

فاز اول پروژه سیستم عامل

مهدی منصوری خواه 9931056

توضیح کلی از فرایند های طی شده برای ساخته شدن اولین process :

```
void
main()
{
    if(cpuid() == 0){
        consoleinit();
        printfinit();
        printf("\n");
        printf("xv6 kernel is booting\n");
        printf("\n");
        kinit();           // physical page allocator
        kvminit();          // create kernel page table
        kvminithart();      // turn on paging
        procinit();         // process table
        trapinit();         // trap vectors
        trapinithart();     // install kernel trap vector
        plicinit();         // set up interrupt controller
        plicinithart();     // ask PLIC for device interrupts
        binit();            // buffer cache
        iinit();            // inode table
        fileinit();         // file table
        virtio_disk_init(); // emulated hard disk
        userinit();         // first user process
        __sync_synchronize();
        started = 1;
    } else {
        while(started == 0)
            ;
        __sync_synchronize();
    }
}
```

همانطور که میبینیم تنها cpu که id برابر 0 دارد مسئول init کردن است (userinit())

```
void
userinit(void)
{
    struct proc *p;

    p = allocproc();
    initproc = p;

    // allocate one user page and copy initcode's instructions
    // and data into it.
    uvmfirst(p->pagetable, initcode, sizeof(initcode));
    p->sz = PGSIZE;
```

```

// prepare for the very first "return" from kernel to user.
p->trapframe->epc = 0;      // user program counter
p->trapframe->sp = PGSIZE;  // user stack pointer

safestrcpy(p->name, "initcode", sizeof(p->name));
p->cwd = namei("/");

p->state = RUNNABLE;

release(&p->lock);
}

```

در userinit() تابع allocproc() صدا زده می شود :

```

static struct proc*
allocproc(void)
{
    struct proc *p;

    for(p = proc; p < &proc[NPROC]; p++) {
        acquire(&p->lock);
        if(p->state == UNUSED) {
            goto found;
        } else {
            release(&p->lock);
        }
    }
    return 0;

found:
    p->pid = allocpid();
    p->state = USED;

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

    // An empty user page table.
    p->pagetable = proc_pagetable(p);

```

```

if(p->pagetable == 0){
    freeproc(p);
    release(&p->lock);
    return 0;
}

// Set up new context to start executing at forkret,
// which returns to user space.
memset(&p->context, 0, sizeof(p->context));
p->context.ra = (uint64)forkret;
p->context.sp = p->kstack + PGSIZE;

return p;
}

```

در تابع `allocproc()`، کل `process` های موجود در آرایه پیمایش می شوند و اگر `process` غیر فعالی وجود داشت، فیلدهای یک `proc` با مقادیر لازم مقداردهی می شوند.

مراحل طی شده برای فراخوانی سیستم کال

ابتدا تغییرات فایل `kernel` رو بررسی می کنیم.

ابتدا در فایل `syscall.h` این سیستم کال رو تعریف میکنیم و همچنین یک شماره برای آن `set` میکنیم (22)

```
#define SYS_kfreememfunc 22
```

سپس در خط 105 در فایل `syscall.c` این قطعه کد را اضافه می کنیم و آن را `extern` می کنیم.

```
extern uint64 sys_kfreememfunc(void);
```

همچنین در خط 131 همون فابل این قسمت را اضافه می کنیم.

```
[SYS_kfreememfunc] sys_kfreememfunc,
```

حال باید یک فایل C. اضافه کنیم و در آن به محاسبه مقدار فضای خالی مموری پردازیم. همانطور که در دستور کار گفته شد یک راه آن بررسی تک تک پردازش های سیستم و جمع کردن سائز مموری آنها می باشد.

```
extern struct proc proc[NPROC];

uint64
sys_kfreememfunc(void)
{
    uint64 used = 0;
    struct proc *p;

    for (p = proc; p < &proc[NPROC]; ++p) {

        if (p->state == UNUSED)
            continue;

        acquire(&p->lock);

        used += p->sz;

        release(&p->lock);
    }

    return (128*1024*1024) - used;
}
```

در اینجا یک for روی پردازنده ها زدیم و آن هایی که در حال اجرا بودن رو سائزشون رو با هم جمع کردیم و در آخر از سائز مموری کم می کنیم که این مقدار نزدیک مقدار نظری از پیش تعیین شده است.

از اطلاعات فایل memlayout استفاده کردیم:

```
// the kernel expects there to be RAM
// for use by the kernel and user pages
// from physical address 0x80000000 to PHYSTOP.
#define KERNBASE 0x80000000L
#define PHYSTOP (KERNBASE + 128*1024*1024)
```

حال تغییرات فایل user را انجام می دهیم:

در فایل user.h این قطعه کد را در خط 25 اضافه می کنیم.

```
uint64 kfreememfunc(void);
```

سپس در فایل usys.pl این سیستم کال را می شناسانیم:

```
entry("kfreememfunc");
```

حال یک فایل c. اضافه می کنیم و خروجی را مشخص می کنیم.

```
int main() {
    uint64 free = kfreememfunc();
    uint64 all = PHYSTOP - KERNBASE; //128*1024*1024
    printf("free memory: %d .\n", free);
    printf("all memory: %d .\n", all);
    exit(0);
}
```

و در آخر به Makefile این دو قسمت را اضافه می کنیم:

در OBJS

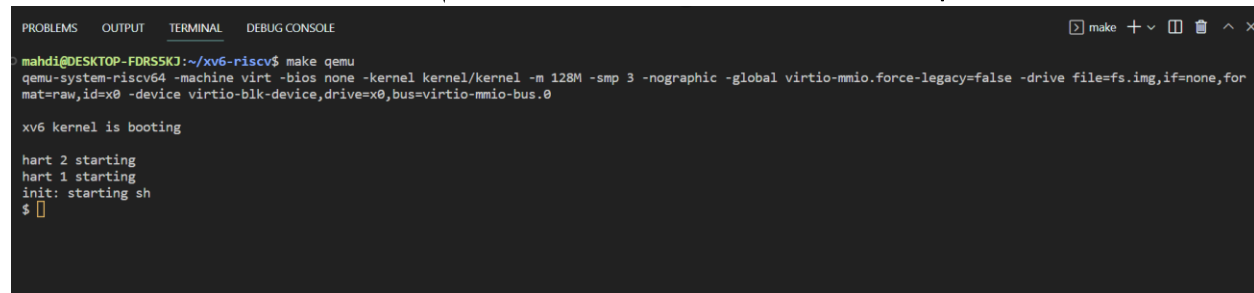
```
$K/kfreememfunc.o
```

در UPROGS

```
$U/_kfreemem\
```

حال یک خروجی می گیریم:

ابتدا با دستور make qemu. xv6 را اجرا می کنیم



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
mahdi@DESKTOP-FDR55KJ:~/xv6-riscv$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$
```

سپس دستور ls وارد می کنیم

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ ls
.          1 1 1024
..         1 1 1024
README    2 2 2305
cat        2 3 32896
echo       2 4 31752
forktest   2 5 15896
grep       2 6 36264
init       2 7 32240
kill       2 8 31712
ln         2 9 31536
ls         2 10 34840
mkdir      2 11 31768
rm         2 12 31752
sh         2 13 54192
stressfs   2 14 32632
usertests  2 15 180528
grind      2 16 47584
wc         2 17 33840
zombie     2 18 31112
kfreemem   2 19 31408
console    3 20 0
$
```

حال kfreemem وارد می کنیم

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
hart 2 starting
init: starting sh
$ ls
.          1 1 1024
..         1 1 1024
README    2 2 2305
cat        2 3 32896
echo       2 4 31752
forktest   2 5 15896
grep       2 6 36264
init       2 7 32240
kill       2 8 31712
ln         2 9 31536
ls         2 10 34840
mkdir      2 11 31768
rm         2 12 31752
sh         2 13 54192
stressfs   2 14 32632
usertests  2 15 180528
grind      2 16 47584
wc         2 17 33840
zombie     2 18 31112
kfreemem   2 19 31360
console    3 20 0
$ kfreemem
free memory: 134164480 .
all memory: 134217728 .
$
```

برای توضیحات بیشتر و همچنین دیدن کامل کد ها میتوانید به لینک گیت هاب <https://github.com/mahdimnkh81/XV6> مراجعه کنید.