Virtual Memory

Annalise Tarhan

October 5, 2020

9 Homework Problems

9.11 Translate the virtual address 0x027c into a physical address and determine whether the TLB misses, whether a page fault occurs, and whether a cache miss occurs.

Virtual address format 00 0010 0111 1100

VPN 0x09

TLB index 0x01

TLB tag 0x02

TLB hit? no

Page fault? no

PPN 0x17

Physical address format 0101 1111 1100

Byte offset 0x00

Cache index 0x0F

Cache tag 0x17

Cache hit? no

Cache byte returned –

9.12 Repeat for virtual address 0x03a9.

Virtual address format 00 0011 1010 1001

VPN 0x0E

TLB index 0x02

TLB tag 0x03

TLB hit? no

Page fault? no

PPN 0x11

Physical address format

Byte offset 0x01

Cache index 0x0A

Cache tag 0x11

Cache hit? no

Cache byte returned –

9.13 Repeat for virtual address 0x0040.

Virtual address format 00 0000 0100 0000

VPN 0x01

TLB index 0x01

TLB tag 0x00

TLB hit? no

Page fault? yes

9.14 Given an input file hello.txt that consists of the string Hello, world!\n, write a C program that uses mmap to change the contents of hello.txt to Jello, world!\n.

```
int main(int argc, char *argv[]) {
        int file = open(argv[1], O_RDWR);
        char *write_here = mmap(0, 9, PROT_WRITE,
         MAP_SHARED, file, 0);
        write_here[0] = 'J';
        close(file);
        return 0;
}
```

9.15 Determine the block sizes and header values that would result from the following sequence of malloc requests.

Request	Block size (decimal bytes)	Block header (hex)
malloc(3)	8	0x09
malloc(11)	16	0x11
malloc(20)	24	0x19
malloc(21)	32	0x21

9.16 Determine the minimum block size for each of the following alignment requirements and block formats, based on the given assumptions.

Alignment	Allocated block	Free block	Minimum block size (bytes)
Single word	Header and footer	Header and footer	16
Single word	Header, but no footer	Header and footer	16
Double word	Header and footer	Header and footer	16
Double word	Header, but no footer	Header and footer	16

9.17 Develop a version of the allocator that performs a next-fit search instead of a first-fit search.

A next-fit search requires the allocator to maintain a pointer to the next block after the most recently allocated one, last_visitedp.

```
static void *find_next_fit(size_t asize) {
        void *rover;

        for (rover = last_visitedp;
        GET_SIZE(HDRP(rover)) > 0;
        rover = NEXT_BLKP(rover)) {
                if (!GET_ALLOC(HDRP(rover))
                && (asize <= GET_SIZE(HDRP(rover)))) {
                        last_visitedp = NEXT_BLKP(rover);
                        return rover;
                }
        }
        for (rover = heap_listp;
        rover < last_visitedp;
        rover = NEXT_BLKP(rover)) {
                if (!GET_ALLOC(HDRP(rover))
                && (asize <= GET_SIZE(HDRP(rover)))) {
                        last_visitedp = NEXT_BLKP(rover);
                        return rover;
                }
        }
        return NULL;
}
```

It also requires updating the coalesce function so the last_visitedp doesn't end up pointing to the middle of a block.

```
if (last_visitedp > bp && last_visitedp < next_alloc) {
        last_visitedp = bp;
}
```

9.18 Modify the allocator so that free blocks require a header and a footer, but allocated blocks require only a header.

Removing the footer from allocated blocks means footers will no longer be placed in the block when the block is allocated.

```
static void place(void *bp, size_t asize) {
        size_t csize = GET_SIZE(HDRP(bp));
        if ((csize - asize) >= (2*DSIZE)) {
                PUT(HDRP(bp), PACK(asize, 1));
                bp = NEXT_BLKP(bp);
                PUT(HDRP(bp), PACK(csize-asize, 0));
                PUT(FTRP(bp), PACK(csize-asize, 0));
        } else {
                PUT(HDRP(bp), PACK(csize, 1));
                PUT(FTRP(bp), PACK(csize, 1));
        }
}
```

Coalesce will check the header instead of the footer of previous blocks.

```
size_t prev_alloc = GET_ALLOC(HDRP(PREV_BLKP(bp)));
size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
```

This also changes the alignment requirements. Free blocks still need space for the header and the footer: two words. Allocated blocks only need room for the header and some non-zero amount of space for data: also two words. This means the minimum block size is now only a double word.

```
if (size <= WSIZE) {
        asize = DSIZE;
} else {
        asize = DSIZE*((size+(WSIZE)+(DSIZE-1)) / DSIZE);
}
```

9.19 For each group, indicate which statement is true.

9.19.1

(d) The buddy system suffers from internal fragmentation, but not external fragmentation.

9.19.2

(d) Using the first-fit algorithm on a free list that is ordered according to increasing block sizes is equivalent to using the best-fit algorithm.

9.19.3 Mark&Sweep garbage collectors are called conservative if

(b) They treat everything that looks like a pointer as a pointer.