Quiz 1

Operating Systems, Fall 2019
Iran University of Science and Technology

Solutions

Problem 0: I have written a simple **C** code as it is shown below. What could be the result of running this code?
Will it terminate successfully or there is some error?

```c
#include <stdio.h>


int* func1()
{
  const int size = 10;
  int arr[size];
  for (int i = 0; i < size; i++)
    arr[i] = i * i;
  return arr;
}

int main(int argc, char* argv[])
{
  printf("Hello World!\n");
  int *arr;
  arr = func1();
  printf("8^2 = %d\n", arr[8]);
  return 0;
}
```

p1.c

Answer 0:
The *arr* is allocated in stack section of memory. When the function returns the memory allocated for array is poped. As a result the pointer is no longer valid and a 'Segmentation error' is caused for accessing the memory pointer is addressing.

```
Hello World!
Segmentation fault (core dumped)
```

Note: Values returned are delicately placed in memory (or registers) so the **caller** function can access it. Check link below for more information.
`https://en.wikipedia.org/wiki/X86_calling_conventions`

Problem 1: After running the code shown below, what is going to be printed on the screen. Assume system architecture is 32-bits.
Will using a 64-bits architecture change the output?

```c
#include <stdio.h>


int main(int argc, char * argv[]) {
  int a = 584;
  int *ptr = &a ;
  printf("value is : %d\n", *ptr);
  printf("sizeof: %d\n", sizeof(ptr));
  return 0;
}
```

p2.c

Answer 1: Size of every pointer is fixed and dependant to system architecture. A 32-bit system uses 32 bits for addressing so the size of a pointer will be 4 bytes. In a 64-bit system the pointer size will be 8 bytes.

```
value is : 584
sizeof: 4
```

Problem 2: What happens if I run this program and give 'farbod' as input?

```c
#include <stdio.h>
#include <string.h>



int main(int argc, char *argv[])
{
   char name[5];
   printf("name: ");
   scanf("%s", (char *)name);
   return 0;
}
```

p3.c

Answer 2:
The *name* array has the fixed size of 5 bytes (5 charachters). When trying to write data outside the boundries of the array, it is possible that the program access an unprivileged memory region. This will result in process termination.
(Notice the array is allocated in the stack memory.)

```
1 name: farbod
2 *** stack smashing detected ***: <unknown>
    terminated
3 Aborted (core dumped)
```