ex04

CSE 333 Winter 2021

Instructor: John Zahorjan

It's more complicated than I thought…

Teaching Assistants:

Matthew Arnold
Nonthakit Chaiwong
Jacob Cohen
Elizabeth Haker
Henry Hung
Chase Lee
Leo Liao
Tim Mandzyuk
Benjamin Shmidt
Guramrit Singh

What You Have to Do

1. Figure out how to implement memove and then implement memmove
2. Figure out how to use the timer library
3. Write driver/test code to call and time memmove
4. Figure out how to build your code with the timer class
5. Figure out to build your code to use either your memmove or the built-in memmove

1. Figure out how to implement memmove

- Hints:
  - none

2. Figure out how to use the timer library

- Hints:
  - The library distributes a test program that shows how to use the library
  - You build the test program using the makefile distributed with the library

3. Write driver/test code to call memmove

- Hints:
  - From the write-up: characterize performance by *throughput* measured in MB/sec. moved
  - Does performance depend on the arguments to memmove? If so, you might want to measure at least a few distinct scenarios. (This isn't the main point of the exercise, though.)

4. Figure out how to build you code…

- Hints:
  - The test program and makefile that come with the library not only show how to use the library but how to build an app that uses the library

5. Figure out how to build your app using either your memmove or built-in memmove

- Unexpected problem: the compiler seems to know what memmove is

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

int main(int argc, char *argv[])  {
  char src_buffer[] = "1234567890";
  char dest_buffer[] = "abcdefghij";

  memmove(dest_buffer, src_buffer+1, 9);
  printf("%s\n", dest_buffer);
  memmove(dest_buffer, src_buffer+2, argc);
  printf("%s\n", dest_buffer);

  return 0;
}
```

Output: 234567890j
334567890j

$ gcc –O0 –S test.c

```
main:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $48, %rsp
        movl    %edi, -36(%rbp)
        movq    %rsi, -48(%rbp)

        movabsq $4050765991979987505, %rax
        movq    %rax, -11(%rbp)
        movw    $12345, -3(%rbp)
        movb    $0, -1(%rbp)

        movabsq $7523094288207667809, %rax
        movq    %rax, -22(%rbp)
        movw    $27241, -14(%rbp)
        movb    $0, -12(%rbp)

        leaq    -11(%rbp), %rax
        addq    $1, %rax
        leaq    -22(%rbp), %rdx
        movq    (%rax), %rcx
        movq    %rcx, (%rdx)
        movzbl  8(%rax), %eax
        movb    %al, 8(%rdx)

        leaq    -22(%rbp), %rax
        movq    %rax, %rdi
        call    puts

        movl    -36(%rbp), %eax
        movslq  %eax, %rdx
        leaq    -11(%rbp), %rax
        addq    $2, %rax
        leaq    -22(%rbp), %rcx
        movq    %rax, %rsi
        movq    %rcx, %rdi
        call    memcpy

        leaq    -22(%rbp), %rax
        movq    %rax, %rdi
        call    puts

        movl    $0, %eax
        ret
```

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

char src_buffer[] = "1234567890";

'87654321'

'90'

'\0'

char dest_buffer[] = "abcdefghij";

memmove(dest_buffer, src_buffer+1, 9);

move 8 bytes

move 9th byte

printf("%s\n", dest_buffer);

memmove(dest_buffer, src_buffer+1, argc);

!!!

printf("%s\n", dest_buffer);

$ gcc –O3 –S test.c

```
main:
        movabsq $4050765991979987505, %rax
        pushq   %rbx
        movl    $27241, %edx
        movl    %edi, %ebx
        subq    $32, %rsp
        movq    %rax, 10(%rsp)
        movl    $12345, %eax
        leaq    21(%rsp), %rdi
        movw    %ax, 18(%rsp)
        movq    11(%rsp), %rax
        movw    %dx, 29(%rsp)
        movq    %rax, 21(%rsp)
        movzbl  19(%rsp), %eax
        movb    $0, 20(%rsp)
        movb    $0, 31(%rsp)
        movb    %al, 29(%rsp)
        call    puts

        movslq  %ebx, %rdx
        leaq    12(%rsp), %rsi
        leaq    21(%rsp), %rdi
        call    memcpy

        leaq    21(%rsp), %rdi
        call    puts

        addq    $32, %rsp
        xorl    %eax, %eax
        popq    %rbx
        ret
```

So, What To Do?

- Looks like link time is too late to make a decision about which version of memmove to use because the compiler will already have decided by then
- But, we "must" make the decision about which version to use something that happens at build time
  - Can't modify source, but…
  - Can modify compile commands, and…
  - Can modify link command

Link Time Control

- This was the original intent…
  - Our app calls memmove
  - If we link the app with our implementation of memmove, the linker connects the calls to our implementation
  - If we fail to provide our implementation, the linker connects the calls to the standard implementation
- Instead, the app makes a call to some different method name, like, say, MEMMOVE_CHOICE()
- We create two different versions of MEMMOVE_CHOICE, one that is our implementation of memmove and one that just invokes standard memmove
- Downside: Performance
  - Compiler can't do the opimizations we just saw with memove invoked in this way
  - We added an extra procedure call to each invocation of memmove

Compile Time Control

- Like before, we write in our code calls to some name that isn't memmove – say, MEMMOVE_CHOICE
- Then we use the pre-processor to convert the string "MEMMOVE_CHOICE" to either "memmove" (for standard version) or "my_memmove" (or whatever, for our version)
  - If memmove, compiler can optimize exactly as if we had hard-coded a call to memmove
- gcc –DMEMMOVE_CHOICE=my_memmove –std=c17 *.c
- Note: Need to create a .h file my_memmove and include it in the app code

The Timer Library

- The timer library uses this technique, but for a related but different reason
  - They supply a test app with the library and a makefile
  - The makefile builds four versions of the test app
- [attu4] ex04/c-timer-lib-master> make
  gcc -g -Wall -Wextra -std=gnu99 -c timer.c
  gcc -g -Wall -Wextra -std=gnu99 test.c timer.o -o test_s.out -DUNITS="s“
  gcc -g -Wall -Wextra -std=gnu99 test.c timer.o -o test_ms.out -DUNITS="ms“
  gcc -g -Wall -Wextra -std=gnu99 test.c timer.o -o test_ns.out -DUNITS="us“
  gcc -g -Wall -Wextra -std=gnu99 test.c timer.o -o test_mis.out -DUNITS="ns"

Test.c

```
#ifndef UNITS
#define UNITS s
#endif
int main()
{
   interval_t * a;
   interval_t * b;
   interval_t * c;

   create_interval(&a, "Test 1", mono, UNITS);
   create_interval(&b, "Test 2", mono, UNITS);
   create_interval(&c, "Test 3", mono, UNITS);
```

Timer.h

```
/* Enum of time units */
typedef enum
{
   s,   // seconds
   ms,  // milli-seconds
   us,  // micro-seconds
   ns,  // nano-seconds
   unit_check, // used for enum check
   none // use the intervals unit
} unit_e;

int create_interval( interval_t ** tmp,
                     char * name,
                     clock_e ck,
                     unit_e ut
                   );
```