.metadata (62 kB)
Collecting packaging>=20.0 (from matplotlib)
  Downloading packaging-24.2-py3-none-any.whl.metadata (3.2 kB)
Collecting pillow>=8 (from matplotlib)
  Downloading pillow-11.1.0-cp313-cp313-macosx_10_13_x86_64.whl.metadata (9.1 kB)
Collecting pyparsing>=2.3.1 (from matplotlib)
  Downloading pyparsing-3.2.1-py3-none-any.whl.metadata (5.0 kB)
Collecting python-dateutil>=2.7 (from matplotlib)
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadata (8.4 kB)
Collecting mdurl~=0.1 (from markdown-it-py>=2.2.0->rich)
  Downloading mdurl-0.1.2-py3-none-any.whl.metadata (1.6 kB)
Collecting six>=1.5 (from python-dateutil>=2.7->matplotlib)
  Downloading six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Downloading rich-13.9.4-py3-none-any.whl (242 kB)
Downloading matplotlib-3.10.0-cp313-cp313-macosx_10_13_x86_64.whl (8.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.2/8.2 MB 3.9 MB/s eta 0:00:00
Downloading contourpy-1.3.1-cp313-cp313-macosx_10_13_x86_64.whl (271 kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.56.0-cp313-cp313-macosx_10_13_x86_64.whl (2.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 5.8 MB/s eta 0:00:00
Downloading kiwisolver-1.4.8-cp313-cp313-macosx_10_13_x86_64.whl (66 kB)
Downloading markdown_it_py-3.0.0-py3-none-any.whl (87 kB)
Downloading numpy-2.2.3-cp313-cp313-macosx_14_0_x86_64.whl (6.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.7/6.7 MB 7.0 MB/s eta 0:00:00
Downloading packaging-24.2-py3-none-any.whl (65 kB)
Downloading pillow-11.1.0-cp313-cp313-macosx_10_13_x86_64.whl (3.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 7.4 MB/s eta 0:00:00
Downloading pygments-2.19.1-py3-none-any.whl (1.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 5.9 MB/s eta 0:00:00
Downloading pyparsing-3.2.1-py3-none-any.whl (107 kB)
Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
Downloading mdurl-0.1.2-py3-none-any.whl (10.0 kB)
Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, pyparsing, pygments, pillow, packaging, numpy, mdurl, kiwisolver, fonttools, cycler, python-dateutil, markdown-it
-py, contourpy, rich, matplotlib
```



```
#!/bin/bash
# final.sh - Creates a virtual environment, installs dependencies, and runs the simulation.

# Exit immediately if a command exits with a non-zero status.
set -e

# Define the virtual environment directory name.
VENV_DIR="venv"

```

```
Requirement already satisfied: rich in ./venv/lib/python3.13/site-packages (13.9.4)
Requirement already satisfied: matplotlib in ./venv/lib/python3.13/site-packages (3.10.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in ./venv/lib/python3.13/site-packages (from rich) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in ./venv/lib/python3.13/site-packages (from rich) (2.19.1)
Requirement already satisfied: contourpy>=1.0.1 in ./venv/lib/python3.13/site-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in ./venv/lib/python3.13/site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in ./venv/lib/python3.13/site-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in ./venv/lib/python3.13/site-packages (from matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in ./venv/lib/python3.13/site-packages (from matplotlib) (2.2.3)
Requirement already satisfied: packaging>=20.0 in ./venv/lib/python3.13/site-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in ./venv/lib/python3.13/site-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in ./venv/lib/python3.13/site-packages (from matplotlib) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in ./venv/lib/python3.13/site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: mdurl~=0.1 in ./venv/lib/python3.13/site-packages (from markdown-it-py>=2.2.0->rich) (0.1.2)
Requirement already satisfied: six>=1.5 in ./venv/lib/python3.13/site-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
-------------------------------------
Running the simulation...
-------------------------------------
=== Memory Management Simulation: FIFO ===
[Process 1] *** Page 1 fault! ***
[Process 1] *** Page 2 fault! ***
[Process 1] *** Page 3 fault! ***
[Process 1] Page 1 accessed (in memory).
[Process 1] *** Page 4 fault! ***
[Process 2] *** Page 1 fault! ***
[Process 2] *** Page 2 fault! ***
[FIFO] Replacing: Removed page 1 from process 1
[Process 2] Page 1 accessed (in memory).
[Process 2] *** Page 3 fault! ***
[FIFO] Replacing: Removed page 2 from process 1
[Process 2] *** Page 5 fault! ***
[FIFO] Replacing: Removed page 3 from process 1

--- Memory Status ---
Process 1: Pages [4]
Process 2: Pages [1, 2, 3, 5]
Page Faults per process: {1: 4, 2: 4}
----------------------

Matplotlib is building the font cache; this may take a moment.
```



```
#!/usr/bin/env python3
"""
```

```
Saved graph: graphs/3/page_faults_FIFO.png
Saved graph: graphs/3/memory_usage_FIFO.png
=== Memory Management Simulation: LRU ===
[Process 1] *** Page 1 fault! ***
[Process 1] *** Page 2 fault! ***
[Process 1] *** Page 3 fault! ***
[Process 1] Page 1 accessed (in memory).
[Process 1] *** Page 4 fault! ***
[Process 2] *** Page 1 fault! ***
[Process 2] *** Page 2 fault! ***
[LRU] Replacing: Removed page 2 from process 1
[Process 2] Page 1 accessed (in memory).
[Process 2] *** Page 3 fault! ***
[LRU] Replacing: Removed page 3 from process 1
[Process 2] *** Page 5 fault! ***
[LRU] Replacing: Removed page 1 from process 1

--- Memory Status ---
Process 1: Pages [4]
Process 2: Pages [1, 2, 3, 5]
Page Faults per process: {1: 4, 2: 4}
----------------------

Saved graph: graphs/3/page_faults_LRU.png
Saved graph: graphs/3/memory_usage_LRU.png
=== Process Synchronization Simulation: Producer-Consumer ===
[Producer] Produced Item-0. Buffer: ['Item-0']
[Consumer] Consumed Item-0. Buffer: []
[Producer] Produced Item-1. Buffer: ['Item-1']
[Consumer] Consumed Item-1. Buffer: []
[Producer] Produced Item-2. Buffer: ['Item-2']
[Producer] Produced Item-3. Buffer: ['Item-2', 'Item-3']
[Consumer] Consumed Item-2. Buffer: ['Item-3']
[Producer] Produced Item-4. Buffer: ['Item-3', 'Item-4']
[Consumer] Consumed Item-3. Buffer: ['Item-4']
[Producer] Produced Item-5. Buffer: ['Item-4', 'Item-5']
[Consumer] Consumed Item-4. Buffer: ['Item-5']
[Producer] Produced Item-6. Buffer: ['Item-5', 'Item-6']
[Producer] Produced Item-7. Buffer: ['Item-5', 'Item-6', 'Item-7']
[Consumer] Consumed Item-5. Buffer: ['Item-6', 'Item-7']
[Producer] Produced Item-8. Buffer: ['Item-6', 'Item-7', 'Item-8']
[Consumer] Consumed Item-6. Buffer: ['Item-7', 'Item-8']
[Producer] Produced Item-9. Buffer: ['Item-7', 'Item-8', 'Item-9']
[Consumer] Consumed Item-7. Buffer: ['Item-8', 'Item-9']
[Consumer] Consumed Item-8. Buffer: ['Item-9']
[Consumer] Consumed Item-9. Buffer: []
Simulation complete.
-------------------------------------
Deactivating virtual environment....
-------------------------------------
(venv) labaik@Labaiks-Air 3 %
```

Memory Management and Process Synchronization Shell

This memory management simulation provides a simplified model of a paging system. It simulates how multiple processes are allocated pages within a limited number of frames. The simulation addresses key aspects of memory management, including page faults and page replacement.

Page Faults occur when a process attempts to access a page that is not currently present in memory. This triggers a page fault, and the required page must be loaded from secondary storage into memory.

Page Replacement Algorithms When memory is full and a page fault occurs, a page replacement algorithm is used to select a victim page to be evicted from memory to make space for the new page. The simulation implements two common page replacement algorithms:

> **FIFO (First-In-First-Out)** algorithm replaces the page that has been in memory for the longest time. It is straightforward to implement but may not always be the most efficient.
>
> **LRU (Least Recently Used)** algorithm replaces the page that has not been accessed for the longest time. It is often more efficient than FIFO and is implemented using an OrderedDict for efficient tracking of page access times.

Allocation and Replacement

1. **Page Request:** When a process requests a page, the simulation checks if the page is already in memory.
2. **Page Hit:** If the page is found in memory, it is marked as accessed (for LRU) and no page fault occurs.
3. **Page Fault:** If the page is not in memory, a page fault is logged.
4. **Page Replacement (if memory is full):** If memory is full, a victim page is selected for replacement based on the chosen algorithm (FIFO or LRU). The new page is then loaded into the frame previously occupied by the victim page.

Tracking the simulation keeps track of the pages allocated to each process and maintains a count of page faults for each process. This information can be used to analyze the performance of the different page replacement algorithms.

Real-world operating systems employ more sophisticated memory management techniques, including hardware support, virtual memory, and more complex page replacement algorithms. This simulation simplifies these complexities to focus on the fundamental concepts of page faults and page replacement.

Potential Enhancements

- **Additional Page Replacement Algorithms:** The simulation could be extended to include other page replacement algorithms, such as the Clock algorithm or the Second-Chance algorithm.
- **Variable Page Sizes:** The current simulation assumes a fixed page size. Allowing for variable page sizes would make the simulation more realistic.
- **Memory Hierarchy:** The simulation could be expanded to include multiple levels of memory, such as cache and main memory, to simulate a more complete memory hierarchy.
- **Thrashing:** The simulation could be modified to demonstrate the concept of thrashing, where a system spends excessive time swapping pages in and out of memory.
- **Demand Paging:** The simulation could be enhanced to implement demand paging, where pages are only loaded into memory when they are actually accessed by a process.

Synchronization Simulation

Core Concepts

- **Producer-Consumer Problem:** This classic synchronization problem involves two types of threads: producers, which generate data and place it into a shared buffer, and consumers, which retrieve data from the buffer and process it. The challenge is to ensure that the producers and consumers cooperate correctly, preventing issues like buffer overruns or underruns.
- **Race Conditions:** These occur when multiple threads access a shared resource concurrently, leading to unpredictable and erroneous results. In the context of the Producer-Consumer problem, a race condition could arise if both a producer and a consumer try to modify the buffer simultaneously.

Synchronization Mechanisms

- **Semaphores:** Semaphores are signaling mechanisms that maintain a count and allow threads to block until a certain condition is met. They are commonly used in the Producer-Consumer problem to control access to the buffer and signal the availability of data.
- **Mutexes:** Mutexes (short for "mutual exclusion") are locking mechanisms that ensure that only one thread can access a critical section of code at a time. They are used to prevent race conditions by providing exclusive access to shared resources.

Solution Components

- **Mutual Exclusion:** This is the fundamental principle of preventing race conditions. In the Producer-Consumer problem, mutual exclusion ensures that only one thread (either a producer or a consumer) can access the buffer at any given time. This is typically achieved using a mutex.

- **Signaling:** Signaling mechanisms are used to coordinate the actions of the producer and consumer threads. In the context of the Producer-Consumer problem, semaphores are often used to signal when the buffer is empty or full.
- **Semaphore "empty":** This semaphore keeps track of the number of empty slots in the buffer. It is initialized to the buffer's capacity and decremented by the producer when an item is added. The consumer increments it when an item is removed.
- **Semaphore "full":** This semaphore keeps track of the number of full slots in the buffer. It is initialized to zero and incremented by the producer when an item is added. The consumer decrements it when an item is removed.
- **Mutex:** A mutex is used to ensure that only one thread can access the buffer at a time, providing the necessary mutual exclusion.

Operational Flow

1. **Producer:**
   - Acquires the mutex to ensure exclusive access to the buffer.
   - Waits on the "empty" semaphore to ensure there is space in the buffer.
   - Places an item in the buffer.
   - Signals the "full" semaphore to indicate that an item is available.
   - Releases the mutex.
2. **Consumer:**
   - Acquires the mutex to ensure exclusive access to the buffer.
   - Waits on the "full" semaphore to ensure there is an item in the buffer.
   - Removes an item from the buffer.
   - Signals the "empty" semaphore to indicate that there is space available.
   - Releases the mutex.

Advantages of Semaphores and Mutexes

- **Efficient Synchronization:** Semaphores and mutexes provide efficient mechanisms for synchronizing the actions of multiple threads, preventing race conditions and ensuring correct program execution.
- **Flexibility:** They can be used in a variety of synchronization scenarios, making them versatile tools for concurrent programming.
- **Portability:** Semaphores and mutexes are supported by most operating systems, making them portable across different platforms.

Key Considerations

- **Deadlocks:** Care must be taken to avoid deadlocks, which can occur if threads are waiting on each other in a circular fashion.
- **Starvation:** Starvation can occur if a thread is perpetually denied access to a resource.

Challenges and Improvements:

Challenges and Solutions

- **Tracking Access Times:** Initially, tracking access times to determine the Least Recently Used (LRU) page was implemented using timestamps and required iterating through the entire list of pages (O(n) complexity). This was optimized by utilizing an OrderedDict, which maintains keys in order of insertion and access, reducing the time complexity to O(1).
- **Thread Coordination:** The Producer-Consumer model, where one thread adds pages (Producer) and another removes pages (Consumer), introduced the potential for deadlocks. Careful synchronization using locks or semaphores was implemented to ensure threads did not block each other indefinitely.

Potential Improvements

- **Dynamic Process Creation and Varying Page Sizes:**
  - The current simulation could be extended to allow for the dynamic creation and termination of processes during execution, simulating a more realistic operating system environment.
  - Additionally, incorporating varying page sizes would introduce another layer of complexity and more accurately reflect real-world memory management scenarios.
- **Implementation of Other Synchronization Problems:**
  - The framework developed for the page replacement simulation could be leveraged to implement and visualize other classic synchronization problems, such as the Dining Philosophers problem or the Reader-Writer problem. This would provide a valuable educational tool for understanding concurrent programming concepts.
- **Enhanced Logging and Performance Metrics:**
  - The logging functionality of the simulation could be enhanced to track a wider range of performance metrics, including:
    - Total execution time
    - Page fault rate
    - Replacement counts for different page replacement algorithms
    - CPU utilization
    - Context switch overhead
  - This data could then be used to analyze the efficiency of different page replacement algorithms under various conditions and workload scenarios.

By implementing these potential improvements, the page replacement simulation could be transformed into a more powerful and versatile tool for understanding operating system memory management and concurrent programming concepts.

Conclusion

This project serves as an effective demonstration of core operating system principles, notably memory management through paging with FIFO and optimized LRU algorithms, and process synchronization using the Producer-Consumer model. The simulation offers valuable insights into the ways these mechanisms work to maintain system stability and enhance overall performance.