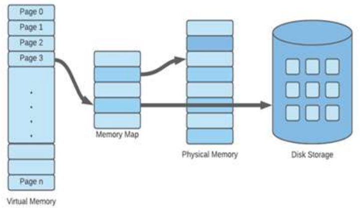
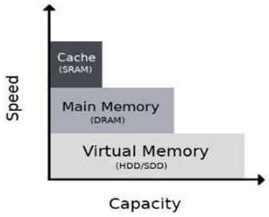
Operating Systems – Virtual Memory Analytics Suite

Name : Vaibhav U Navalagi [1RV22CS222]
Tejas Ganesh Hegde [1RV22CS219]

Introduction to Virtual Memory Analytics Suite :

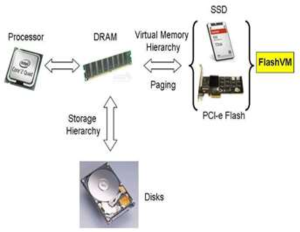
In today's data-driven world, the ability to extract meaningful insights from vast amounts of information is paramount. Enter the Virtual Memory Analytics Suite (VMAS), a cutting-edge solution designed to transform the way organizations analyze and harness their data resources. Furthermore, VMAS prioritizes user-friendliness and accessibility, with an intuitive interface that empowers both data scientists and business users alike to derive actionable insights with ease.

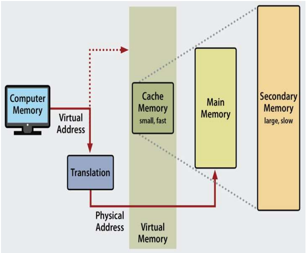




Problem Statement :

Develop a comprehensive Virtual Memory Analytics Suite that employs advanced algorithms to analyze, optimize, and monitor virtual memory usage in computing systems, enhancing overall performance and resource efficiency.

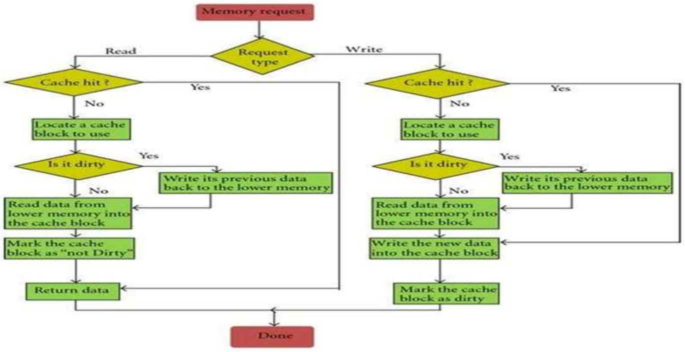




System Architecture :

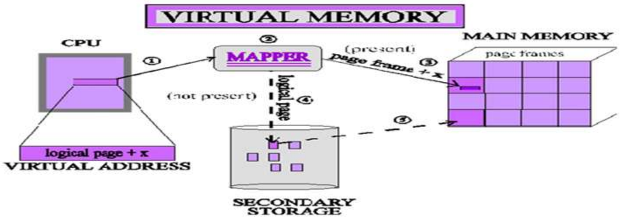
This architecture operates on the principle of virtualization, where the physical memory space is abstracted into smaller, more manageable units known as pages or blocks. Through intelligent memory mapping techniques, VMSA dynamically manages the allocation and retrieval of data, optimizing performance while minimizing latency.

Methodology – Flowchart :

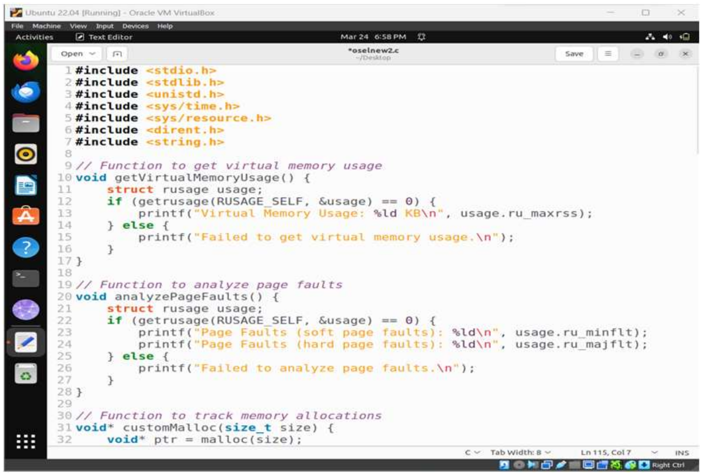

Memory request
Request type
Read
Write
Cache hit ?
Yes
No
Locate a cache block to use
Is it dirty
Yes
No
Write its previous data back to the lower memory
Read data from lower memory into the cache block
Mark the cache block as "not Dirty"
Return data
Cache hit ?
Yes
No
Locate a cache block to use
Is it dirty
Yes
No
Write its previous data back to the lower memory
Read data from lower memory into the cache block
Write the new data into the cache block
Mark the cache block as dirty
Done

Tools and APIs used :

1. C programming language for system-level programming.
2. System calls such as **getrusage** for gathering memory usage statistics.
3. **#include<sys/resources.h>** header file to get all the recently used up processes by the system through kernel.
4. Visualization libraries such as matplotlib or **gnuplot** for generating graphical representations of data.



C Program - 1



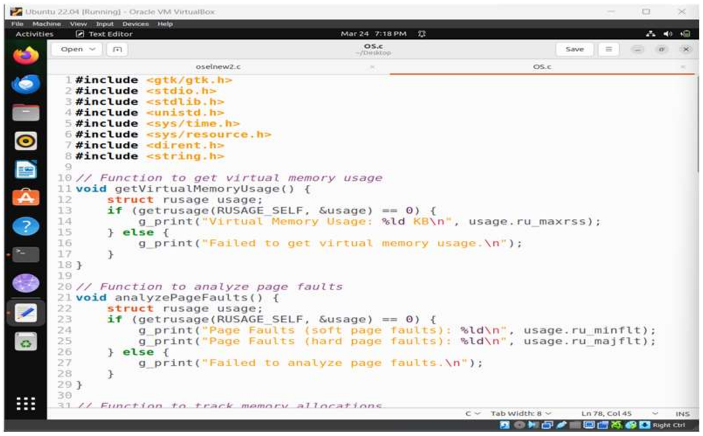
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <dirent.h>
#include <string.h>

// Function to get virtual memory usage
void getVirtualMemoryUsage() {
    struct rusage usage;
    if (getrusage(RUSAGE_SELF, &usage) == 0) {
        printf("Virtual Memory Usage: %ld KB\n", usage.ru_maxrss);
    } else {
        printf("Failed to get virtual memory usage.\n");
    }
}

// Function to analyze page faults
void analyzePageFaults() {
    struct rusage usage;
    if (getrusage(RUSAGE_SELF, &usage) == 0) {
        printf("Page Faults (soft page faults): %ld\n", usage.ru_minflt);
        printf("Page Faults (hard page faults): %ld\n", usage.ru_majflt);
    } else {
        printf("Failed to analyze page faults.\n");
    }
}

// Function to track memory allocations
void* customMalloc(size_t size) {
    void* ptr = malloc(size);
```

C Program - 2



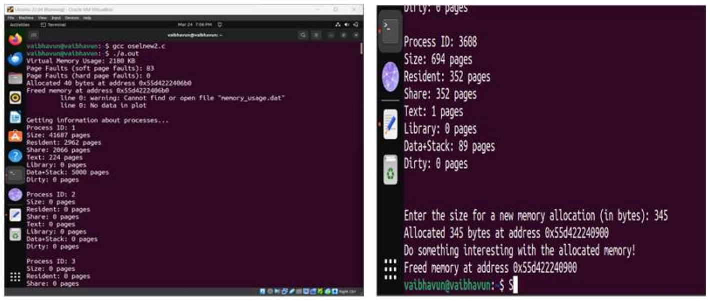
```c
#include <gtk/gtk.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <dirent.h>
#include <string.h>

// Function to get virtual memory usage
void getVirtualMemoryUsage() {
    struct rusage usage;
    if (getrusage(RUSAGE_SELF, &usage) == 0) {
        g_print("Virtual Memory Usage: %ld KB\n", usage.ru_maxrss);
    } else {
        g_print("Failed to get virtual memory usage.\n");
    }
}

// Function to analyze page faults
void analyzePageFaults() {
    struct rusage usage;
    if (getrusage(RUSAGE_SELF, &usage) == 0) {
        g_print("Page Faults (soft page faults): %ld\n", usage.ru_minflt);
        g_print("Page Faults (hard page faults): %ld\n", usage.ru_majflt);
    } else {
        g_print("Failed to analyze page faults.\n");
    }
}

// Function to track memory allocations
```

Output - 1



```
vaibhavun@vaibhavun:~$ gcc oselnew2.c
vaibhavun@vaibhavun:~$ ./a.out
Virtual Memory Usage: 2180 KB
Page Faults (soft page faults): 83
Page Faults (hard page faults): 0
Allocated 40 bytes at address 0x55d4222406b0
Freed memory at address 0x55d4222406b0
         line 0: warning: Cannot find or open file "memory_usage.dat"
         line 0: No data in plot

Getting information about processes...
Process ID: 1
Size: 41687 pages
Resident: 2962 pages
Share: 2066 pages
Text: 224 pages
Library: 0 pages
Data+Stack: 5000 pages
Dirty: 0 pages

Process ID: 2
Size: 0 pages
Resident: 0 pages
Share: 0 pages
Text: 0 pages
Library: 0 pages
Data+Stack: 0 pages
Dirty: 0 pages

Process ID: 3
Size: 0 pages
Resident: 0 pages
Share: 0 pages
```

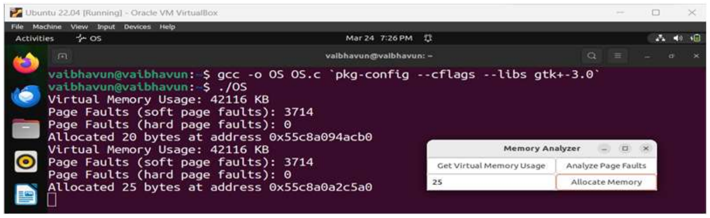
```
Dirty: 0 pages

Process ID: 3608
Size: 694 pages
Resident: 352 pages
Share: 352 pages
Text: 1 pages
Library: 0 pages
Data+Stack: 89 pages
Dirty: 0 pages



Enter the size for a new memory allocation (in bytes): 345
Allocated 345 bytes at address 0x55d422240900
Do something interesting with the allocated memory!
Freed memory at address 0x55d422240900
vaibhavun@vaibhavun:~$ $
```

Output - 2



```
vaibhavun@vaibhavun:~$ gcc -o OS OS.c `pkg-config --cflags --libs gtk+-3.0`
vaibhavun@vaibhavun:~$ ./OS
Virtual Memory Usage: 42116 KB
Page Faults (soft page faults): 3714
Page Faults (hard page faults): 0
Allocated 20 bytes at address 0x55c8a094acb0
Virtual Memory Usage: 42116 KB
Page Faults (soft page faults): 3714
Page Faults (hard page faults): 0
Allocated 25 bytes at address 0x55c8a0a2c5a0
```

Memory Analyzer input field: 25

Applications :

- Performance Monitoring and Optimization
- Capacity Planning and Resource Allocation
- Troubleshooting and Debugging

Guide Information:

Dr. Jyoti Shetty
Assistant Professor