DIVYANSHU MAKHARIA
221070036

ASSIGNMENT 9

AIM: Implement paging technique

THEORY:

Paging is a memory management method that enables processes to use virtual storage. A process has access to the pages it needs without waiting for them to be loaded into physical memory. The technique stores and retrieves data from a computer's secondary or virtual storage (hard drive, SSD, etc.) to the primary storage (RAM).Paging divides both physical memory (RAM) and logical memory (process address space) into fixed-size blocks called frames and pages, respectively. These blocks have the same size, typically a power of 2 for efficiency (e.g., 4KB, 8KB).

Benefits of Paging:

- **Non-contiguous Allocation:** Unlike segmentation, paging allows processes to be loaded into non-contiguous memory frames. This eliminates external fragmentation, a common issue where free memory becomes scattered throughout physical memory, making it unusable for larger processes.
- **Simplified Memory Management:** The OS allocates pages to processes, and a page table translates logical addresses (used by the process) into physical addresses (used by memory). This simplifies memory allocation and reduces complexity.
- **Protection:** The page table controls access permissions for each page. This helps protect memory from unauthorized access and potential crashes.

DIVYANSHU MAKHARIA
221070036

Paging Mechanism:

1. **Division:** The OS divides physical memory into fixed-size frames and the process's logical address space into pages of the same size.
2. **Allocation:** When a process needs memory, the OS allocates one or more free frames and maps them to the process's pages.
3. **Translation:** The Memory Management Unit (MMU) uses the page table to translate logical addresses generated by the CPU into physical addresses. The logical address typically has two parts: a page number and an offset within the page. The MMU uses the page number to find the corresponding frame number in the page table and combines it with the offset to get the final physical address.

Page Faults:

- When a process tries to access a page that is not currently loaded in memory (resident), a page fault occurs.
- The MMU detects the page fault and informs the OS.
- The OS then locates the required page on the secondary storage (like a hard disk) and loads it into an available frame in physical memory.
- The page table is updated to reflect the new mapping.
- The process execution resumes from the point where the page fault occurred.

Page Replacement Algorithms:

- Since physical memory is limited, the OS needs to decide which page to evict when a new page needs to be loaded.
- Page replacement algorithms determine which page in memory is the least recently used (LRU), least frequently used (LFU), or uses some other criteria for replacement.

- This helps minimize the number of page faults and improve overall system performance.

Shared Pages:

- Paging allows for efficient memory sharing between processes.
- Multiple processes can have read-only access to the same page in memory, reducing redundancy and improving memory utilization.
- This is particularly useful for shared libraries or code segments used by multiple programs.

Inverted Page Table:

- A less common alternative to the traditional page table is the inverted page table.
- This approach stores a mapping for each frame in physical memory, indicating which process's page resides in that frame.
- Inverted page tables can be more efficient for systems with a large number of processes but have a higher memory overhead.

Paging vs. Segmentation:

- While paging offers advantages in memory allocation and protection, segmentation provides better flexibility for processes with varying memory requirements for different sections (code, data, stack).
- Some systems even use a combined approach called segmentation with paging to leverage the benefits of both techniques.

DIVYANSHU MAKHARIA
221070036

CODE:

```
frames=[]
page_faults = 0
page_table={}
def paging(page_num,num_frames):
    if page_num not in frames:
            # page_faults += 1
            if len(frames) < num_frames:
                frames.append(page_num)
                page_table[page_num]=len(frames)-1
            else:
                print("Replacing using FIFO algorithm")
                first=frames[0]
                frames.pop(0)
                frames.append(page_num)
                page_table.pop(first)
                page_table[page_num]=0
            print('Miss')

    else:
         print('Page already in Main Memory')
         print('Hit')
    return page_table
logical_address=int(input('Enter size of logical address'))
physical_address=int(input('Enter size of primary adress'))
lst=[]
for i in range(1,physical_address):
    if physical_address%i==0 and logical_address%i==0:
        lst.append(i)
```

```
print(lst)
page_size=int(input('Enter any one page size  '))
if page_size in lst:
    num_pages=int(logical_address/page_size)
    num_frames=int(physical_address/page_size)
    # page_string=[]
    # for i in range(0,num_pages):
    #     print(i)
    #     page_string.append(i)

    while True:
        print('Enter page number in range 0 to ',num_pages-1)
        page_num=int(input())
        if page_num<0 or page_num>=num_pages:
            print('Invalid page number')
        else:
           page_table=paging(page_num,num_frames)
        print("Page Table",page_table)
else:
    print('Not a valid page size')
```

DIVYANSHU MAKHARIA
221070036

OUTPUT SCREENSHOT:

```
PS D:\OS_lab> & C:/Users/Divyanshu/AppData/Local/Microsoft/WindowsApps/pyt
y
Enter size of logical address2000
Enter size of primary adress1000
[1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250, 500]
Enter any one page size  250
Enter page number in range 0 to  7
0
Miss
Page Table {0: 0}
Enter page number in range 0 to  7
-1
Invalid page number
Page Table {0: 0}
Enter page number in range 0 to  7
3
Miss
Page Table {0: 0, 3: 1}
Enter page number in range 0 to  7
4
Miss
```

```
Page Table {0: 0, 3: 1, 4: 2}
Enter page number in range 0 to  7
7
Miss
Page Table {0: 0, 3: 1, 4: 2, 7: 3}
Enter page number in range 0 to  7
3
Page already in Main Memory
Hit
Page Table {0: 0, 3: 1, 4: 2, 7: 3}
Enter page number in range 0 to  7
5
Replacing using FIFO algorithm
Miss
Page Table {3: 1, 4: 2, 7: 3, 5: 0}
Enter page number in range 0 to  7
2
Replacing using FIFO algorithm
Miss
Page Table {4: 2, 7: 3, 5: 0, 2: 0}
Enter page number in range 0 to  7
```

```
Enter page number in range 0 to  7
7
Page already in Main Memory
Hit
Page Table {4: 2, 7: 3, 5: 0, 2: 0}
Enter page number in range 0 to  7
```

CONCLUSION:

Thus, we have successfully learnt about the concept of paging,page replacement algorithms etc. and implemented it.