CS 2021: Practice Final Exam SOLUTION

Spring 2019
University of Minnesota

Exam period: 20 minutes
Points available: 40

Problem 1 (10 pts): There are two terminal sessions below both with errors in getting a program running. Describe in each case whether the problem is a (1) Compile error (2) Static Linking Error or (3) Dynamic Linker/Loader Error, and **how to fix the problem**.

```
# SESSION A
> gcc tree_main.c -I mylibraries/ -lbstree
/usr/bin/ld: cannot find -lbstree
collect2: error: ld returned 1 exit status
```

Type of Error and Fix for it:
SOLUTION: This is a static linking error. The linker `ld` is complaining that it cannot locate the library `bstree` which is required. The location of this should be passed to the compiler with the `-L` option; the library is probably in `mylibraries` as indicated by the directory passed for headers with `-I`.

```
# SESSION B
> gcc math_main.c -lm
math_main.c: In function ‘main’:
math_main.c:3:14: error:
  implicit declaration of function ‘pow’
     double x = pow(3.14, 4.6);
```

Type of Error and Fix for it:
SOLUTION: This is a compilation error complaining that the `pow()` *function is not specified; fix it by adding* `#include <math.h>` *to the source code which will specify the prototype for* `pow()`*.*

Background: To the right is the output of `pmap` showing page table virtual memory mapping information for a running program called `memory_parts`. Answer the following questions about this output.

Problem 2 (5 pts): The mapped memory references something called `libc-2.26.so`. Describe this entity and what kind of information you would expect to find at the mapped locations.
SOLUTION: This is the C standard library. It is a shared object with the `.so` *extension and is likely to contain binary assembly instructions standard C functions like* `printf()` *and* `malloc()`*.*

Problem 3 (5 pts): Why does `pmap` only show a limited number of virtual addresses? What would happen if the program attempted to access an address not listed in the output? Example: address `0x00` is not in the listing.
SOLUTION: The page table only contains mapped pages for program. These mapped addresses are what is shown. The large number of other addresses are unmapped. Attempting to access these unmapped addresses will result in errors such as `segmentation faults`; this usually causes the program to be immediately terminated.

```
pmap 7986
7986:   ./memory_parts
00005579a4abd000      4K r-x-- memory_parts
00005579a4cbd000      4K r---- memory_parts
00005579a4cbe000      4K rw--- memory_parts
00005579a4cbf000      4K rw---   [ anon ]
00005579a53aa000    132K rw---   [ heap ]
00007f441f2e1000   1720K r-x-- libc-2.26.so
00007f441f48f000   2044K ----- libc-2.26.so
00007f441f68e000     16K r---- libc-2.26.so
00007f441f692000      8K rw--- libc-2.26.so
00007f441f694000     16K rw---   [ anon ]
00007f441f698000    148K r-x-- ld-2.26.so
00007f441f88f000      8K rw---   [ anon ]
00007f441f8bb000      4K r---- gettysburg.txt
00007f441f8bc000      4K r---- ld-2.26.so
00007f441f8bd000      4K rw--- ld-2.26.so
00007f441f8be000      4K rw---   [ anon ]
00007fff96ae1000    132K rw---   [ stack ]
00007fff96b48000     12K r----   [ anon ]
00007fff96b4b000      8K r-x--   [ anon ]
 total             4276K
```

Problem 4 (20 pts): Below are two functions that augment El Malloc with the block shrinking; this allows a user to specify that the originally requested size for a memory area can be adjusted down potentially creating open space. Fill in the definitions for these functions.

```
////////////////////////////// SOLUTION //////////////////////////////
el_blockhead_t *el_shrink_block(el_blockhead_t *head, size_t newsize){
// Shrinks the size of the given block potentially creating a new block.  Computes remaining space
// as the difference between the current size and parameter newsize. If this is smaller than
// EL_BLOCK_OVERHEAD, does nothing further and returns NULL. Otherwise, reduces the size of the
// given block by adjusting its header and footer and establishes a new block above it with
// remaining space beyond the block overhead. Returns a pointer to the newly introduced blocks. Does
// not modify any links in lists.

  // NOTE: could simplify considerably using el_split_block()
  size_t remaining = head->size - newsize;
  if(remaining < EL_BLOCK_OVERHEAD){
    return NULL;
  }
  head->size = newsize;                       // adjust size
  el_blockfoot_t *foot = el_get_footer(head); // allows middle foot to be found
  foot->size = newsize;                       // set

  el_blockhead_t *above_head = el_block_above(head); // new header location
  above_head->size = remaining - EL_BLOCK_OVERHEAD;  // set its size
  el_blockfoot_t *above_foot = el_get_footer(above_head); // should be old foot
  above_foot->size = remaining - EL_BLOCK_OVERHEAD;       // set size
  return above_head;
}

int el_shrink(void *ptr, size_t newsize){
// Shrink the area associated with the given ptr if possible.  Checks to ensure that the block
// associated with the given user ptr is EL_USED and exits if not.  Uses el_shrink_block() to
// adjust the block size and create a block for the remaining space.  If not possible to shrink,
// returns 0.  Otherwise moves the current block to the front of the Used List and places the newly
// created block to the front of the Available List after setting its state to EL_AVAILABLE. Returns
// 1 on successfully shrinking.

  el_blockhead_t *head = PTR_MINUS_BYTES(ptr,sizeof(el_blockhead_t));
  if(head->state != EL_USED){              // error check
    printf("Does not appear to be a used block\n");
    exit(1);
  }
  el_blockhead_t *above = el_shrink_block(head, newsize);
  if(above == NULL){
    return 0;                              // could not shrink
  }
  above->state = EL_AVAILABLE;             // now available for use
  el_remove_block(el_ctl.used, head);      // out of used list
  el_add_block_front(el_ctl.used,  head);  // to front of used list
  el_add_block_front(el_ctl.avail, above); // to front of available list
  return 1;                                // could shrink
  // likely want to attempt merging above with block above to limit fragmentation
  // in a full implementation of shrinking
}
```