

**WIA2004 OPERATING SYSTEM**

**LAB 6 PROJECT REPORT**

**PAGING**

**GROUP MEMBERS**

|  |  |  |
| --- | --- | --- |
| **No.** | **Member Name** | **Matric Number** |
| **1.** | **MUHAMMAD IZZAT HAKEEM BIN ROSDI** | **17204856** |
| **2.** | **WAN MUHAMMAD AZIB BIN WAN ABDUL RAHIM** | **17207678** |
| **3.** | **MUHAMMAD AMIRUL AMIN BIN ARMAN** | **17202969** |
| **4.** | **JONE WAN GAN** | **S2031153** |
| **5.** | **LAWRENCE LEROY CHIENG TZE YAO** | **S2018935** |

**OBJECTIVE**

\*Write a program to simulate the paging technique of Memory Management.”

**DESCRIPTION**

Variables to be defined by the user include **memory size, page size, number of processes,** and **size of each process.**

Paging is a technique where the process address space is broken into equal-sized blocks of memory called pages. While empty pages exist, the program allocates each process into pages in the memory. Internal fragmentation occurs when a process size modulo page size is not zero- i.e. there will be a page that is not completely filled.

Each page in logical address is mapped to a frame in physical address. The physical addresses are calculated based on the logical address as well as offset.

The final output is shown in the Paging Table where each page has a corresponding frame, likewise for a physical address and logical address.

**ADVANTAGES OF PAGING MEMORY MANAGEMENT:**

This approach permits the physical address space to be non-contiguous as the process is broken down into pages.

Fast processing speed and relatively simple cum efficient algorithm. No external fragmentation or dynamic rearranging of frames/pages is needed.

Calculation between logical and physical addresses as well as swapping of pages are easily handled by OS as well.

**DISADVANTAGES OF PAGING MEMORY MANAGEMENT:**

Internal fragmentation exists as a waste of memory that could be as high as **pagesize-1**. An example is shown in sample I/O where 33MB takes up 2 full 16MB pages plus an additional page holding only 1MB, hence wasting 15MB on that page.

Another problem is this method requires extra memory space, so is not so favorable for systems with smaller RAM.

**CODE**

#Initialisation of Paging and Page Table/TLB

import random

#utility function

def print\_as\_table(table=[], title="", columns=[]):

    print("\n%s"%title)

    for p in columns:

        print("%s\t" %p, end="")

    for i in range(len(table)):

        print("\n%s\t" %i, end="")

        if isinstance(table[i], list):

            for j in range(len(table[i])):

                print("%s\t\t"%table[i][j], end="")

        else:

            print("%s\t"%table[i], end="")

print()

######################################## RUNNER CODE ########################################

if \_\_name\_\_ == "\_\_main\_\_":

    memory\_size = int(input("Enter RAM memory size(MB): "))

    page\_size = int(input("Enter page size(MB): "))

    process\_count = int(input("Enter process count: "))

    total\_wasted\_space = memory\_size % page\_size #any remainder space is wasted

    memory\_size -= total\_wasted\_space

    total\_pages = memory\_size // page\_size

    memory\_space = []

    page\_table = []

    print("\nAvailable space: %sMB\nEmpty pages: %s\nTotal Wasted space: %sMB\n"

            %(memory\_size, total\_pages, total\_wasted\_space))

    for i in range(process\_count):

        process\_size = int(input("Enter size of process[%d](MB): " %i))

        if process\_size > memory\_size: #overload

            memory\_space.append([process\_size, 0, 0])

            print("Memory full: unable to parse process")

            break

        num\_pages = process\_size // page\_size

        wasted\_space = process\_size % page\_size

        if wasted\_space != 0: #internal fragmentation

            num\_pages += 1

            wasted\_space = page\_size - wasted\_space #actual wasted space

            total\_wasted\_space += wasted\_space

        memory\_size -= num\_pages \* page\_size

        total\_pages -= num\_pages

        memory\_space.append([process\_size, num\_pages, wasted\_space])

        print("\nAvailable space: %sMB\nEmpty pages: %s\nTotal Wasted space: %sMB\n"

            %(memory\_size, total\_pages, total\_wasted\_space))

    print\_as\_table(memory\_space, "Process Table", ["Process", "Process Size", "Pages Used", "Wasted Space"])

#get total number of pages

used\_pages = sum([memory\_space[i][1] for i in range(len(memory\_space))])

#create page-to-frame mapping table

    page\_frame\_table = [i for i in range(used\_pages)]

    random.shuffle(page\_frame\_table)

    #allocate logical address (in MB for simplicity)

    logical\_addresses = [i\*page\_size for i in range(used\_pages)]

    # calculate physical address (in MB) based on logical address/offset

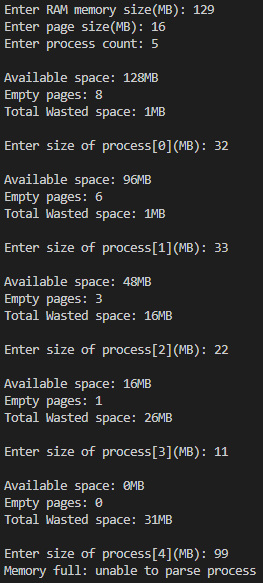
    physical\_addresses = [page\_frame\_table[i]\*(page\_size) + logical\_addresses[i]//(page\_size) for i in page\_frame\_table]

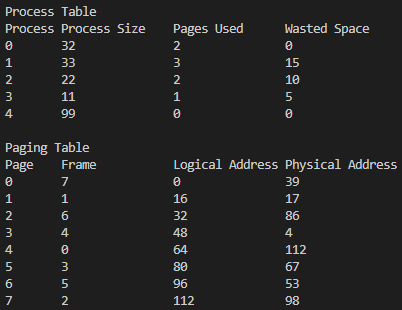
    #zip all tables into 1 paging table

    logical\_physical\_table = [list(x) for x in zip(page\_frame\_table, logical\_addresses, physical\_addresses)]

print\_as\_table(logical\_physical\_table, "Paging Table", ["Page", "Frame", "\tLogical Address", "Physical Address"])

**OUTPUT**

****

****