**ASSIGNMENT 2**

1. **Activation Records :-**

Case – 1)Recursive Function :-

Function which calculates factorial of a number.

Program:-

int factorial(int n) {

Int f;

If(n == 1 || n == 0)

f = 1;

else

f = n \* factorial(n – 1);

return f;

}

int main() {

a = 3;

result = factorial(3);

return 0;

}

* The global variables are allocated statically in the initial portion of the stack. Note that stack begins at 4096 according to the [ABI specification](https://silcnitc.github.io/run_data_structures/abi.html).

https://silcnitc.github.io/img/data_structure_39.png

* Assuming that user inputs 3 resulting in a=3, the main functions sets up stack locations for its local variables and calls the function factorial(3) after setting up a part of the callee's activation record.

https://silcnitc.github.io/img/data_structure_40.png

* Factorial(3) saves the old Base pointer and sets up locations for its local variables.

https://silcnitc.github.io/img/data_structure_41.png

* Factorial(3) calls factorial(2) and the activation record of factorial(2) is setup similar to the above steps. The register R0 is assumed to be used for temporary storage of the value of n in the expression *n \* factorial(n-1)* i.e, 3.

https://silcnitc.github.io/img/data_structure_42.png

* Activation record for factorial(1) (called by factorial(2)) is seup similarly.

https://silcnitc.github.io/img/data_structure_43.png

* factorial(1) calculates the result and returns it by setting the value at return value location and pops off it local variables and sets back the base pointer.

https://silcnitc.github.io/img/data_structure_44.png

* Similarly, factorial(2) calculates the steps and pops off its activation record till the result value after setting back the old base pointer.

https://silcnitc.github.io/img/data_structure_45.png

* Similarly, factorial(3) also calculates the result and returns it to the main function.

https://silcnitc.github.io/img/data_structure_46.png

* Main function calculates and sets the 'result' variable.

https://silcnitc.github.io/img/data_structure_47.png

Case – 2) Stack Smashing.

Let’s take an example where the code tries to access forbidden regions of computer memory.

**CODE :**

#include<stdio.h>

int main(){

char str[10];

int i;

for(i = 0; i < 20; i++)

str[i] = ‘a’;

return 0;

}

rsp1 rbp0 rbp1

Overflow %fs : 0x28 .

of ‘a’

allocated ‘a’

memory ‘a’

‘a’ rbp1 - 8

‘a’

‘a’

‘a’

‘a’ rbp1 - 10

0x1 to 0xf

rbp1 – 14

rsp1 – 20H

(rsp2)

|  |
| --- |
| rax -->%fs = 0x28 eax = 0 |

rdx content is changed due to overwriting of ‘a’

So rdx content is no more %fs : 0x28

Therefore , Stack\_chk\_fail is called

And Stack Smashing takes place