Lab N°04 : Memory Management

Exercise V.1: (Process Virtual Memory)

A process's virtual memory is made up of a set of regions. A region is a contiguous section of a process's virtual memory. The system associates specific protections and roles with each region. The aim of this exercise is to visualize and understand these regions.

Q1) Start by compiling (with the -static option) the following program:

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
  int one_global;
  int one_other = 4;
  char* alloc;
  int main(){
  int one_local;
  alloc = malloc(1024L * 1024L * 10L);                    /*10MB */
  printf("PID = %d\n", getpid());
  printf("address of one_global = %8lx\n",(unsigned long)& one_global);
  printf("address of other_global= %8lx\n",(unsigned long)& one_other);
  printf("address of one_local = %8lx\n",(unsigned long) & one_local);
  printf("address of alloc = %8lx\n", (unsigned long) alloc);
  printf("address of main = %8lx\n", (unsigned long) & main);
  printf("adresse of printf = %8lx\n", (unsigned long) & printf);

  /* show memory card */
  sprintf(alloc, "cat /proc/%d/maps", getpid());
  system(alloc);
  return 0;
}
```

Q2) By comparing the addressable space of each region with the addresses given by the program, give a meaning to each region.

Q3) Same question, but compile the program without the -static directive.

Exercise V.2: (Process Segments)

Q1) Start by compiling the following program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

  int global;
  int global_init = 999;
  static int global_static;

  void f() {
     printf("oups\n");
  }

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

```
  int *p = malloc(10);
  static int local_static;

  printf("%18p Global                  \n", &global);
  printf("%18p Global static           \n", &global_static);
  printf("%18p Local static            \n", &local_static);
  printf("%18p Global init             \n", &global_init);
  printf("%18p String literal          \n", "bonjour");
  printf("%18p Function (f)            \n", &f);
  printf("%18p Function (main)         \n", &main);
  printf("%18p Local                   \n", &local);
  printf("%18p Argc                    \n", &argc);
  printf("%18p Argv                    \n", argv);
  printf("%18p Alloc                   \n", p);
  printf("%18p Function (lib)          \n", &printf);

  free(p);
  p = (void *) - 1;
  printf("%18p Adr max                 \n", p);

  if (argc == 2)
     sleep(600);
}
```

Q2) Sort the addresses in ascending and descending order

Q3) View the memory map of this process.