Operating Systems Lab

Fall 2024-25(L59+60)

Student Name:- Parth Suri

Registration No:- 22BDS0116

Class No. :- VL2024250102445

Faculty Name:- Rahul Srivastava

Single Level Paging Implementation in Python

Experiment Title:

Implementation of Single-Level Paging in Memory Management

Objective:

To implement the concept of Single-Level Paging in Python, and to understand the process of logical to physical address translation using paging. Also, calculate important parameters such as the number of pages, page table size, and frame size.

Theory:

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. It divides the program's logical address space into blocks of the same size called pages, and the physical memory into blocks of the same size called frames. The page table maps the logical pages to the physical frames.

In single-level paging, the logical address is divided into two parts:

1. Page number: Which indicates the page in the logical memory.

2. Offset: Which indicates the specific location within the page. The operating system maintains a page table that

stores the mapping between page numbers and frame numbers.

Algorithm:

1. Input Parameters o Total logical address space (in bytes) :
   o Page size (in bytes)
   o Total physical memory size (in bytes)
   o Page table (mapping page numbers to frame numbers)
   2. Steps:
   1. Calculate the Number of Pages:
   Number of Pages=Logical Address Space/Page Size
   2. Calculate the Number of Frames:
   Number of Frames=Physical Memory Size/Page size
   3. Determine Page Table Size:
   Page Table Size=Number of Pages×Size of Page Table Entry
   4. For Each Logical Address:
   ▪ Divide the logical address into:
   ▪ Page number: Page number=Logical AddressPage Size/Page Size
   ▪ Offset: Offset=Logical Address%Page Size
   ▪ Use the page number to look up the corresponding frame number in the page table.
   ▪ Compute the physical address: Physical Address=Frame number×Page Size+Offset 3. Output: The physical address for the given logical address.

Code:-

```
# Function to calculate the number of pages, frames, page table size, and physical address translation
def single_level_paging(logical_address_space, page_size, physical_memory_size, page_table, logical_addresses):
    # Step 1: Calculate the number of pages
    num_pages = logical_address_space // page_size
    print(f"\nNumber of Pages: {num_pages}")

    # Step 2: Calculate the number of frames
    num_frames = physical_memory_size // page_size
    print(f"Number of Frames: {num_frames}")

    # Step 3: Calculate the page table size
    # Assuming each page table entry stores a frame number (integer), typically 4 bytes
    page_table_entry_size = 4  # Size of a page table entry in bytes
    page_table_size = num_pages * page_table_entry_size
    print(f"Page Table Size: {page_table_size} bytes")

    # Step 4: For each logical address, calculate physical address
```

```
    for logical_address in logical_addresses:
        # Step 4.1: Calculate page number and offset
        page_number = logical_address // page_size
        offset = logical_address % page_size

        # Step 4.2: Look up the frame number in the page table
        if page_number < len(page_table):
            frame_number = page_table[page_number]

            # Step 4.3: Calculate the physical address
            physical_address = frame_number * page_size + offset
            print(f"Logical Address: {logical_address} -> Physical Address: {physical_address}")
        else:
            print(f"Logical Address: {logical_address} -> Invalid page number!")

# User input
logical_address_space = int(input("Enter the total logical address space (in bytes): "))
page_size = int(input("Enter the page size (in bytes): "))
```

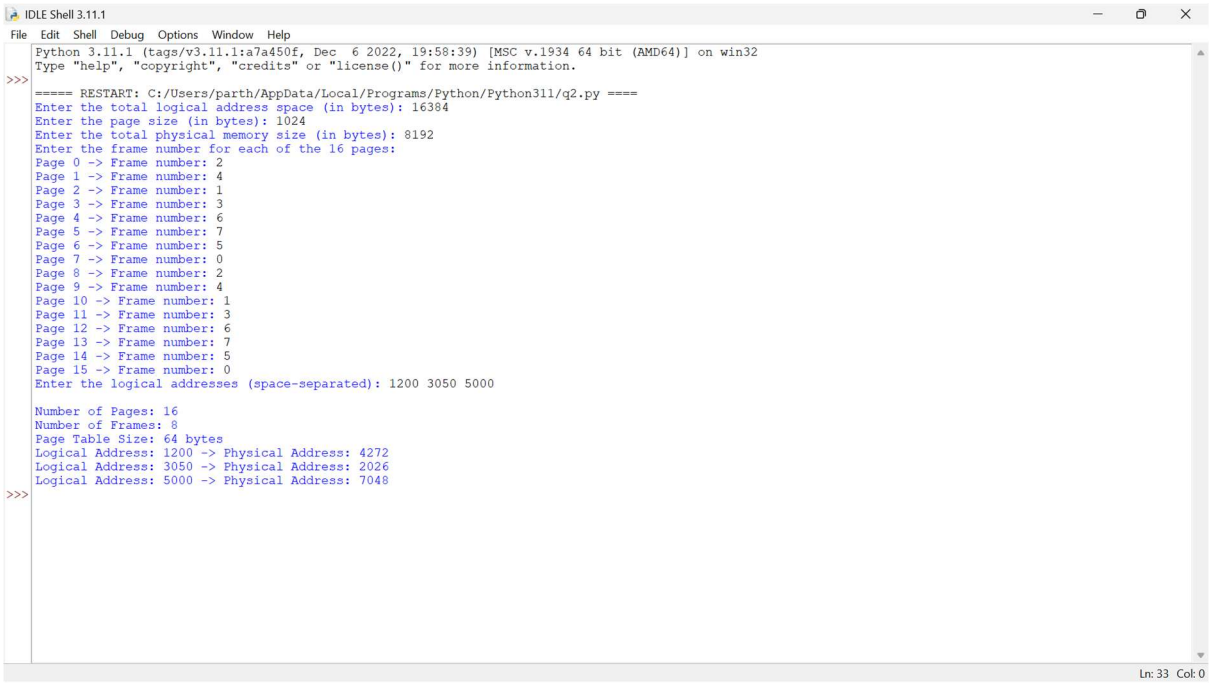
```
physical_memory_size = int(input("Enter the total physical memory size (in bytes): "))

# Input for page table mapping
num_pages = logical_address_space // page_size
page_table = []
print(f"Enter the frame number for each of the {num_pages} pages:")
for i in range(num_pages):
    frame_number = int(input(f"Page {i} -> Frame number: "))
    page_table.append(frame_number)

# Input for logical addresses
logical_addresses = list(map(int, input("Enter the logical addresses (space-separated): ").split()))

# Call the function with user input
single_level_paging(logical_address_space, page_size, physical_memory_size, page_table, logical_addresses)
```

Output:-


IDLE Shell 3.11.1
File Edit Shell Debug Options Window Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec 6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/parth/AppData/Local/Programs/Python/Python311/q2.py ====
Enter the total logical address space (in bytes): 16384
Enter the page size (in bytes): 1024
Enter the total physical memory size (in bytes): 8192
Enter the frame number for each of the 16 pages:
Page 0 -> Frame number: 2
Page 1 -> Frame number: 4
Page 2 -> Frame number: 1
Page 3 -> Frame number: 3
Page 4 -> Frame number: 6
Page 5 -> Frame number: 7
Page 6 -> Frame number: 5
Page 7 -> Frame number: 0
Page 8 -> Frame number: 2
Page 9 -> Frame number: 4
Page 10 -> Frame number: 1
Page 11 -> Frame number: 3
Page 12 -> Frame number: 6
Page 13 -> Frame number: 7
Page 14 -> Frame number: 5
Page 15 -> Frame number: 0
Enter the logical addresses (space-separated): 1200 3050 5000
Number of Pages: 16
Number of Frames: 8
Page Table Size: 64 bytes
Logical Address: 1200 -> Physical Address: 4272
Logical Address: 3050 -> Physical Address: 2026
Logical Address: 5000 -> Physical Address: 7048
>>>
Ln: 33 Col: 0

Multi-Level Paging in Python

Objective:

To implement and understand the working of Multi-Level Paging, where logical addresses are translated into physical addresses using multiple levels of page tables.
Prerequisites:

• Knowledge of memory management concepts, especially paging.

• Familiarity with Python data structures such as lists and dictionaries.

• Understanding of logical to physical address translation.
Algorithm:

Multi-Level Paging

1. Define Constants:
   o Set page size, outer page table size, and inner page table size.
2. Initialize Page Tables:
   o Create an outer page table and multiple inner page tables.
3. Split Logical Address:
   o Break the logical address into three parts:
   ▪ Outer Page Number (higher bits).
   ▪ Inner Page Number (middle bits).
   ▪ Offset (lower bits).
4. Populate Page Tables:

   o Populate outer and inner page tables with frame numbers for each entry.
5. Translate Address:
   o Use the outer page number to access the corresponding inner page table.
   o Use the inner page number to retrieve the frame number.
   o Add the offset to the frame number to get the physical address.
6. Handle Page Fault:
   o If the outer or inner page table entry is missing, generate a page fault.
7. Test the System:
   o Test with various logical addresses and verify the corresponding physical address.

Code:-

```
# Function to simulate multi-level paging
def multi_level_paging(logical_addresses, page_size, outer_table_size, inner_table_size, page_tables):
    for logical_address in logical_addresses:
        # Step 1: Break the logical address into outer page number, inner page number, and offset
        outer_page_bits = outer_table_size.bit_length() - 1  # Number of bits for outer page number
        inner_page_bits = inner_table_size.bit_length() - 1  # Number of bits for inner page number

        # Calculate the offset size (remaining bits after outer and inner page numbers)
        offset_bits = page_size.bit_length() - 1

        # Masking and shifting to extract different parts of the logical address
        outer_page_number = logical_address >> (inner_page_bits + offset_bits)
        inner_page_number = (logical_address >> offset_bits) & (inner_table_size - 1)
        offset = logical_address & (page_size - 1)
```

```
    # Step 2: Check if the outer page table has the inner page table
    if outer_page_number in page_tables:
        inner_page_table = page_tables[outer_page_number]

        # Step 3: Check if the inner page table has the frame number
        if inner_page_number in inner_page_table:
            frame_number = inner_page_table[inner_page_number]

            # Step 4: Compute the physical address
            physical_address = frame_number * page_size + offset
            print(f"Logical Address: {logical_address} -> Physical Address: {physical_address}")
        else:
            print(f"Logical Address: {logical_address} -> Page Fault (inner page table missing entry)")
    else:
        print(f"Logical Address: {logical_address} -> Page Fault (outer page table missing entry)")
```

```
# User input for configuration
page_size = int(input("Enter the page size (in bytes): "))
outer_table_size = int(input("Enter the size of the outer page table: "))
inner_table_size = int(input("Enter the size of the inner page table: "))

# Populate the page tables (outer and inner)
page_tables = {}
print("Enter frame numbers for each outer and inner page:")
for i in range(outer_table_size):
    inner_table = {}
    for j in range(inner_table_size):
        frame_number = int(input(f"Outer Page {i}, Inner Page {j} -> Frame number: "))
        inner_table[j] = frame_number
    page_tables[i] = inner_table

# Input logical addresses
logical_addresses = list(map(int, input("Enter the logical addresses (space-separated): ").split()))
```

Call the function

multi_level_paging(logical_addresses, page_size, outer_table_size, inner_table_size, page_tables)

Output:-