Name: ID#: X.500:

CSCI 2021: Practice Final Exam SOLUTION

Spring 2023
University of Minnesota

Exam period: 20 minutes
Points available: 40

Background: Nearby are several C files along with two attempts to compile them on the left. Study these and answer the questions that follow.

```
> gcc vf_weak_var.c vf_strong_func.c vf_main.c   # COMPILE 1
/usr/bin/ld: warning: size of symbol 'foo' changed from 4 to 14
/usr/bin/ld: warning: type of symbol 'foo' changed from 1 to 2
> file a.out
a.out: ELF 64-bit LSB pie executable, x86-64, version
> ./a.out
-573193927
4
> rm a.out

> gcc vf_strong_var.c vf_strong_func.c vf_main.c # COMPILE 2
/usr/bin/ld: multiple definition of 'foo';
collect2: error: ld returned 1 exit status
> file a.out
a.out: cannot open 'a.out' (No such file or directory)
```

```c
// FILE: vf_main.c
#include <stdio.h>
int foo(int x);
int main(){
  printf("%d\n",foo);
  printf("%d\n",foo(2));
  return 0;
}

// FILE: vf_strong_func.c
int foo(int x){
  return 2*x;
}

// FILE: vf_strong_var.c
int foo = 0;

// FILE: vf_weak_var.c
int foo;
```

Problem 1 (10 pts): Why does `COMPILE 1` succeed while `COMPILE 2` fails? Mention pertinent properties of ELF files in your answer.

SOLUTION: `COMPILE 1` *succeeds because the integer* `int foo` *is uninitialized and therefore weak. It is overridden by the strong symbol* `int foo(int x)` *so the resulting ELF file has only the function version.* `COMPILE 2` *fails as the C file initializes* `int foo=0;` *making both definitions strong. Two strong symbols with the same name cannot exist in an ELF file causing linking to fail.*

Problem 2 (10 pts): Nearby 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.

(A) 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()`.

(B) 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 3 (10 pts): Below is an initial memory/cache configuration along with several memory load operations. Indicate whether these load operations result in cache hits or misses and show the state of the cache after these loads complete.

```
-------------------- SOLUTION --------------------
MAIN MEMORY                             DIRECT-MAPPED Cache, 8-byte lines
| Addr | Addr Bits       | Value |      4 Sets, 8-bit Address = 3-bit tag
|------+-----------------+-------|
|   10 | 000  10 000     |    10 |      INITIAL CACHE STATE
|   14 | 000  10 100     |    11 |      |     |   |     | Blocks/Line |
|   18 | 000  11 000     |    12 |      | Set | V | Tag | 0-3   4-7   |
|   1C | 000  11 100     |    13 |      |-----+---+-----+-------------|
|   20 | 001  00 000     |    20 |      |  00 | 1 | 010 | 200   201   |
|   24 | 001  00 100     |    21 |      |  01 | 1 | 001 | 22    23    |
|   28 | 001  01 000     |    22 |      |  10 | 1 | 000 | 10    11    |
|   2C | 001  01 100     |    23 |      |  11 | 0 |   - | -           |
|   30 | 001  10 000     |   100 |
|   34 | 001  10 100     |   101 |      HITS OR MISSES?
|   38 | 001  11 000     |   102 |      | OPEARTION     | HIT/MISS? |
|   3C | 001  11 100     |   103 |      |---------------+-----------|
|   40 | 010  00 000     |   200 |      | 1. Load 0x48 | Miss       |
|   44 | 010  00 100     |   201 |      | 2. Load 0x4C | Hit        |
|   48 | 010  01 000     |   202 |      | 3. Load 0x24 | Miss       |
|   4C | 010  01 100     |   203 |
|------+-----------------+-------|      FINAL CACHE STATE
|      | Tag Set Offset |        |      |     |   |     | Blocks/Line |           |
                                        | Set | V | Tag | 0-3   8-7   | Changed? |
                                        |-----+---+-----+-------------+----------|
                                        |  00 | 1 | 001 | 20    21    | ***      |
                                        |  01 | 1 | 010 | 202   203   | ***      |
                                        |  10 | 1 | 000 | 10    11    |          |
                                        |  11 | 0 |   - | -           |          |
```

Problem 4 (10 pts): Nearby is the definition for `base_scalvec()` which scales a vector by multiplying each element by a number. Write an optimized version of this function in the space provided. Mention in comments why you performed certain transformations.

```
int vget(vector_t vec, int idx){
  return vec.data[idx];
}
void vset(vector_t vec, int idx, int x){
  vec.data[idx] = x;
}
void base_scalevec(vector_t *vec, int *scale){
  for(int i=0; i < vec->len; i++){
    int cur = vget(*vec,i);
    int new = cur * (*scale);
    vset(*vec,i,new);
  }
}
```

```
//////// SOLUTION //////////
void opt_scalevec(vector_t *vec, int *scale){
  // locals to avoid memory access
  int *data = vec->data, len = vec->len;
  int scal = (*scale), i;
  // unroll x2 with duplicate vars to
  // enable pipelining
  for(i=0; i < len-2; i+=2){
    // no function calls - inline bodies
    // to improve register use
    int cur0 = data[i+0];
    int new0 = cur0 * scal;
    data[i+0] = new0;
    int cur1 = data[i+1];
    int new1 = cur1 * scal;
    data[i+1] = new1;
  }
  // cleanup loop
  for(; i<len; i++){
    int cur0 = data[i+0];
    int new0 = cur0 * scal;
    data[i+0] = new0;
  }
}
```