**Operating System HW4**

**학번:** 2016113832

**이름:** 윤동준

**담당교수:** 강순주

1. **개요**

Heap영역을 사용하는 코드와 Stack영역을 사용하는 코드를 하나의 실행 프로그램에서 실행시켰을 때, 메모리에서 Heap영역과 Stack영역이 충돌하는 지점을 확인하는 과제이다.

1. **프로그램 구조 설명**

**2.1 함수에 대한 설명**

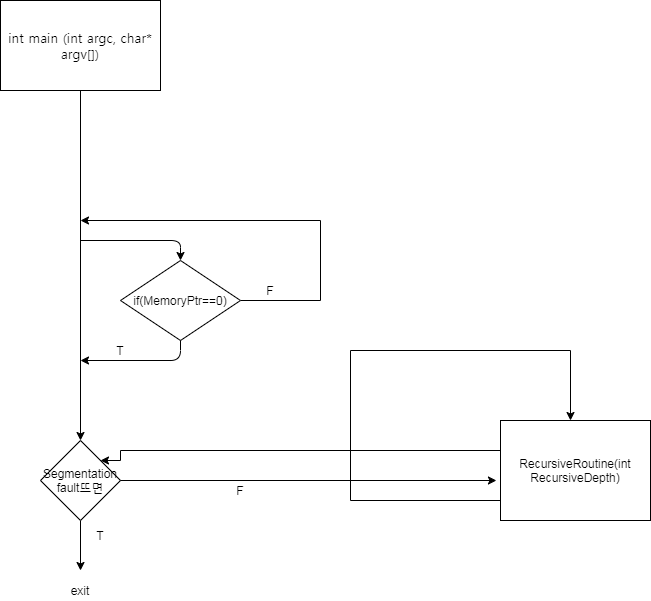
**-void RecursiveRoutine(int Recursive Depth):** 재귀호출하는 함수이다. 코드의 맨 마지막 줄을 보면 자기 자신을 호출하여 계속 돌린다.

**-char \*commas(unsigned long amount):** unsigned int를 string으로 바꿔준다, Base는 16진수를

나타낸다,

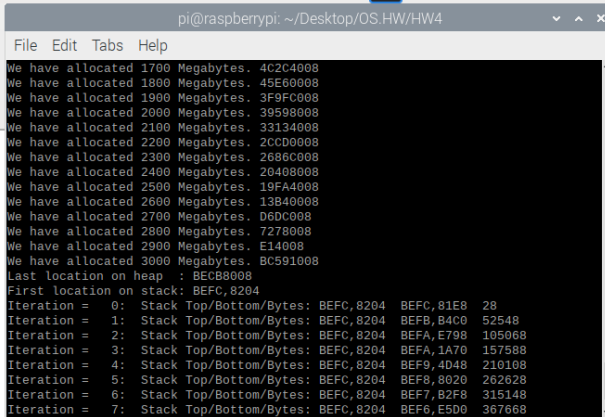
**-int main (int argc, char\* argv[]):** 제일 처음 전역 변수, 메인 함수, 코드, 지역변수의 주소값을 print하여 어디서 시작하는지 확인을 하고, Malloc를 사용하여 동적 할당을 NumberofAllocations에서 지정해준 횟수 만큼 동적 할당을 한다. 그리고 동적 할당 (Heap)부분을 마친 후, RecursiveRoutine를 사용하여 stack를 쌓는다.

* 1. **다이어그램**

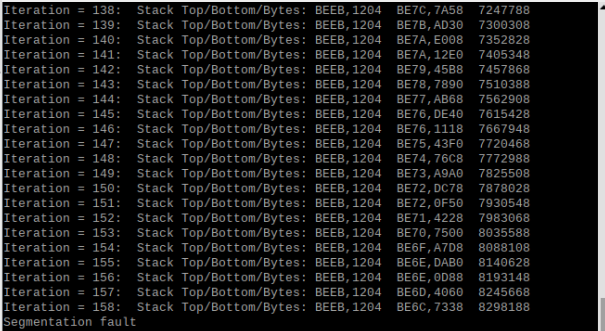
****

1. **실행결과**

Stack 지정된 크기가 8MB였고, 9MB가 최대여서 최대로 늘린 후 충돌을 확인하였다.



**오류**

Stack과 Heap 코드를 합쳐서 Heap 메모리 할당은 NumberOfMegabytes=4000메가바이트까지로 설정하고 실행해본 결과 Heap이 메모리 공간을 다 차지하게 되서 Stack에는 메모리가 할당이 되지 않았다.

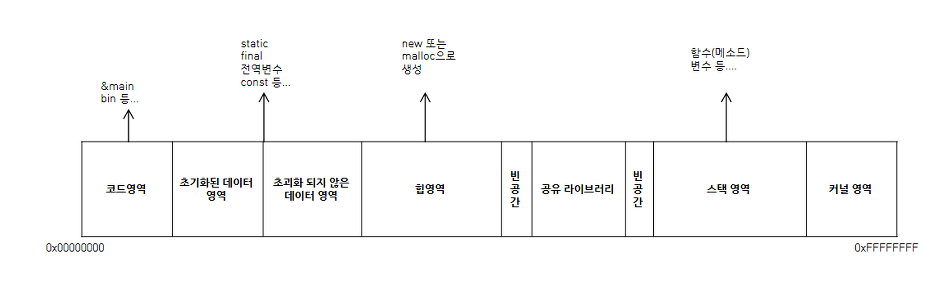
1. **고찰**

이번 과제를 시작하기 전, c언어 기초 지식을 늘렸다. 현재까지 1byte가 8bit라고 알고 있었는데,. 아니었다. 10비트 머신에서는 10비트가 1바이트이라고 한다. short, long, int자료형에 대해 공부해봤다.

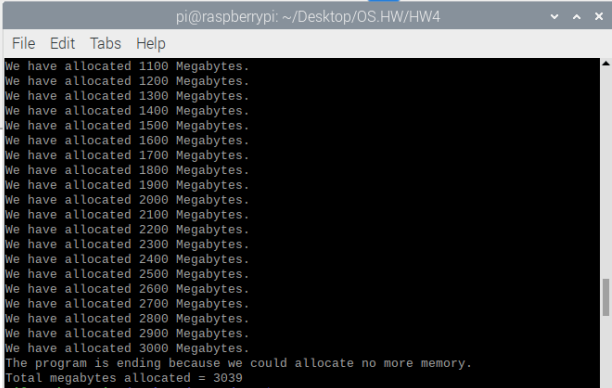
인터넷 검색 결과, c언어는 다양한 비트 단위를 갖는 하드웨어를 지원해야 해서 데이터 모델이 있고, 데이터 모델에 따라 short, int, long 등의 크기가 정해진다고 한다. short는 int형보다 더 적은 기억 공간을 사용할 수 있고, long은 int형보다 더 큰 기억 공간을 사용할 수 있다고 되어 있었다. 하지만 분명 short, int, long은 똑같은 4바이트인데 왜 short와 long은 int형보다 더 적고 큰 기억 공간을 사용할 수 있는지 이해가 가지 않았다. 내가 찾아본 정보에 의하면, “C가 데이터형을 기계에 맞추기 위해 3개를 나눠 놨다”라고 설명되어 있다. 처음 Window 3을 사용하던 시절에는 int와 short는 16비트였고, long은 32비트를 사용했다. 그 후 Windows와 Apple 시스템에서 short형을 16비트로 설정하고, int형과 long형은 32비트를 사용하는 방법을 채택하였다. 지금 현재 컴퓨터는 대부분 64비트 프로세서가 일반적으로 사용되고 있어서 64비트 정수형인 long long형이 만들어 졌다. 그리고 int형은 무조건 4바이트가 아니고 동작 환경에 따라 크기가 변한다는 것을 알게 되었다. 16비트 머신에서는 2바이트가 되고, 32비트 머신에서는 4바이트가 된다고 한다. 64비트 머신에서는 int가 8바이트가 되긴 하지만 실제로 컴파일러 등에서는 int=4바이트로 다룰 때가 많다고 한다.

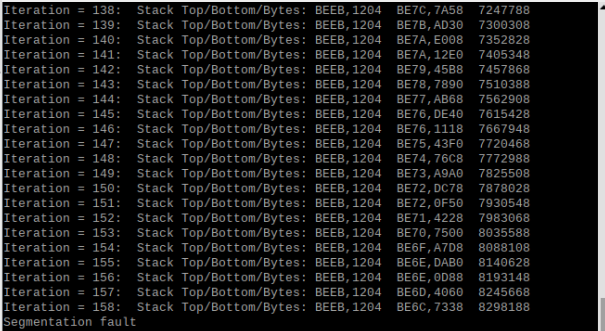
또 Stack overflow와 Heap overflow가 생성되는 이유에 대해 검색해본 결과, Stack overflow는 높은 차원의 배열을 선언하거나, 재귀 호출을 할 때 일어난다는 것을 알게 되었고, Heap overflow는 메모리 할당 방법 중 malloc를 사용하였을 떼 free를 언해줬거나, sizeof(int)\*1000000와 같이 큰 값을 넣을 때 overflow가 발생한다는 것을 알았다.

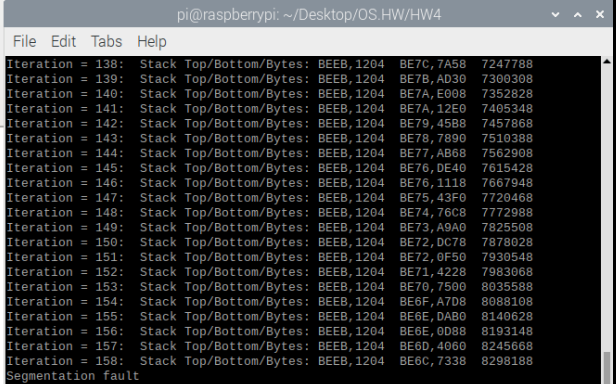
cat /proc/self/maps를 사용하여 실행중인 메모리 시스템, 메모리 주소를 확인해봤더니 heap과 stack사이에 lib가 공간을 차지하는 것을 확인하였다. Stack과 Heap이 서로 마주보는 방향으로 데이터를 저장하는데 그 이유가 궁금해서 찾아보니 버퍼오버플로우를 막기 위해서인걸 알았다.

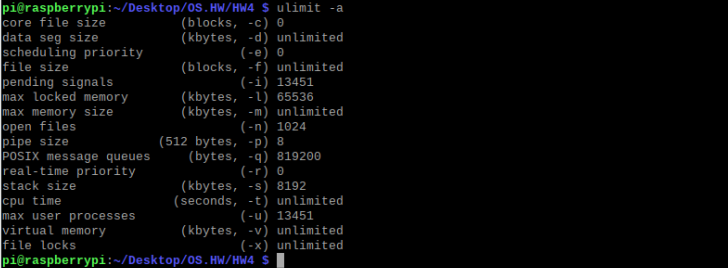


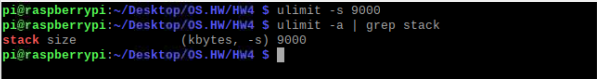
스택 영역 뒤에 바로 커널 영역이 있는데, 옛날 변수에 스택 영역이 초과되는 크기의 데이터를 저장한 후 커널 영역에 침범하여 해킹에 많이 사용됐기 때문이다.



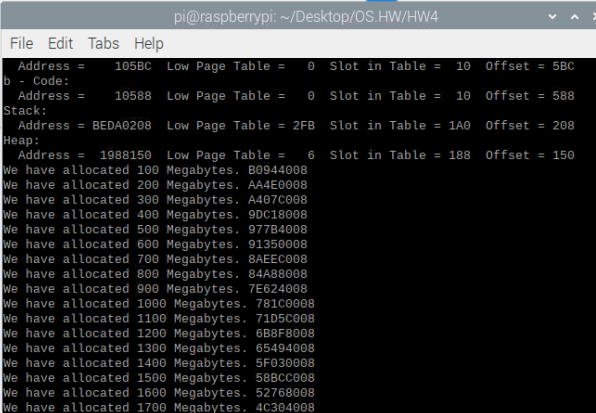
Stack과 Heap 코드를 합쳐서 Heap 메모리 할당은 NumberOfMegabytes=4000메가바이트까지로 설정하고 실행해본 결과 Heap이 메모리 공간을 다 차지하게 되서 Stack에는 메모리가 할당이 되지 않았다.

그래서 NumberOfMegabytes=1000으로 설정하고 실행해본 결과 Heap에 1,000바이트가 할당되고 Stack에도 메모리가 할당된 것을 볼수 있었다.

하지만 heap에 2,000바이트를 할당하였을 때도 똑같이 Iteration이 158번 돌았다. 분명 heap의 크기를 늘리면 Stack의 사이즈가 줄어야 하는데 왜 이런건지 확인해본 결과, 컴퓨터에 기본적으로 Stack의 사이즈가 정해져 있어서, 똑같이 158 iteration이 일어난 것이다.

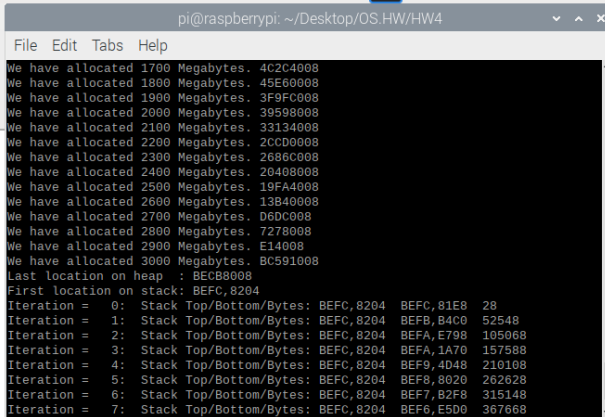
스택에 사이즈가 지정되어 있다는 것을 알고 나서 ulimit -a를 사용하여 stack size를 찾아보았을 때, 약 8MB로 지정되어 있는 것을 알 수 있었다.

Stack 용량을 9MB로 지정하고 실행을 하였을 때, Iteration 수만 늘어났다.



충돌이 왜 일어나지 않나 확인해보니 Heap이 원래 낮은 주소에서 높은 주소로 저장이 되어야 하는데 낮은 주소로 저장이 되고 있다. Stack은 원래대로 높은 주소에서 낮은 주소로 저장되고 있다.

코드를 확인해보니 haep의 주소는 낮은곳부터 시작을 해야하는데 높은 곳에서 시작을 해서 에러가 나고 있다. 계속 인터넷 검색을 해봤지만 이유를 찾지 못했다. 그래서 교수님의 원래 Malloc코드를 실행해보니 똑같은 현상이 일어나서 무시하고 코딩을 진행하였다.



결과적으로 Stack이 Heap영역에 침범한 후 메모리가 생성된다는 것을 볼 수 있었다.

1. **프로그램 소스 파일**

#include <stdio.h>

#include <string.h>

#include <stdarg.h>

#include <stdlib.h>

#define STACK\_ALLOC ONE\_MEG/20

#define HighPageTableSlot( Address ) (Address>>22) & 0x000003FF

#define LowPageTableSlot( Address) (Address>>12) & 0x000003FF

#define Offset( Address ) Address & 0x00000FFF

#define ONE\_MEG 1048576

#define MAXTEXT 25

#define BASE 16

#define GROUP 4

// This is a global so the subroutine can see it.

unsigned long FirstStackLocation;

long \*MemoryPtr; //WILL memory allocate by using malloc function

int Global;

int b() {

Global++;

}

void RecursiveRoutine( int );

char \*commas(unsigned long amount);

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

int Stack;

int Address;

int NumberofAllocations= 0;

int NumberofMegabytes = 3039;

char \*Heap = malloc( 10 \* sizeof(char) );

b();

printf( "\n\nSlicing up the logical address space\n");

printf( "<---- 10 bits ----><---- 10 bits ----><------ 12 bits ------>\n");

printf( "--------------------------------------------------------------------\n");

printf( "| | | |\n");

printf( "| Defines the slot | Defines the slot | Offset within the |\n");

printf( "| in High Level Page | in Low Level Page | Physical Frame. |\n");

printf( "| Table - selects | Table - selects | |\n");

printf( "| the low level page | the Frame in | |\n");

printf( "| table. | Physical Memory. | |\n");

printf( "| | | |\n");

printf( "--------------------------------------------------------------------\n");

printf( "Global Data: \n");

Address = (int)&Global;

printf( " Address = %8X Low Page Table = %3X Slot in Table = %3X Offset = %3X\n",

Address, HighPageTableSlot( Address ), LowPageTableSlot( Address), Offset( Address ) );

printf( "Main - Code: \n");

Address = (int)main;

printf( " Address = %8X Low Page Table = %3X Slot in Table = %3X Offset = %3X\n",

Address, HighPageTableSlot( Address ), LowPageTableSlot( Address), Offset( Address ) );

printf( "b - Code: \n");

Address = (int)b;

printf( " Address = %8X Low Page Table = %3X Slot in Table = %3X Offset = %3X\n",

Address, HighPageTableSlot( Address ), LowPageTableSlot( Address), Offset( Address ) );

printf( "Stack: \n");

Address = (int)&Stack;

printf( " Address = %8X Low Page Table = %3X Slot in Table = %3X Offset = %3X\n",

Address, HighPageTableSlot( Address ), LowPageTableSlot( Address), Offset( Address ) );

printf( "Heap: \n");

Address = (int)Heap;

printf( " Address = %8X Low Page Table = %3X Slot in Table = %3X Offset = %3X\n",

Address, HighPageTableSlot( Address ), LowPageTableSlot( Address), Offset( Address ) );

while (NumberofAllocations < NumberofMegabytes) {

MemoryPtr = (long\* )malloc(ONE\_MEG);

if (MemoryPtr == 0) {

printf("The program is ending because we could allocate no more memory.\n");

printf("Total megabytes allocated = %d\n", NumberofAllocations);

exit(0);

}

NumberofAllocations++;

if ((NumberofAllocations % 100 ) == 0)

printf("We have allocated %d Megabytes. %X\n", NumberofAllocations, MemoryPtr);

}

printf("Last location on heap : %X\n", MemoryPtr);

int TopOfStack;

int Counter = 0;

FirstStackLocation = (unsigned long)(&TopOfStack);

printf("First location on stack: %s\n",

commas( (unsigned long)FirstStackLocation ) );

RecursiveRoutine( 0 );

}

///////////////////////////////////////////////////////////////////////////

// This routine is called recursively. It does no real work but adds

// a variable to the stack which uses up space onthe stack.

// It's expected that the program will crash when memory is exhausted.

///////////////////////////////////////////////////////////////////////////

void RecursiveRoutine( int RecursiveDepth )

{

char Temp[ STACK\_ALLOC ];

char StringTop[32];

char StringBottom[32];

strcpy( StringTop, commas( (unsigned long)(FirstStackLocation) ) );

strcpy( StringBottom, commas( (unsigned long)&(Temp[STACK\_ALLOC]) ) );

printf("Iteration = %3d: Stack Top/Bottom/Bytes: %s %s %d\n",

RecursiveDepth, StringTop, StringBottom,

FirstStackLocation - (unsigned long)&(Temp[STACK\_ALLOC]) );

RecursiveDepth++;

RecursiveRoutine( RecursiveDepth );

}

///////////////////////////////////////////////////////////////////////////

// The following routines are for formatting only and aren't needed for

// an understanding of the memory manipulations done above.

///////////////////////////////////////////////////////////////////////////

// This routine converts an unsigned integer into a string with commas.

// You may need to adjust the base and the number of digits between

// commas as given by BASE and GROUP.

// Need space to hold the digits of an unsigned int,

// intervening commas and a null byte. It depends on

// BASE and GROUP above (but logarithmically, not

// as a constant. so we must define it manually here)

///////////////////////////////////////////////////////////////////////////

int prepend(char \*, unsigned, char \*);

int preprintf(char \*, unsigned, char \*, ...);

char \*commas(unsigned long amount)

{

short i;

short offset = MAXTEXT-1; /\* where the string "starts" \*/

short place; /\* the power of BASE for \*/

/\* current digit \*/

static char text[MAXTEXT];

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

text[i] = '\0';

/\* Push digits right-to-left with commas \*/

for (place = 0; amount > 0; ++place)

{

if (place % GROUP == 0 && place > 0)

offset = prepend(text,offset,",");

offset = preprintf(text,offset,"%X",amount % BASE);

amount /= BASE;

}

return (offset >= 0) ? text + offset : NULL;

}

/\* preprint.c: Functions to prepend strings \*/

int prepend(char \*buf, unsigned offset, char \*new\_str)

{

int new\_len = strlen(new\_str);

int new\_start = offset - new\_len;

/\* Push a string onto the front of another \*/

if (new\_start >= 0)

memcpy(buf+new\_start,new\_str,new\_len);

/\* Return new start position (negative if underflowed) \*/

return new\_start;

}

int preprintf(char \*buf, unsigned offset, char \*format, ...)

{

int pos = offset;

char \*temp = malloc(BUFSIZ);

/\* Format, then push \*/

if (temp)

{

va\_list args;

va\_start(args,format);

vsprintf(temp,format,args);

pos = prepend(buf,offset,temp);

va\_end(args);

free(temp);

}

return pos;

}

1. **자료 출처**

<https://hwan-shell.tistory.com/13>

<http://codinginterviewquestionsans.blogspot.com/2018/01/what-happens-when-stack-and-heap.html>

<https://bpsecblog.wordpress.com/2016/10/06/heap_vuln/>

<https://m.blog.naver.com/PostView.nhn?blogId=ruvendix&logNo=220905335341&proxyReferer=https:%2F%2Fwww.google.com%2F>

<http://blog.naver.com/PostView.nhn?blogId=jevida&logNo=140191334564>

<https://blockdmask.tistory.com/381>

<https://cosyp.tistory.com/206>