



```
void *
calloc(uint nbytes) {
    void *ptr = malloc(nbytes);

    if (ptr != 0) {
        // each char is 1 byte
        char *char_ptr = (char *) ptr;
        for (uint i = 0; i < nbytes; ++i) {
            char_ptr[i] = 0;
        }
    }

    return ptr;
}

void *
realloc(void *ap, uint nbytes) {
    if (nbytes == 0 && ap != 0) {
        free(ap);
        return 0;
    } else if (nbytes != 0 && ap == 0) {
        return malloc(nbytes);
    } else if (nbytes != 0 && ap != 0) {
        void* ptr = malloc(nbytes);
        if (ptr != 0) {
            char* dest = (char*) ptr;
            const char* src = (const char*) ap;

            for (uint i = 0; i < nbytes; ++i) {
                dest[i] = src[i];
            }
            free(ap);
        }
        return ptr;
    }
    return 0;
}
```

کد مربوط به دو تابع calloc و realloc را در تصویر بالا مشاهده می‌کنید.

```
amir$ alloctest
alloctest: calloc tested successfully
alloctest: realloc tested successfully
amir$ aQEMU: Terminated
```

```

}

// Allocate a usyscall page.
if((p->usyscall = (struct usyscall *)kalloc()) == 0) {
    freeproc(p);
    release(&p->lock);
    return 0;
}
(p->usyscall)->pid = p->pid;

```

```

freeproc(struct proc *p)
{
    if(p->trapframe)
        kfree((void*)p->trapframe);
    if(p->usyscall)
        kfree((void*)p->usyscall);
    p->trapframe = 0;
    if(p->pagetable)
        proc_freepagetable(p->pagetable, p->sz);

```

```

}

if(mappages(pagetable, USYSCALL, PGSIZE,
            (uint64)p->usyscall, PTE_R | PTE_U) < 0) {
    uvmunmap(pagetable, TRAMPOLINE, 1, 0);
    uvmunmap(pagetable, TRAPFRAME, 1, 0);
    uvmfree(pagetable, 0);
    return 0;
}

return pagetable;
}

```

```

proc_freepagetable(pagetable_t pagetable, uint64 sz)
{
    uvmunmap(pagetable, TRAMPOLINE, 1, 0);
    uvmunmap(pagetable, TRAPFRAME, 1, 0);
    uvmunmap(pagetable, USYSCALL, 1, 0);
    uvmfree(pagetable, sz);
}

```

کد های تغییر داده شده برای سوال دوم را تصاویر بالا مشاهده می کنید.

در پاسخ به سوال مطرح شده باید گفت فراخوانی هایی که نیاز به write بر روی و صرفا نیاز به خواندن از حافظه دارند می توانند از این صفحه مشترک استفاده کنند. Syscall های دیگری که مانند getpid اطلاعات پردازش ها یا سیستم را به کاربر می دهند می توانند گزینه خوبی باشند.

۳.

```

void
vmprint(pagetable_t pagetable)
{
    printf("page table %p\n", pagetable);
    walk_and_print(pagetable, 0);
}

```

```

void
walk_and_print(pagetable_t pagetable, int depth)
{
    // there are 2^9 = 512 PTEs in a page table.
    for(int i = 0; i < 512; i++){
        pte_t pte = pagetable[i];
        if((pte & PTE_V) && (pte & (PTE_R|PTE_W|PTE_X)) == 0){
            // this PTE points to a lower-level page table.
            uint64 child = PTE2PA(pte);
            print_pagetable(i, depth, pte, child);
            walk_and_print((pagetable_t)child, depth+1);
        } else if(pte & PTE_V){
            uint64 child = PTE2PA(pte);
            print_pagetable(i, depth, pte, child);
        }
    }
}

```

```

void
print_pagetable(int index , int depth , uint64 pte , uint64 pa){
    for(int i = 0; i < depth; ++i) {
        printf(".. ");
    }
    printf("..");
    printf("%d: pte %p pa %p\n", index, pte, pa);
}

```

```

uint64      uvmdealloc(pagetable_t, uint64, uint64);
int         uvmcopy(pagetable_t, pagetable_t, uint64);
void        vmprint(pagetable_t pagetable);
void        uvmfree(pagetable_t, uint64);
void        uvmunmap(pagetable_t, uint64, uint64, int);

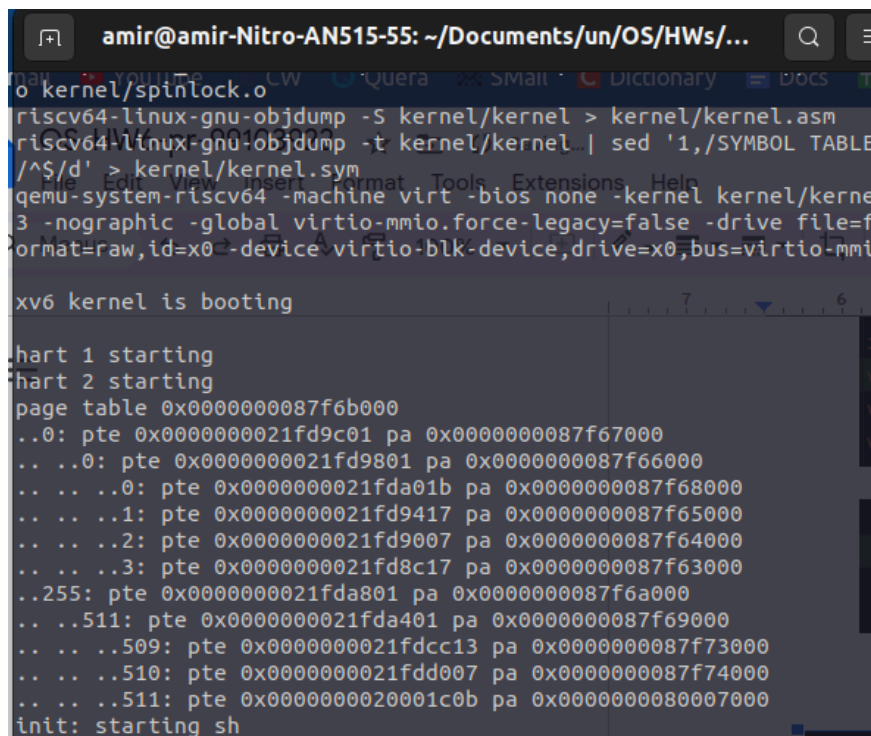
```

```

if(p->pid==1) vmprint(p->pagetable);
return argc; // this ends up in a0, the first argument to main(argc)

```

کد های اضافه شده برای پاسخ به این سوال را در تصاویر بالا می‌توانید مشاهده کنید.



```

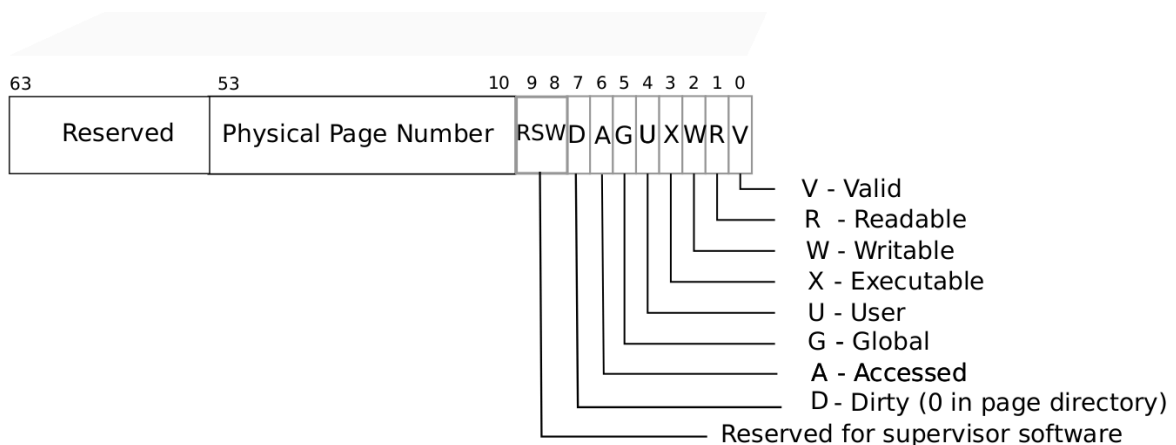
amir@amir-Nitro-AN515-55: ~/Documents/un/OS/HWs/...
o kernel/spinlock.o
riscv64-linux-gnu-objdump -S kernel/kernel > kernel/kernel.asm
riscv64-linux-gnu-objdump -t kernel/kernel | sed '1,/SYMBOL TABLE
/^$/d' > kernel/kernel.sym
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kerne
3 -nographic -global virtio-mmio.force-legacy=false -drive file=f
format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio
xv6 kernel is booting

hart 1 starting
hart 2 starting
page table 0x0000000087f6b000
..0: pte 0x0000000021fd9c01 pa 0x0000000087f67000
.. ..0: pte 0x0000000021fd9801 pa 0x0000000087f66000
.. .. ..0: pte 0x0000000021fda01b pa 0x0000000087f68000
.. .. ..1: pte 0x0000000021fd9417 pa 0x0000000087f65000
.. .. ..2: pte 0x0000000021fd9007 pa 0x0000000087f64000
.. .. ..3: pte 0x0000000021fd8c17 pa 0x0000000087f63000
..255: pte 0x0000000021fda801 pa 0x0000000087f6a000
.. ..511: pte 0x0000000021fda401 pa 0x0000000087f69000
.. .. ..509: pte 0x0000000021fdcc13 pa 0x0000000087f73000
.. .. ..510: pte 0x0000000021fdd007 pa 0x0000000087f74000
.. .. ..511: pte 0x0000000020001c0b pa 0x0000000080007000
init: starting sh

```

```
amir@amir-Nitro-AN515-55: ~/Documents/un/OS/HWs/...
kernel/pipe.o kernel/exec.o kernel/sysfile.o kernel/kernelvec.o
kernel/virtio_disk.o kernel/start.o kernel/console.o kernel/printf
o kernel/spinlock.o
riscv64-linux-gnu-objdump -S kernel/kernel > kernel/kernel.asm
riscv64-linux-gnu-objdump -t kernel/kernel | sed '1,/SYMBOL TABLE
/^$/d' > kernel/kernel.sym
make[1]: Leaving directory '/home/amir/Documents/un/OS/HWs/hw6/xy
== Test pgtbltest ==
$ make qemu-gdb
(2.2s)
== Test  pgtbltest: ugetpid ==
pgtbltest: ugetpid: OK
== Test  pgtbltest: pgaccess ==
pgtbltest: pgaccess: FAIL
...
ugetpid_test starting
ugetpid_test: OK
pgaccess_test starting
pgtbltest: pgaccess_test failed: incorrect access bits s
$ qemu-system-riscv64: terminating on signal 15 from pid
MISSING '^pgaccess_test: OK$'
== Test pte printout ==
$ make qemu-gdb
pte printout: OK (0.5s)
== Test answers-pgtbl.txt ==
answers-pgtbl.txt: FAIL
Cannot read answers-pgtbl.txt
== Test usertests ==
```

همچنین بررسی های ممکن برای سلامت کد را نیز می‌توانید در تصاویر بالا مشاهده کنید.  
در پاسخ به سوال مطرح شده در سوال ۳ باید به تصویر زیر توجه کنیم که هر یک از بیت های pte را بررسی می‌کند.



اینکه در نمایش hex عدد کم ارزش ترین رقم هر pte یک عدد فرد است نشان می‌دهد که تمام pte های چاپ شده دارای کم ارزش ترین بیت برابر ۱ هستند و همگی pte های valid هستند.

همچنین طبق منطقی که برای این بخش وجود دارد هر سه مقدار X و W و R برای pte های که برگ نیستند برابر صفر است.

برای ۳ pte آخر داریم:

.....0011 : not executable , not writable , only readable

.....0111: not executable , writable and readable

.....1011: excusable and readable but not writable

همچنین برای بعضی از pte ها رقم دوم در نمایش hex برابر ۱ است که این یعنی این pte ها از مود یوزر قابل دسترسی هستند اما برای بعضی دیگر صفر است که عکس آن معنی را دارد.

برای هر سوال ۲ و ۳ عملی بنده یک فایل آپلود کردم که شامل دو کامیت می‌باشد.

