Advanced OS

ASSIGNMENT#2

By Qazi Danish Ayub msds20075

Question

Perform the critical analysis of the virtual memory management of windows operating system and Linux. Explain the role of mmap() in Linux.

Windows:

Windows uses a virtual memory management system that is based on demand paging. When a process is started, the operating system allocates a certain amount of virtual memory for it. The process can then access this memory as if it were actual physical memory. However, the operating system will only allocate physical memory to the process as it is needed. This means that a process can have more virtual memory than physical memory.

Linux:

Linux also uses a virtual memory management system that is based on demand paging. However, Linux uses a feature called mmap() that allows a process to map a file into its virtual memory space. This means that a process can access a file as if it were part of its own memory. This can be useful for processes that need to access large files, such as databases.

The mmap() function in Linux allows a process to map a file or device into its virtual address space. This provides a way for a process to directly access files or devices without going through the kernel's normal file-access functions. It can also be used to create shared memory regions that can be shared between multiple processes.

code to test the above all

```
#include <windows.h>

#include <stdio.h>
```

```
int main()
{
   printf("Windows virtual memory management\n");
   printf("--------------------------------\n");

   // Allocate 1MB of virtual memory.
   void* p = VirtualAlloc(NULL, 1024*1024, MEM_RESERVE, PAGE_READWRITE);
   if (p == NULL)
   {
      printf("Error allocating virtual memory\n");
      return 1;
   }

   // Write some data to the memory.
   memset(p, 0x01, 1024*1024);

   // Free the memory.
   VirtualFree(p, 0, MEM_RELEASE);

   return 0;
}

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>

int main()
{
```

```
printf("Linux virtual memory management\n");
printf("-------------------------------\n");

// Open a file.
FILE* fp = fopen("test.txt", "w+");
if (fp == NULL)
{
   printf("Error opening file\n");
   return 1;
}

// Write some data to the file.
fprintf(fp, "This is a test\n");

// Get the size of the file.
fseek(fp, 0, SEEK_END);
long size = ftell(fp);

// Map the file into memory.
void* p = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED, fileno(fp), 0);
if (p == MAP_FAILED)
{
   printf("Error mapping file into memory\n");
   fclose(fp);
   return 1;
}

// Read the data from the mapped memory.
printf("%s", (char*)p);
```

```
    // Unmap the memory.
    munmap(p, size);

    // Close the file.
    fclose(fp);

    return 0;
}
```