Queen's School of Computing
CISC324 – Winter 2024
Instructor: Dr. Anwar Hossain
Lab 5 - Due Date: April 6, 2024

Lab 5: Multiprogramming Memory Management with Working Set

Welcome to the Multiprogramming Memory Management with Working Set lab! In this lab, you will explore the implementation of a multiprogramming environment with a focus on memory management using the working set principle. The lab is designed to deepen your understanding of operating system concepts, including page replacement algorithms, TLB caching, and the working set model. This lab is based on the last lab but with adding the working set model.

Background and Outcomes

The goal of this assignment is to deepen your understanding of the fundamental concepts related to memory management in operating systems, specifically page tables, page frames, page replacement algorithms, and working sets concept.

Code Overview

As usual, you will be given a boilerplate Python code to start with and you're required to complete the missing lines, i.e., the lines containing the word "TODO".

In this section, we will explain the different parts of the code and how to use each part.

MemoryPage Class

The MemoryPage class represents a memory page with a virtual address, content, and a timestamp indicating the last time the page was referenced.

```python
class MemoryPage:
    def __init__(self, virtual_address, content):
        self.content = content
        self.virtual_address = virtual_address
        self.last_referenced_time = time.time()
```

PageTable Class

The PageTable class maintains a mapping between virtual pages and physical frames. It includes methods to check if a page exists, retrieve the physical frame for a virtual page, and remove an entry from the page table.

```python
class PageTable:
    def __init__(self):
        self.table = {}

    def map_page(self, virtual_page, physical_frame) -> bool:
        # ...

    def get_frame(self, virtual_page):
        # ...

          def remove_page_table_entry(self, frame_index):
              # ...
```

TLBCache Class

The `TLBCache` class simulates the Translation Lookaside Buffer (TLB) cache. It maintains a cache of virtual-to-physical page mappings, supporting lookup and insertion operations.

- It has been added to the code to enhance memory management and reduce the need for expensive page table lookups.
- The TLB Cache includes methods for looking up virtual pages and inserting mappings.
- The TLB Cache size can be configured to suit the available memory resources.
- Your will use the TLB Cache in the code. Don't worry, we've added some hints to the code.

```python
class TLBCache:
    def __init__(self, size):
        self.cache = {}
        self.size = size
        self.queue = deque()

    def lookup(self, virtual_page):
        # ...

          def insert(self, virtual_page, physical_frame):
              # ...
```

Page Frame Simulation (Implemented)

In the code, you'll find a class named 'PageFrame' that represents that physical frames in the memory.

```python
class PageFrame:
    def __init__(self, size: int):
        self.frames = [None] * size

    def allocate_frame(self, page_content: MemoryPage):
        # ...

    def deallocate_frame(self, frame_index):
        # ...
```

1- 'allocate_frame': allocates a page to a physical frame. If there is no free physical frames available, the function will return -1.
2- 'deallocate_frame': given a frame index, this function deallocates a physical frame.

WorkingSetPageReplacementAlgorithm Class (NOT Implemented)

Your job is to complete the implementation of WorkingSetPageReplacementAlgorithm page replacement algorithm. The WorkingSetPageReplacementAlgorithm class implements a page replacement algorithm based on the working set principle. It checks the TLB cache, page table, and handles page replacement when necessary.

```
class WorkingSetPageReplacementAlgorithm:
    def __init__(self, page_frame: PageFrame, page_table: PageTable, tlb_cache: TLBCache, time_window: float):
        # ...

    def try_allocate(self, new_page):
        # ...

    def map_page(self, virtual_page, physical_frame) -> bool:
        # ...

        def replace_page(self, new_page):
            # ...
```

MultiprogrammingMemoryManager Class

The MultiprogrammingMemoryManager class manages multiple programs, each with its set of pages and a separate instance of the WorkingSetPageReplacementAlgorithm.

Queen's School of Computing
CISC324 – Winter 2024
Instructor: Dr. Anwar Hossain
Lab 5 - Due Date: April 6, 2024

```python
class MultiprogrammingMemoryManager:
    def __init__(self, num_programs: int, program_pages: List[List[MemoryPage]], num_frames: int, time_window: float):
        # ...

    def simulate_memory_management(self):
        # ...
```

Your Task

Your task is to complete the implementation of the WorkingSetPageReplacementAlgorithm class. Follow the TODO comments in the code to guide your implementation. The main components to complete include:

1- Check if there's an available frame:
Implement the code to check if there's an available frame using the try_allocate method.
2- Get the current time and identify the working set:
Implement the code to get the current time and identify the working set based on the specified time window.
3- Replace the page using the working set:
Replace the page outside the working set with the least recently used page inside the working set.
4- Complete the remaining TODOs:
Complete the remaining TODOs related to removing page table entries, deallocating old pages, and updating the TLB cache.

Prepare your environment!

IF YOU'VE ALREADY DONE THIS PART PREVIOUSLY, PLEASE SKIP IT.

This code can work on any environment (Linux, Windows, MacOS, etc.) so please feel free to search for how to install Python on your machine.

IDEs that you can use (feel free to use any other IDE)

1. Visual Studio Community Edition for Windows-based machines only (it's something different from VS Code).
2. Visual Studio Code (Can work on any operating system).
3. PyCharm (Can work on any operating system).

Install Python on Your Machine

There are too many methods to install Python (feel free to use any other method):

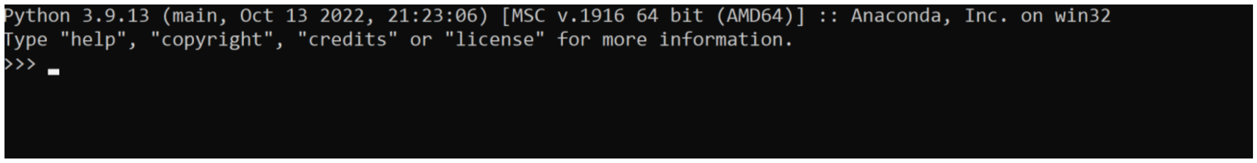
1. Download and install Anaconda (which works on any operating system).

2. Download and install Python from the official website (works on any operating system).

Required Libraries

We will use “threading” library to implement threads and use semaphores. After installing Python using either method mentioned, make sure that Python is working properly on your machine.

1. Open your command prompt or terminal if you’re using Mac/Linux.
2. Type the command python.
3. Check the output on the console screen. You should be seeing something like that:



```
Python 3.9.13 (main, Oct 13 2022, 21:23:06) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

If you don’t see the python version or if you encounter any errors, then there might be something wrong with the installation.

Download the Boilerplate Code

There are two methods to download the boilerplate code:

1. Download the whole repository of the course (Zip file) and open the folder named “Lab 5”.
2. If you are familiar with Git and Git commands, you can clone the repository or you can update your local repository if you already have it cloned from the previous labs.

General Instructions

1. Read the code thoroughly: Understand the provided code, including the classes and their relationships.
2. Follow the TODO comments: Implement the missing parts of the code as indicated by the TODO comments. Use the hints provided to guide your implementation.
3. Test your implementation: After completing the code, run the main function to simulate memory management for multiple programs. Check the output for any error messages or unexpected behavior.
4. Reflect on the results: Analyze the total page faults for each program and the overall total page faults. Consider how the working set principle affects the efficiency of the page replacement algorithm.
5. Write a report (Optional): If required, document your observations, findings, and any modifications you made to the code in a report. Discuss the impact of the working set principle on the multiprogramming environment.

Queen’s School of Computing
CISC324 – Winter 2024
Instructor: Dr. Anwar Hossain
Lab 5 - Due Date: April 6, 2024

What to submit?

1. Place all your source codes in a folder named in the following format:

   324-group**X**-Lab**5**, where X stands for the group number, e.g.: example folder name for group 1 is 324-group**1**-Lab**5**.

2. Put the program output in a file and save it as pdf.
3. Compress the above folder using Zip (the extension must be .zip or .rar).
4. Log into OnQ, locate the lab’s dropbox in OnQ, and upload the compressed folder.