CSci 2021, Fall 2018, Written Exercise Set 5 Solutions

This assignment is not due. It is for practice.

Problem 1:

Part A:

```
int range(struct vector* v) {
  int length = v->length;
  int* arr = v->arr;
  if (!length) {
    return NULL;
  }
  int max = arr[0];
  int min = arr[0];
  int cur;
  for (int i = 1; i < length; ++i) {
    cur = arr[i];
    if (cur > max) {
      max = cur;
    }
    if (cur < min) {
      min = cur;
    }
  }
  return max - min;
}
```

Part B:

```
int* compute_sums(int** mat, int* res) {
  int times_two = 0;
  int plus_ten = 0;
  int cur;
  for (int i = 0; i < M; ++i) {
    for (int j = 0; j < N; ++j) {
      cur = mat[i][j];
      times_two += cur * 2;
      plus_ten += cur + 10;
    }
  }
  res[0] = times_two;
  res[1] = plus_ten;
  return res;
}
```

Problem 2:

```
A.) Virtual Address Layout

VPN - Virtual Page Number
VPO - Virtual Page Offset
TLBI - TLB Index
TLBT - TLB Tag

|-----VPN------------|----VPO----|
|13|12|11|10|9|8|7|6 |5|4|3|2|1|0|
|-----TLBT------|TLBI|

B.) Physical Address Layout

|-----PPN-----|----PPO----|
|11|10|9|8|7|6|5|4|3|2|1|0|

C.) 0x32E

Virtual Address in Binary:
|-----VPN------------|----VPO----|
|0 |0 |0 |0 |1|1|0|0 |1|0|1|1|1|0|
|-----TLBT------|TLBI|

VPN: 12 (decimal), C (Hex)
TLBI: 00 (binary), 0 (decimal), 0 (hex)
TLBT: 000011 (binary), 3 (decimal), 3 (hex)
TLB Hit (Y/N): Y
Page Fault (Y/N): N
PPN: 10010 (binary), 18 (decimal), 12 (hex)
Physcial Address In Binary:
|-----PPN-----|----PPO----|
|0 |1 |0|0|1|0|1|0|1|1|1|0|
|----CT-------|--CI---|-CO|

Cach Offset: 2 (decimal)
Cache Index: 11 (decimal), B (hex)
Cache Tag: 18 (decimal), 12 (hex)
Cache Hit (Y/N): Y
Byte: 2A
```

```
D.) 0x2C5

Virtual Address in Binary:
|-----VPN------------|----VPO----|
|0 |0 |0 |0 |1|0|1|1 |0|0|0|1|0|1|
|-----TLBT------|TLBI|

VPN:  11 (decimal),  B (Hex)
TLBI:  11 (binary),  3 (decimal), 3 (hex)
TLBT:  000010 (binary), 2 (decimal), 2 (hex)
TLB Hit (Y/N): Y
Page Fault (Y/N): N
PPN: 10101 (binary),  21 (decimal),  15 (hex)
Physcial Address In Binary:
|-----PPN-----|----PPO----|
|0 |1 |0|1|0|1|0|0|0|1|0|1|
|----CT-------|--CI---|-CO|

Cach Offset: 1 (decimal)
Cache Index:  1 (decimal), 1 (hex)
Cache Tag:  21 (decimal), 15 (hex)
Cache Hit (Y/N): Y
Byte: 21

E.) 0x57B

Virtual Address in Binary:
|-----VPN------------|----VPO----|
|0 |0 |0 |1 |0|1|0|1 |1|1|1|0|1|1|
|-----TLBT------|TLBI|

VPN:  21 (decimal),  15 (Hex)
TLBI:  01 (binary),  1 (decimal), 1 (hex)
TLBT:  000101 (binary), 5 (decimal), 5 (hex)
TLB Hit (Y/N): N
Page Fault (Y/N): N
PPN: 1001 (binary),  09 (decimal),  09 (hex)
Physcial Address In Binary:
|-----PPN-----|----PPO----|
|0 |0 |1|0|0|1|1|1|1|0|1|1|
|----CT-------|--CI---|-CO|

Cach Offset: 3 (decimal)
Cache Index:  14 (decimal), E (hex)
Cache Tag:  9 (decimal), 9 (hex)
Cache Hit (Y/N): N
Byte: mem
```

Problem 3:

```
a.) find_range

...
 for (int i = 0; i < length; i++) {
   max = arr[i] < max ? max : arr[i];
   min = arr[i] < min ? arr[i] : min;
 }
...

b.) bubble_sort

...
 for (i = 0; i < length - j - 2; i++) {
   int first_swap = arr[i] > arr[i+1] ? arr[i+1] : arr[i];
   int second_swap = arr[i] > arr[i+1] ? arr[i] : arr[i+1];
   arr[i] = first_swap;
   arr[i+1] = second_swap;
 }
...
```

In practice, these optimizations will show a significant decrease in branching when being used on increasingly large input sizes. On smaller inputs, results of the optimization may not be actually that evident. For example, if we measure the amount of branching that occurs in the bubble sort program with an array of size 100, both versions of the function actually branch around the same amount. When we increase this size to around 10,000 , the function using conditional moves branches almost exactly half the time the unoptimized function branches.

Problem 4: (This problem is related to the textbook's 9.10)

As stated in the question, there are many sequences of mmap and munmap calls that can result in the scenario specified. One example would be repeated calls to mmap with 50 MB until the entire region mmap uses is taken up, or in other words, a request of 50 MB finally fails. Then, make repeated calls to munmap on every other block. This will lead to the entire region being filled with alternating allocated and free blocks of 50 MB. There will obviously be well over 100 MB free in the region mmap uses, but none of it will be contiguous for 100 MB. Therefore the request cannot be satisfied.