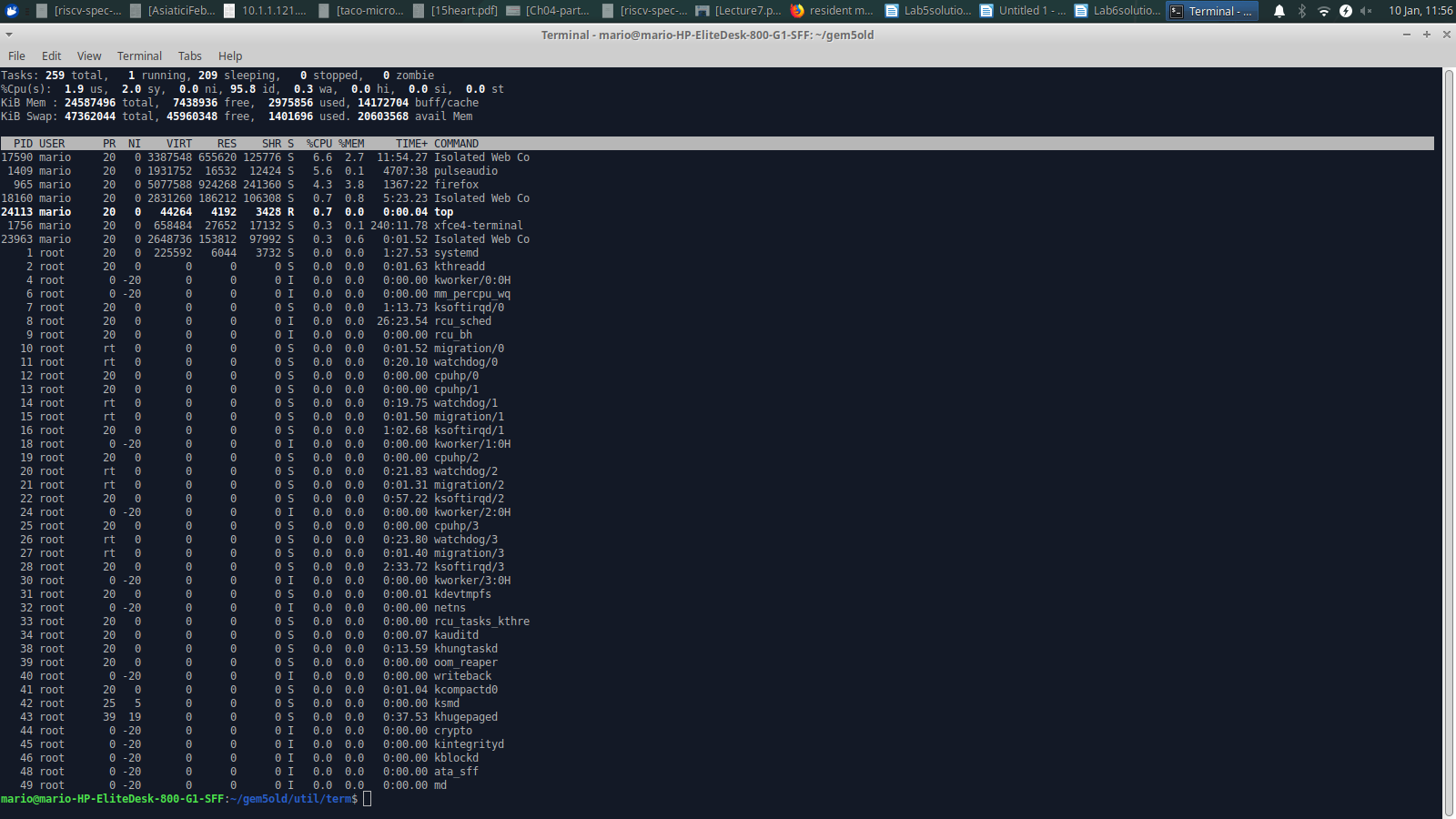
Exam 2 – OSP – 18/01/2024 – Duration 1:30min

Figure 1: A snapshot of top command

**Question 1 (total: 20 marks)**

Considering the previous top screenshot (Figure 1), respond to the following questions:

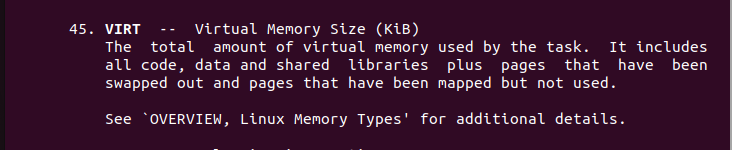
**1a.** In top, RES represents the amount of physical memory a process is using. On the screenshot, for the PID = 17590, how much physical memory is being used? (1 mark)

Answer:

***The PHYSICAL MEMORY being used is 655620 KiB.***

**1b.** You can notice that PID = 17590 shows 3387548 kiB as VIRT. Do man top <enter> and find what the **VIRT** field represents in terms of memory (provide a man screenshot as answer). (2 marks)

Answer:



**1c.** On top’s upper left side (Figure 1), observing the line Kib Mem, you find that this computer has the total amount of memory 24587496 or ~24.59 GB. At this same line (Kib Mem), find and list below the **used** amount of memory by all programs running on this computer. (1 mark)

Answer:

**The amount of used memory by all programs running on this computer is 2975856.**

**1d.** The virtual memory address space is managed via segments, and these are very useful to the programmer. List five different segments of the virtual address space (2 marks). Hint: do man proc.

Answer:

**The virtual memory address space is divided into several segments. Here are five of them:**

1. *Text Segment (Code Segment)*: **This segment contains the executable code of the program.**
2. *Data Segment*: **It includes initialized data and global variables used by the program.**
3. *Heap Segment*: **This is the area where dynamic memory allocation occurs, and it is managed by functions like malloc and free.**
4. *Stack Segment*: **It stores local variables and function call information. The stack grows and shrinks as functions are called and return.**
5. *Memory-Mapped Segment*: **This segment is used for memory-mapped files and shared memory.**

**1e.** Suppose a user would do: cd /proc/17590 <enter> and started to look at the output of cat status | grep Vm as follows:

VmData: 195332 kB

VmStk: 776 kB

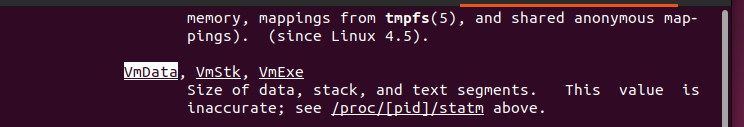
VmExe: 768 kB

VmLib: 176672 kB

VmRSS: 250684 kB

Do man proc <enter> and find what VmData and VmStk are (you can provide a screenshot as answer). (1.5 marks)

Answer:



**1f.** The OS uses the concept of pages as units of management of its virtual memory. To facilitate management, pages are used instead of addresses. These pages are organized in page tables kept in the PCB. As other PCB elements, page tables are maintained in memory. Explain how many memory accesses are required to access a generic page (that is placed in the page table) (2 marks) and what the OS does do to speed them up? (2 marks) (total 4 marks)

Answer:

**When accessing a page in virtual memory, two memory accesses are typically required: one to the Page Table to retrieve the frame number, and another to access the actual physical memory using that frame number. To expedite this process, the operating system utilizes a Translation Lookaside Buffer (TLB) as a cache for frequently used Page Table entries. TLB helps speed up memory access by storing recent translations, reducing the need for repeated accesses to the slower main memory.**

**When a page is already in the TLB (TLB hit), the system can skip accessing the Page Table in memory, optimizing the overall speed of virtual-to-physical address translation.**

**1g.** Refer to Figure 1, on top’s upper left side, you find that top shows **Kib Swap 47362044 (~47.36GB - gigabytes)**.

Assume you have a program running on this computer and this program is using 30GB of memory. Briefly explain how the OS would manage its execution, since the amount of memory it uses is larger than the RAM size (24.59GB). (3 marks)

Answer:

**In a situation where a program's memory usage exceeds the available RAM, the operating system employs a mechanism known as "swapping" to manage the execution. The OS transfers less frequently used or inactive portions of the program's memory from RAM to the swap space on the disk, freeing up space in the RAM for more immediate needs. In this case, with a program using 30GB of memory on a system with only 24.59GB of RAM, the OS would likely swap out some of the program's data to the swap space, utilizing the 47.36GB available in this example. This allows the program to continue running, albeit at a potentially slower pace due to the increased disk I/O involved in swapping data between RAM and the swap space. Swapping helps prevent the program from being terminated due to insufficient physical memory and allows the system to efficiently manage its resources.**

**1h.** Explain the following sentence: the virtual address space needs to be backed up by physical media. (1 mark)

Answer:

**This sentence emphasizes the need for physical storage to support the virtual address space. In a computer system, programs and processes interact with virtual memory addresses, which may not directly correspond to physical locations in RAM. The virtual address space, provided by the operating system, needs to be backed up by physical storage media like RAM or swap space on a hard drive. This ensures that the data associated with virtual addresses has a tangible location in the computer's hardware, allowing for efficient execution of programs by storing and retrieving information from physical memory as needed.**

**1i.** One of widely used algorithms in memory management is *“best-fit”*, highlight on the advantage(s) and disadvantage(s) of *best-fit* algorithm (2 marks).

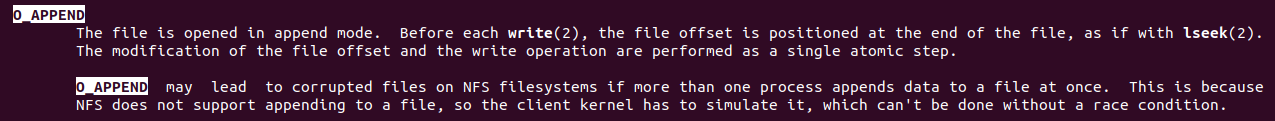
Answer:

**Advantage: Efficiently allocates the smallest available memory block that fits a process, minimizing wasted space.**

**Disadvantage: Fragmentation can still occur, and the search for the best-fit block may lead to slightly higher overhead.**

**1j.** One of the file creation flags (and file status flags) is *“O\_APPEND”;* from the Linux man pages,find the system callthat uses O\_APPEND, provide a screenshot. (1.5 mark)

Answer:



**1k**. List the three possible file permissions in Linux. (1 mark)

Answer:

**In Linux, the three possible file permissions are read(r), write(w), and execute(x).**

**Question 2 (22 marks)**

**2a.** In single-user dedicated systems, user can debug and find out when a program goes into an infinite loop. But in multiuser systems running large numbers of processes, user cannot easily determine that an individual process is not progressing. Can the OS determine whether a process is in an infinite loop? (3 marks) If you were implementing the OS, what would you use to prevent processes in infinite loops from running indefinitely? (2 marks) (total 5 marks)

Answers:

**In a multiuser system with numerous processes, it becomes challenging for individual users to easily determine if a process is stuck in an infinite loop. However, the operating system can employ various mechanisms to detect and handle processes in infinite loops.**

1. ***Detecting Infinite Loops:***

***Timeouts and Timers***: **The OS can set timeouts for processes, and if a process takes longer than expected to complete a certain task, it may be considered stuck.**

***CPU Usage Monitoring***: **Continuous high CPU usage by a process may indicate a potential infinite loop.**

2. ***Preventing Infinite Loops:***

***Watchdog Timer***: **Implementing a watchdog timer can be effective. If a process exceeds a predefined time limit, the watchdog timer can intervene, terminate the process, and alert the system or administrator**.

***Resource Quotas***: **Setting resource limits (CPU time, memory usage) for processes can prevent them from consuming excessive resources and getting stuck.**

**Implementing a combination of these techniques can enhance the OS's ability to detect and handle processes in infinite loops, maintaining system stability and responsiveness in multiuser environments.**

**2b.** List and explain and two OS scheduling policies. (4 marks).

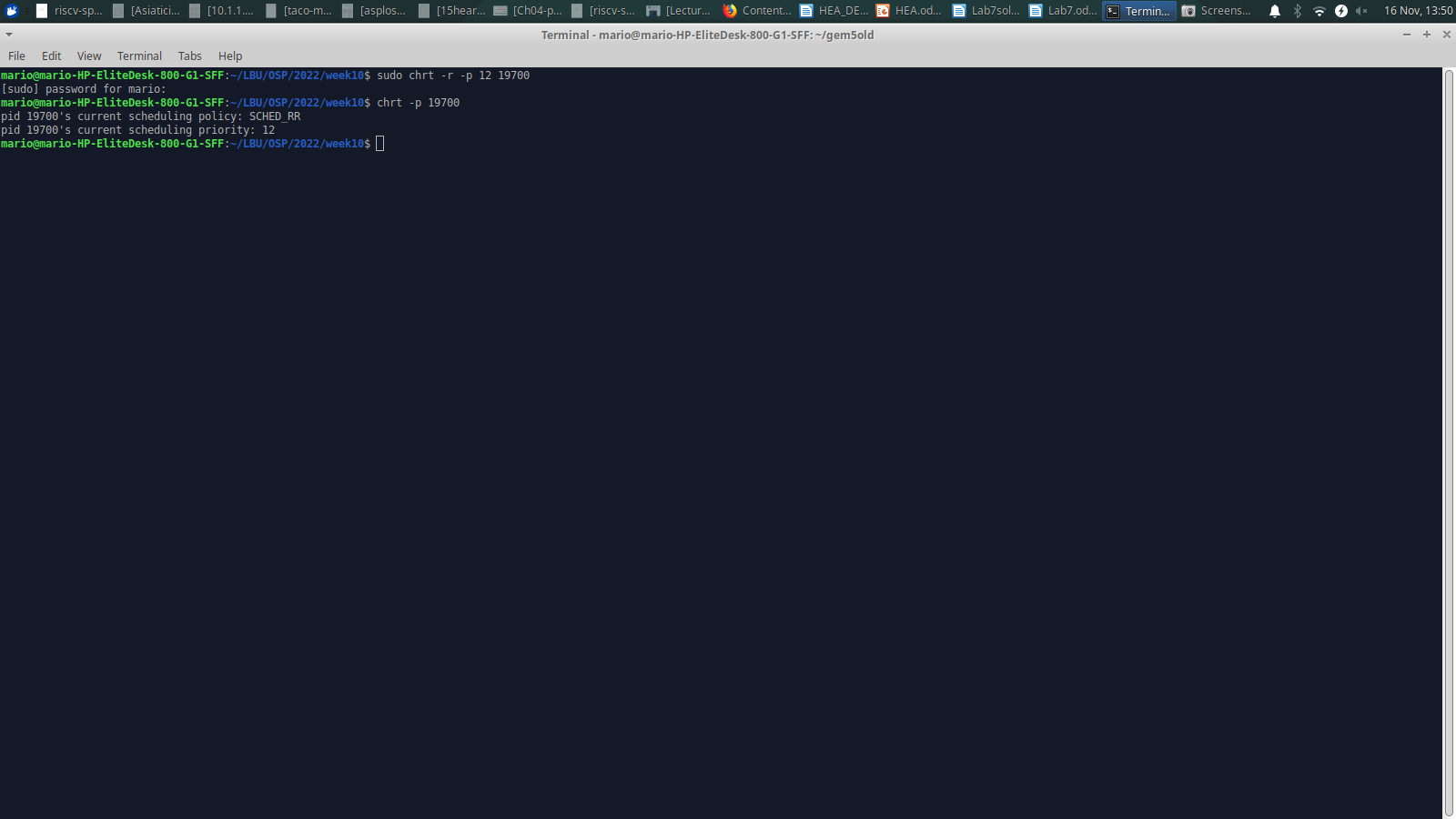
Answer:

**Two OS scheduling policies are:**

1. **Round Robin Scheduling: Allocates fixed time slices to processes, ensuring fairness and preventing any single process from monopolizing resources for too long; however, it may lead to inefficiencies with CPU-bound tasks due to frequent context switches**.

2. **Shortest Job First: The process with the shortest burst time is selected first. Minimizes waiting time and turnaround time. Requires knowledge of burst times in advance which is often not available.**

**2c.** Consider the screenshot below.

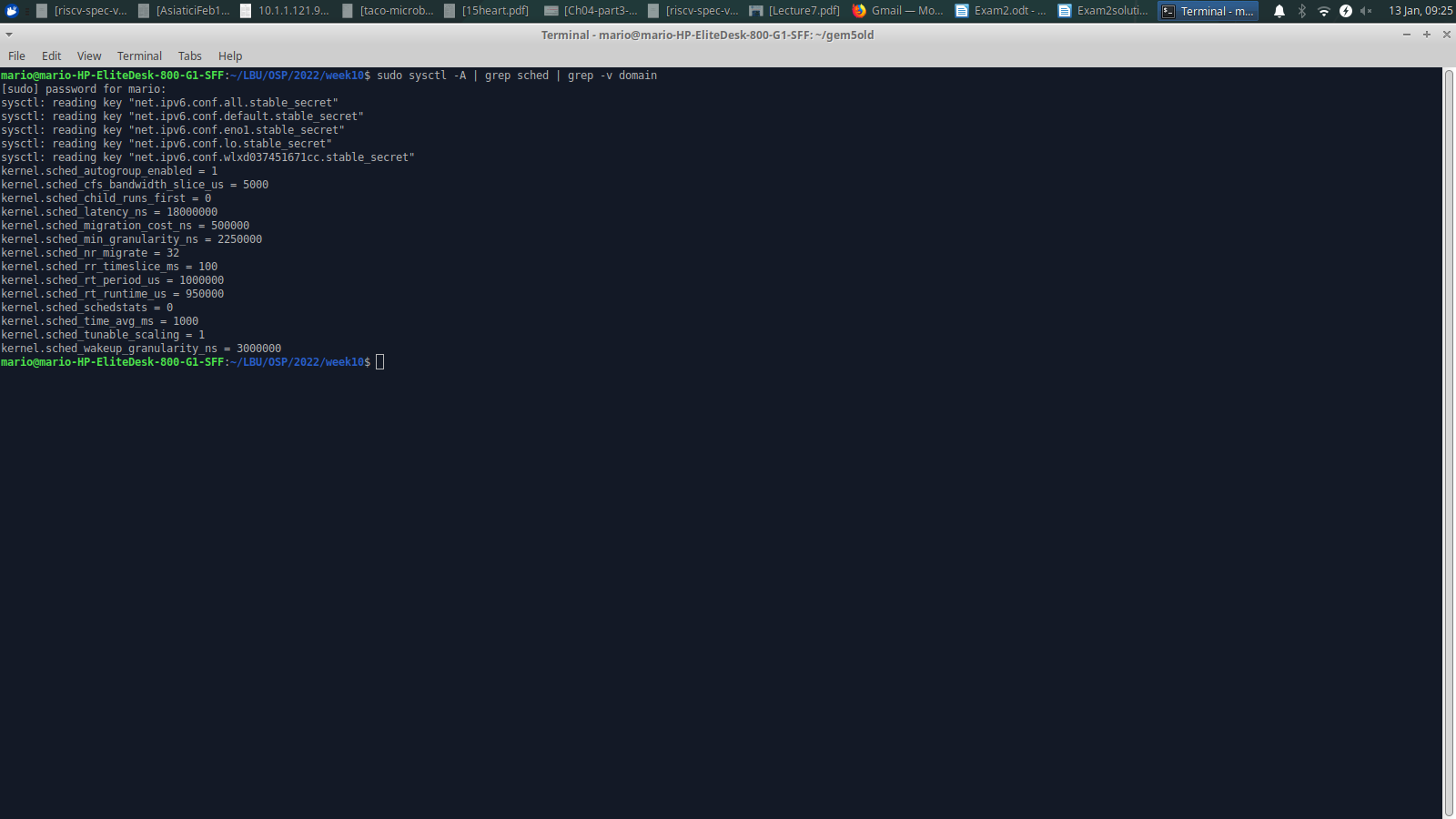
Figure 2: Output of chrt command

Inspect the Figure 2 above and list below the scheduling policy used. Hint: use man chrt <enter> (2 marks)

Answer:

**Based on the output provided from the `chrt` command, the scheduling policy used for the process with PID 19700 is "SCHED\_RR" (Round Robin). The current scheduling priority is set to 12.**

**2d.** Consider the following screenshot:

Figure 3: sysctl command showing schedule (sched) kernel parameters

Consider now that you have one CPU and the following set of processes (table given below) are about to run, with the length of the CPU-burst time given in milliseconds (ms):

|  |  |  |
| --- | --- | --- |
| Process | Arrival time (ms) | Burst Time (ms) |
| P1 | 0 | 120 |
| P2 | 100 | 80 |
| P3 | 120 | 60 |
| P4 | 140 | 40 |
| P5 | 160 | 40 |

Using the policy you found in item **2c**, draw one Gantt chart illustrating the execution of the above processes. Suggestion: draw it as a table with one line and multiple columns, and each column representing the time when a process starts/continues to be executed. To facilitate, use columns of 10 ms and make it start from time 0. Hint: you can extract the time slice from Figure 3. (10 marks)

Answer:

**Let's create a Gantt chart for the given processes using the Round Robin scheduling policy with a time slice of 100 ms (as extracted from Figure 3):**

|  |  |
| --- | --- |
| **Time** | **Process** |
| 0 | P1 |
| 100 | P2 |
| 180 | P3 |
| 240 | P1 |
| 260 | P4 |
| 300 | P5 |

**Question 3 (total 8 marks)**

The program below is also referred as question3.c. It is copied below to facilitate reading.

*// Source code modified from: http://www.informit.com*

*#include <fcntl.h>*

*#include <signal.h>*

*#include <stdio.h>*

*#include <string.h>*

*#include <sys/mman.h>*

*#include <sys/stat.h>*

*#include <sys/types.h>*

*#include <unistd.h>*

*int size;*

*char\* array;*

*int main ()*

*{*

*int fd;*

*struct sigaction sa;*

*size = getpagesize ();*

*fd = open ("file.txt", O\_RDWR);*

*array = mmap (NULL, 10 \* size, PROT\_NONE, MAP\_SHARED, fd, 0);*

*mprotect (array, 8 \* size, PROT\_READ);*

*printf("array[30000] = %c\n", array[30000]) ;*

*printf("writing to array: press enter");*

*getchar();*

*array[30000] = 'x';*

*close (fd);*

*printf ("all done\n");*

*munmap (array, 10\*size);*

*return 0;*

*}*

question3.c programme does mimic some OS operations. Note the size of a page is stored in the int size variable and size = getpagesize(), thus size = 4096.

**Note:** you do **not** have to compile or execute this programme to solve this question, therefore we do not provide file.txt.

**3a.** When question3 process is running, it extends its virtual memory space via the mmap system call. What is the size of the area created by mmap? (1 mark) What is the protection of the area created by mmap (2 marks)? (total 3 marks) Hint: do man mmap <enter> to find about its parameters.

Answers:

**The question3.c program extends its virtual memory space using “mmap”, creating a 40KB area with an initial protection of no access (“PROT\_NONE”). Later, the protection is modified to allow read access (“PROT\_READ”) for a portion of the virtual memory area using the “mprotect” system call.**

**3b.** Observe the mprotect call in the programme. Explain how many pages are changed in the mprotect call? (1 mark). What is the protection of the affected pages after executing mprotect ? (2marks) (total 3 marks) Hint: do man mprotect to find about its parameters.

Answers:

**The mprotect call in the question3.c program changes the protection of 8 pages, with the affected pages now having read access (“PROT\_READ”), while write and execute access remain disallowed.**

**3c.** Given the protections of the above programme, does Linux allow the following programme line to be executed? Explain. (1 mark)

*printf("array[30000] = %c\n", array[30000]) ;*

Answer:

**Yes, the given program line `printf("array[30000] = %c\n", array[30000]);` can be executed. The mprotect call in the question3.c program allows read access to the virtual memory area, including the specified index 30000 of the array. Therefore, reading the value at index 30000 and printing it using printf is permitted.**

**3d.** Given the protections of the above programme, does Linux allow the following program line to be executed? Explain. (1 mark).

*array[30000] = 'x';*

Answer:

**No, the given program line “array[30000] = 'x';” would not be allowed to execute. The mprotect call in the question3.c program provides read access to the virtual memory area but does not grant write access. Therefore, attempting to write the value 'x' to index 30000 of the array would result in a segmentation fault or a similar error, as the operation violates the protection settings**.