Problem 2:

- In which cases we should use aligned_malloc() instead of standard malloc.

Standard malloc() or realloc() in GNU systems always return address which is a multiple of eight (or sixteen on 64-bit systems), mean that malloc will return a suitably aligned memory block for any of the standard types.

But if you need a stricter alignment (for example, on SSE2 (SIMD) instructions need their data aligned on 16-byte boundaries, or a memory alignment is needed for better performance on some system), you'll need to use aligned_malloc()

- How can we increase the size of heap in a running process?

Heap in a running program is a continuous (in term of virtual addresses) space of memory with three bounds: a starting point, a maximum limit and an end point called break. You can further increase the size of heap by moving the end point farther with sbrk(), or place the break at a farther address using brk(). But, a user program shouldn't, and won't need to call these system calls, as these system calls will be called when using malloc().

However, we can write a simple program to try increase the heap size by using sbrk()

increase_heap.c > main()

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/resource.h>   //for rlimit
#include <stdlib.h>

int main() {
    printf("my pid is: %d\n", getpid());
    printf("before increasing\n");
    printf("current break is at %p\n", sbrk(0));
    //sleep(20);
    sbrk(80);
    printf("after increase\n");
    printf("We have a new break, it's %p\n", sbrk(0));
    //sleep(20);
    return 0;
}
```

```
my pid is: 3139
before increasing
current break is at 0x55adcc6d2000
after increase
We have a new break, it's 0x55adcc6d2050
```