## 8/8 Questions Answered

Saved at 1:26 PM

# Check-in Quiz 04, memory allocaiton, caches, threads

### Q1

**1 Point**

When will MMU evict an entry from TLB? Select all that apply.

- ☐ When a requested virtual page number is missed in the TLB and TLB is full.
- ☐ When a requested virtual page number is hit in the TLB.
- ☐ When an entry in the page table is evicted from memory to swap file and the same address is also evicted from the TLB.

**Explanation**

Correct! After a TLB miss, MMU will evict one of the entry from TLB and replaces with the new entry from the page table. So option A is correct. The TLB will not evict an entry when it is looked up in the TLB since TLB's follow LRU style replacement policies, so option B is not correct. During a page fault, MMU will use page replacement policy to swap a page from memory to disk, if the address of the page is also stored in TLB. It should be evicted as well. So option C is correct.

Save Answer Last saved on **Oct 09 at 1:25 PM**

### Q2

**1 Point**

The best-fit memory allocation scheme always results in less fragmentation and satisfies more block requests compared to the first-fit scheme.

True

False

**Explanation**

Correct! In practice, it turns out that there is no fixed consensus on which is the best technique. This is highly dependent on the workload.

Save Answer Last saved on **Oct 09 at 1:25 PM**

## Q3
**2 Points**

Lets say that someone wanted to allocate 99 contiguous pages using the buddy algorithm. You can assume that the buddy algorithm is able to complete this allocation request.

### Q3.1
**1 Point**

How many pages would be allocated by the buddy algorithm?

Please simplify your answer to a number (e.g. `1337`), do not leave any exponents or multiplication symbols in your answer.

128

**Explanation**

Correct! The buddy algorithm only allocates in powers of 2, so 99 would be rounded up to the next power of 2 which is 2^7 = 128

Save Answer Last saved on **Oct 09 at 1:25 PM**

**Q3.2**
**1 Point**

How many pages of the allocation would be fragmentation?

Please simplify your answer to a number (e.g. `1337`), do not leave any exponents or multiplication symbols in your answer.

29

**Explanation**

Correct! we only requested 99 pages, so 128 - 99 = 29 pages would be internal fragmentation.

Save Answer  Last saved on **Oct 09 at 1:25 PM**

## Q4 Cache
**4 Points**

For this question, assume that we are working in an architecture that has 64-byte cache lines.

Lets say we had the following C array:

```
// array of bytes. Array is 1024 long
uint8_t data[1024];
```

For simplicity, assume that:

- our L1 cache can only hold 1 cache line of the array data at a time
- any other data accessed outside of the array is inside the cache already (and won't be evicted)
- the array is aligned to a cache line (the first byte in the array is the beginning of a new cache line)

Lets say we had the following C Code to initialize the array:

```
// array of bytes. Array is 1024 long
uint8_t data[4096];
```

```
for (int i = 0; i < 1024; i++) {
  data[i] = rand()  % 256;  // assign it a random number
}
```

**Q4.1**
**2 Points**

How many times will the cache miss (data we access is not in the cache) when setting `data[i]`?

Assume that the cache originally contains no cache lines that contain a part of the `data` array.

16

**Explanation**

Correct! When we access the first element of the data array, we get a cache miss since the cache starts empty. That brings in the cache line containing `data[0]` and we then hit for the next 63 bytes (since a cache line is 64 bytes). When we access `data[64]` we get another cache miss but hit again for the next 63 bytes. From this we can see we miss once every 64 bytes, giving us 1024 / 64 = 16 cache misses.

Save Answer Last saved on **Oct 09 at 1:25 PM**

**Q4.2**
**2 Points**

How many times will the cache hit (data we access is in the cache) when setting `data[i]`?

Assume that the cache originally contains no cache lines that contain a part of the `data` array.

1008

**Explanation**

Correct! When we access the first element of the data array, we get a cache miss since the cache starts empty. That brings in the cache line containing `data[0]` and we then hit for the next 63 bytes (since a cache line is 64 bytes). When we access `data[64]` we get another cache miss but hit again for the next 63 bytes. From this we can see we miss once every 64 bytes, giving us 1024 / 64 = 16 cache misses. 1024 accesses - 16 misses gets us 1008 hits. You could also read this a hitting every 63/64 accesses, getting us 1024 * (63 / 64) = 1008 cache hits.

Save Answer   Last saved on **Oct 09 at 1:26 PM**

## Q5 Threads vs Processes

**2 Points**

### Q5.1

**1 Point**

Which of these are unique to every thread?

- ☐ Data in Files on Disk
- ☐ File Descriptors
- ☐ Global Variables
- ☐ Dynamic Storage (The Heap)
- ☐ Local Variables (The Stack)
- ☐ CPU Registers

**Explanation**

Correct! Each thread has its own stack and registers, but share an address space and other attributes that are specific to a process. Note that while each thread has its own stack, a thread could still be given a pointer to some value in another threads stack if we wanted.

Save Answer Last saved on **Oct 09 at 1:26 PM**

**Q5.2**
**1 Point**

Which of these are unique to every process?

- ☐ Data in Files on Disk
- ☐ File Descriptors
- ☐ Global Variables
- ☐ Dynamic Storage (The Heap)
- ☐ Local Variables (The Stack)
- ☐ CPU Registers

**Explanation**

Correct! Each process has its own address space (and the thread(s) within it have their own registers). Each process also has its own file descriptor table, but the actual contents of data in the files are accessible by any process with appropriate permissions

Save Answer Last saved on **Oct 09 at 1:26 PM**

Save All Answers

Submit & View Submission ›