ASSIGNMENT : 6

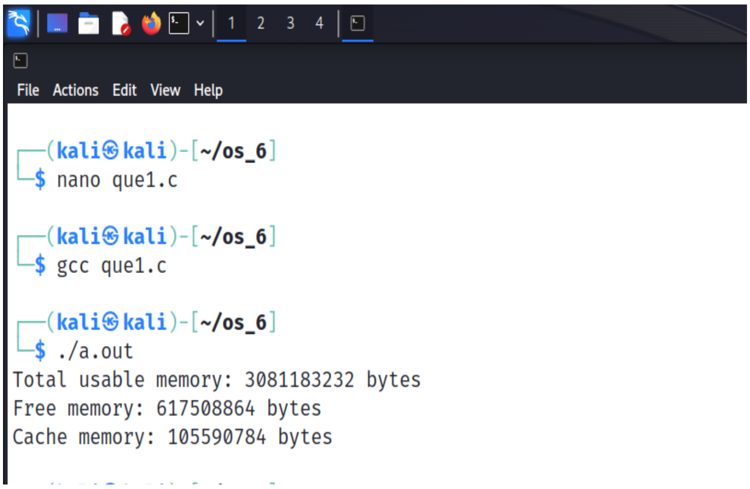
I. Write a ‘C’ program to find out the:

1. Total amount of usable memory, free memory and cache memory available in a system.

CODE :-

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/sysinfo.h>

int main() {

    struct sysinfo info;

    if (sysinfo(&info) != 0) {

        printf("Error getting system information.\n");

        exit(1);

    }

    printf("Total usable memory: %ld bytes\n", info.totalram);

    printf("Free memory: %ld bytes\n", info.freeram);

    printf("Cache memory: %ld bytes\n", info.bufferram);

    return 0;

}
```

OUTPUT :-



```
┌──(kali㉿kali)-[~/os_6]
└─$ nano que1.c

┌──(kali㉿kali)-[~/os_6]
└─$ gcc que1.c

┌──(kali㉿kali)-[~/os_6]
└─$ ./a.out
Total usable memory: 3081183232 bytes
Free memory: 617508864 bytes
Cache memory: 105590784 bytes
```

2. For a particular process (given its PID):

i. Total program size (in terms of number of pages).

ii. Size of program resident in RAM.

iii. Number of pages that are shared.

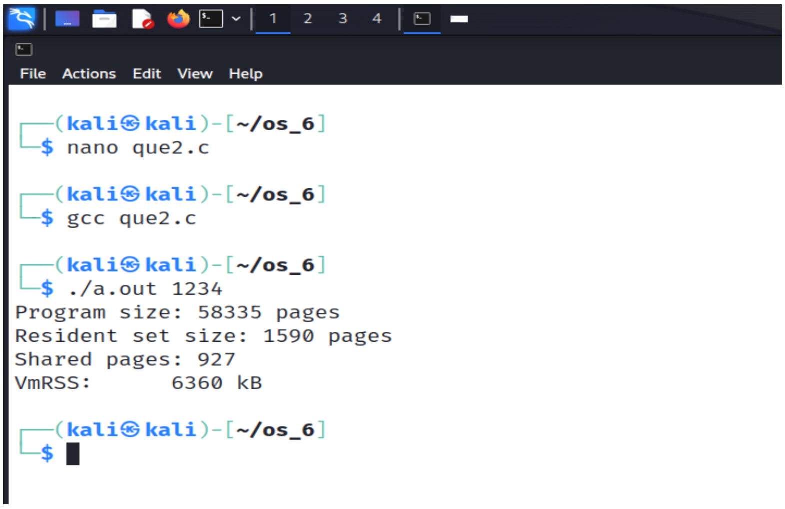
iv. Number of dirty pages

CODE :-

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char* argv[]) {

   if (argc != 2) {

      printf("Usage: %s <PID>\n", argv[0]);

      exit(1);

   }

   int process_id = atoi(argv[1]);

   char statm_filename[50];

   sprintf(statm_filename, "/proc/%d/statm", process_id);

   FILE* statm_file = fopen(statm_filename, "r");

   if (statm_file == NULL) {

      printf("Error opening %s file.\n", statm_filename);

      exit(1);

   }

   unsigned long program_size, resident_set_size, shared_pages, text_size, library_size,
data_size, dt_size;

   fscanf(statm_file, "%lu %lu %lu %lu %lu %lu %lu", &program_size, &resident_set_size,
&shared_pages, &text_size, &library_size, &data_size, &dt_size);

   fclose(statm_file);

   char status_filename[50];
```

```
    sprintf(status_filename, "/proc/%d/status", process_id);
    FILE* status_file = fopen(status_filename, "r");
    if (status_file == NULL) {
        printf("Error opening %s file.\n", status_filename);
        exit(1);
    }
    char line[128];
    while (fgets(line, sizeof(line), status_file)) {
        if (strncmp(line, "VmRSS:", 6) == 0) {
            printf("Program size: %lu pages\n", program_size);
            printf("Resident set size: %lu pages\n", resident_set_size);
            printf("Shared pages: %lu\n", shared_pages);
            printf("%s", line);
        }
        if (strncmp(line, "Dirty:", 6) == 0) {
            printf("%s", line);
        }
    }
    fclose(status_file);
    return 0;
}
```

OUTPUT :-



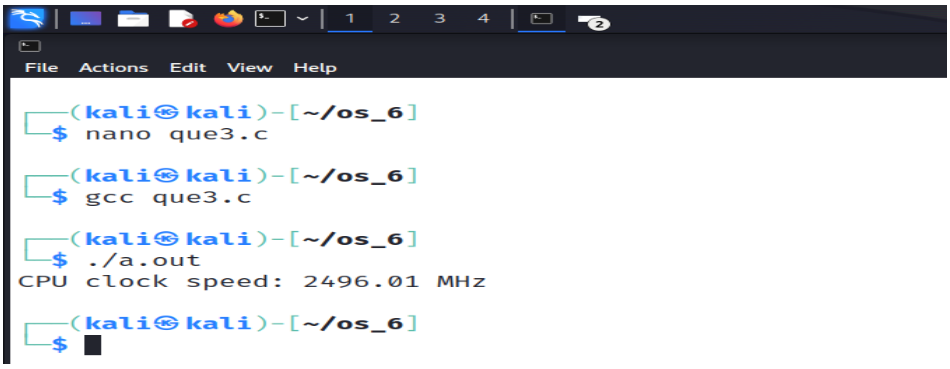
```
┌──(kali㉿kali)-[~/os_6]
└─$ nano que2.c

┌──(kali㉿kali)-[~/os_6]
└─$ gcc que2.c

┌──(kali㉿kali)-[~/os_6]
└─$ ./a.out 1234
Program size: 58335 pages
Resident set size: 1590 pages
Shared pages: 927
VmRSS:      6360 kB

┌──(kali㉿kali)-[~/os_6]
└─$ 
```

3. Clock speed of the CPU.

CODE :-

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main() {

    char buffer[1024];

    char* match;

    double clock_speed;

    FILE* cpu_info_file = fopen("/proc/cpuinfo", "r");

    if (cpu_info_file == NULL) {

        printf("Error opening /proc/cpuinfo file.\n");

        exit(1);

    }

    while (fgets(buffer, sizeof(buffer), cpu_info_file)) {

        match = strstr(buffer, "cpu MHz");

        if (match != NULL) {

            sscanf(match, "cpu MHz : %lf", &clock_speed);

            printf("CPU clock speed: %.2lf MHz\n", clock_speed);

            break;

        }
    }
    fclose(cpu_info_file);
    return 0;
}
```

OUTPUT :-



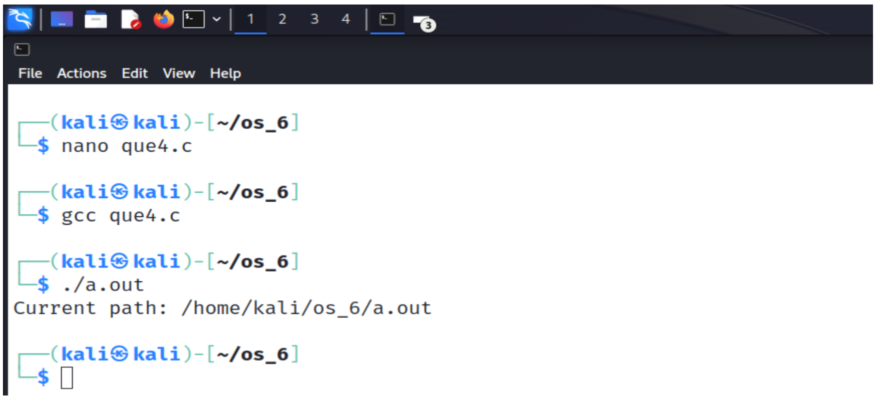
```
┌──(kali㉿kali)-[~/os_6]
└─$ nano que3.c

┌──(kali㉿kali)-[~/os_6]
└─$ gcc que3.c

┌──(kali㉿kali)-[~/os_6]
└─$ ./a.out
CPU clock speed: 2496.01 MHz

┌──(kali㉿kali)-[~/os_6]
└─$ █
```

4. Path of currently running programs.

CODE :-

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
    char path[1024];
    int count = readlink("/proc/self/exe", path, 1024);
    if (count < 0) {
        printf("Error getting current path.\n");
        exit(1);   }
    path[count] = '\0';
    printf("Current path: %s\n", path);
    return 0;  }
```

OUTPUT :-



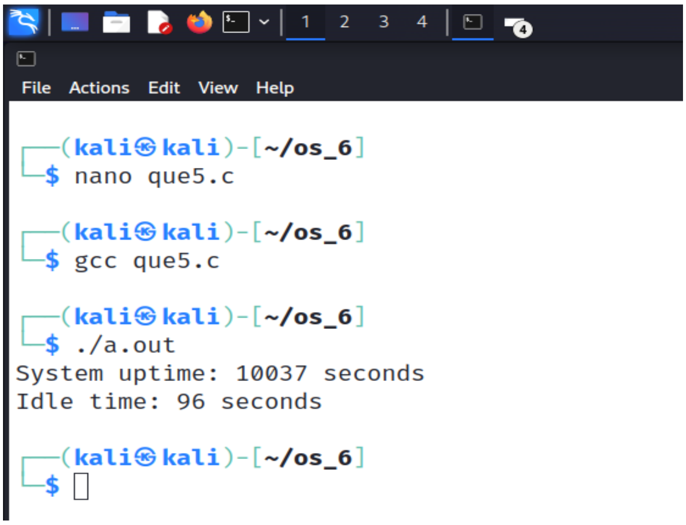
```
┌──(kali㉿kali)-[~/os_6]
└─$ nano que4.c

┌──(kali㉿kali)-[~/os_6]
└─$ gcc que4.c

┌──(kali㉿kali)-[~/os_6]
└─$ ./a.out
Current path: /home/kali/os_6/a.out

┌──(kali㉿kali)-[~/os_6]
└─$ 
```

5. System UP time and Idle time.

CODE :-

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main() {

    char buffer[1024];

    unsigned long uptime, idle_time;

    FILE* uptime_file = fopen("/proc/uptime", "r");

    if (uptime_file == NULL) {

        printf("Error opening /proc/uptime file.\n");

        exit(1);

    }

    fgets(buffer, sizeof(buffer), uptime_file);

    sscanf(buffer, "%lu %lu", &uptime, &idle_time);

    fclose(uptime_file);

    printf("System uptime: %lu seconds\n", uptime);

    printf("Idle time: %lu seconds\n", idle_time);

    return 0;

}
```

OUTPUT :-



```
┌──(kali㉿kali)-[~/os_6]
└─$ nano que5.c

┌──(kali㉿kali)-[~/os_6]
└─$ gcc que5.c

┌──(kali㉿kali)-[~/os_6]
└─$ ./a.out
System uptime: 10037 seconds
Idle time: 96 seconds

┌──(kali㉿kali)-[~/os_6]
└─$ 
```