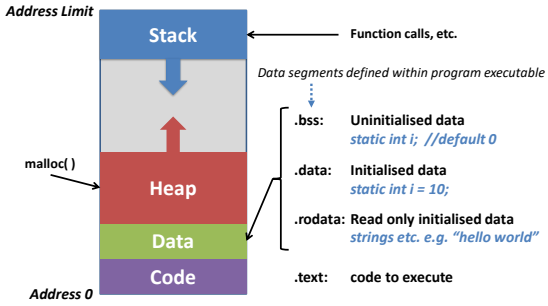
Processes: Memory Allocation

Dr Andrew Scott

a.scott@lancaster.ac.uk

1

Program Address Space

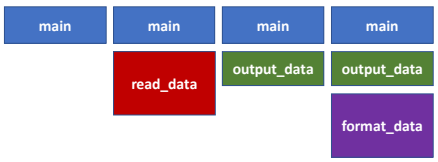


BSS (Block Started by Symbol) is an historic name used by a 1950's assembler for the IBM 704

2

Process Stack

- Common way of implementing function calls
  - Functions parameters + return address placed on stack before call
  - There is no standard, could use mix of registers and stack, …
- Grows and shrinks in blocks (frames) with function calls
  - On x68 stack grows down



3

Stack Allocation Test

- What would this code output?

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

char *
first( ) {
   char    first_buff[ 7 ];

   return  strcpy( first_buff, "Hello" );
}

char *
second( ) {
   char    second_buff[ 7 ];

   return  strcpy( second_buff, "World!" );
}

int
main( ) {
   char *  str1 = first(  );
   char *  str2 = second( );


   printf( "%s %s\n", str1, str2 );
}
```

4

Stack Allocation Test

- What would this code output?
- World! World!

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

char *
first( ) {
   char    first_buff[ 7 ];

   return  strcpy( first_buff, "Hello" );
}

char *
second( ) {
   char    second_buff[ 7 ];

   return  strcpy( second_buff, "World!" );
}

int
main( ) {
   char *  str1 = first(  );
   char *  str2 = second( );


   printf( "%s %s\n", str1, str2 );
}
```

5

Stack Allocation Test Output

- Let's output some addresses…

```
str1 = 0x8BD1
str2 = 0x8BD1
World! World!
```

…they're the same address!

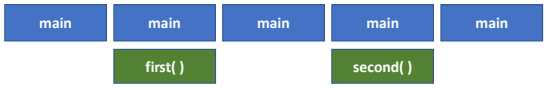
```
#include <stdio.h>
#include <string.h>

char *
first( ) {
   char    first_buff[ 7 ];

   return  strcpy( first_buff, "Hello" );
}

char *
second( ) {
   char    second_buff[ 7 ];

   return  strcpy( second_buff, "World!" );
}

int
main( ) {
   char *  str1 = first(  );
   char *  str2 = second( );

→  printf( "str1 = %p\nstr2 = %p\n", str1, str2 );
   printf( "%s %s\n", str1, str2 );
}
```

6

Think About the Process Stack

- To understand what's happening…

  …think how the stack grows and shrinks



7

Let's watch the stack pointer…

```
unsigned long long   stackVal;
asm( "mov %%rsp, %0" : "=rm" ( stackVal ) )
```

- Notice
  - The stack grows down as we enter first function, F0→C0
  - Then shrinks back to same address it started at, C0→F0
  - Same pattern repeated for second function, F0→C0→F0
    - Addresses are the same, so same address (8BD1) used for first_buff and second_buff
    - Therefore, when we fill second_buff, we overwrite what was there before

```
Stack before first( ):0x8BF0
Stack in first( ):    0x8BC0
Stack after first( ): 0x8BF0

Stack in second( ):   0x8BC0
Stack after second( ):0x8BF0

str1 = 0x8BD1, str2 = 0x8BD1
World! World!
```

8

Stack Allocation Test: *a simple fix?*

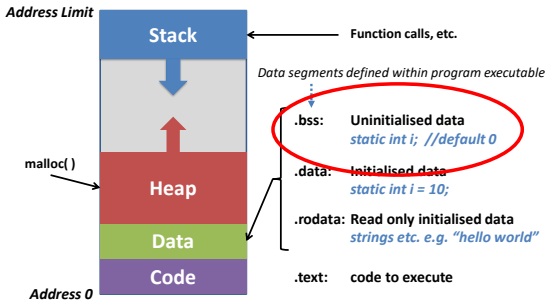
- Buffers not on stack
  - 2x7 bytes now safe in BSS segment *(next slide)* …and never freed, but…
- Wasteful if much bigger
  - Need better way…

```
Stack before first( ):0x8BF0
Stack in first( ):    0x8BD0
Stack after first( ): 0x8BF0

Stack in second( ):   0x8BD0
Stack after second( ):0x8BF0

str1 = 0x5011, str2 = 0x5018
Hello World!
```

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

char *
first( ) {
    static char   first_buff[ 7 ];

    return  strcpy( first_buff, "Hello" );
}

char *
second( ) {
    static char   second_buff[ 7 ];

    return  strcpy( second_buff, "World!" );
}

int
main( ) {
    char *  str1 = first(  );
    char *  str2 = second( );


    printf( "%s %s\n", str1, str2 );
}
```

9

Program Address Space



BSS (Block Started by Symbol) is an historic name used by a 1950's assembler for the IBM 704

10

Process Heap

- Predefined memory areas often too restrictive
  - For example, arrays can't expand if more data than expected
    - End up over sizing to be safe
- Heap allows dynamic allocation of memory as needed
  - Elements in linked-list, queue, graph, …
  - New object instances, etc.
- Use library calls to expand and (possibly) contract heap
  - memory = *malloc( )*
  - *free(* memory *)*

11