

اولین متد این فایل که consoleinit است و برای مقداردهی اولیه سخت افزار استفاده میشود
دومین متد این فایل که printfinit است میروود لاک pr را 1 قرار میدهد

```
void
printfinit(void)
{
    initlock(&pr.lock, "pr");
    pr.locking = 1;
}
```

سپس متد kinit اجرا میشود در این متد ابتدا لاک استراکت kmem باز میشود و سپس متد freerange صدا زده میشود در این متد به تعداد PGSIZE متد kfree صدا زده میشود که در متد kfree با استفاده از memset حافظه در مموری گرفته میشود

```
void
kinit()
{
    initlock(&kmem.lock, "kmem");
    freerange(end, pa_end: (void*)PHYSTOP);
}

void
freerange(void *pa_start, void *pa_end)
{
    char *p;
    p = (char*)PGROUNDUP(SZ: (uint64)pa_start);
    for(; p + PGSIZE <= (char*)pa_end; p += PGSIZE)
        kfree(p);
}

// Free the page of physical memory pointed at by pa,
// which normally should have been returned by a
// call to kalloc(). (The exception is when
// initializing the allocator; see kinit above.)
```

```

// Free the page of physical memory pointed at by pa,
// which normally should have been returned by a
// call to kalloc(). (The exception is when
// initializing the allocator; see kinit above.)
void
kfree(void *pa)
{
    struct run *r;

    if(((uint64)pa % PGSIZE) != 0 || (char*)pa < end || (uint64)pa >= PHYSTOP)
        panic("kfree");

    // Fill with junk to catch dangling refs.
    memset(pa, 1, PGSIZE);

    r = (struct run*)pa;

    acquire(&kmem.lock);
    r->next = kmem.freelist;
    kmem.freelist = r;
    release(&kmem.lock);
}

```

متد kvm_{init} یک جدول صفحه هسته را با استفاده از kvm_{ma} ایجاد کند. kvm_{make} ابتدا یک صفحه از حافظه فیزیکی را برای نگهداری صفحه اصلی صفحه جدول اختصاص می دهد. سپس kvm_{map} را فراخوانی می کند تا ترجمه های هسته را نصب کند

```

kvmmake(void)
{
    pagetable_t kpgtbl;

    kpgtbl = (pagetable_t) kalloc();
    memset(kpgtbl, 0, PGSIZE);

    // uart registers
    kvmmap(kpgtbl, UART0, UART0, PGSIZE, perm: PTE_R | PTE_W);

    // virtio mmio disk interface
    kvmmap(kpgtbl, VIRTIO0, VIRTIO0, PGSIZE, perm: PTE_R | PTE_W);

    // PLIC
    kvmmap(kpgtbl, PLIC, PLIC, sz: 0x400000, perm: PTE_R | PTE_W);

    // map kernel text executable and read-only.
    kvmmap(kpgtbl, KERNBASE, KERNBASE, sz: (uint64)etext-KERNBASE, perm: PTE_R | PTE_X);

    // map kernel data and the physical RAM we'll make use of.
    kvmmap(kpgtbl, (uint64)etext, (uint64)etext, sz: PHYSTOP-(uint64)etext, perm: PTE_R | PTE_W);

    // map the trampoline for trap entry/exit to
    // the highest virtual address in the kernel.
    kvmmap(kpgtbl, va: TRAMPOLINE, (uint64)trampoline, PGSIZE, perm: PTE_R | PTE_X);
}

```

kvminithart برای نصب page table هسته سیستم عامل است. آدرس فیزیکی صفحه اصلی جدول را در ثبات satp می نویسد.

procinit جدول مربوط به هر پراسه را مقدار دهی میکند pcb

```

// initialize the proc table.
void
procinit(void)
{
    struct proc *p;

    initlock(&pid_lock, "nextpid");
    initlock(&wait_lock, "wait_lock");
    for(p = proc; p < &proc[NPROC]; p++) {
        initlock(&p->lock, "proc");
        p->state = UNUSED;
        p->kstack = KSTACK(p: (int) (p - proc));
    }
}

```

trapinit میاد یک vectors از حالت های trap ما مقدار دهی میکند

trapinit hart میاد حالت trap را برای هسته نصب و درست میکند

install kernel trap vector

تابع binit حافظه نهان یا کش بافر را که یک doubly-linked list است را با NBUF در آرایه استاتیک buf مقداردهی اولیه می کند.

متد fileinit میرود لاک pr را مقدار دهی اولیه میکند

```
void
fileinit(void)
{
    initlock(&ftable.lock, "ftable");
}
```

virtio_disk_init شبیه سازی از دیسک درست میکند

sync_synchronize_ یک مانع حافظه است و به کامپایلر و CPU می گوید order را مجدداً لود یا ذخیره نکنند

```

// Set up first user process.
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);
}

```

این متد یک پردازش را میسازد و آن را آماده اجرا میکند در خط اول ما یک استراک از پردازش که شامل اطلاعات عکس زیر است میسازیم

```

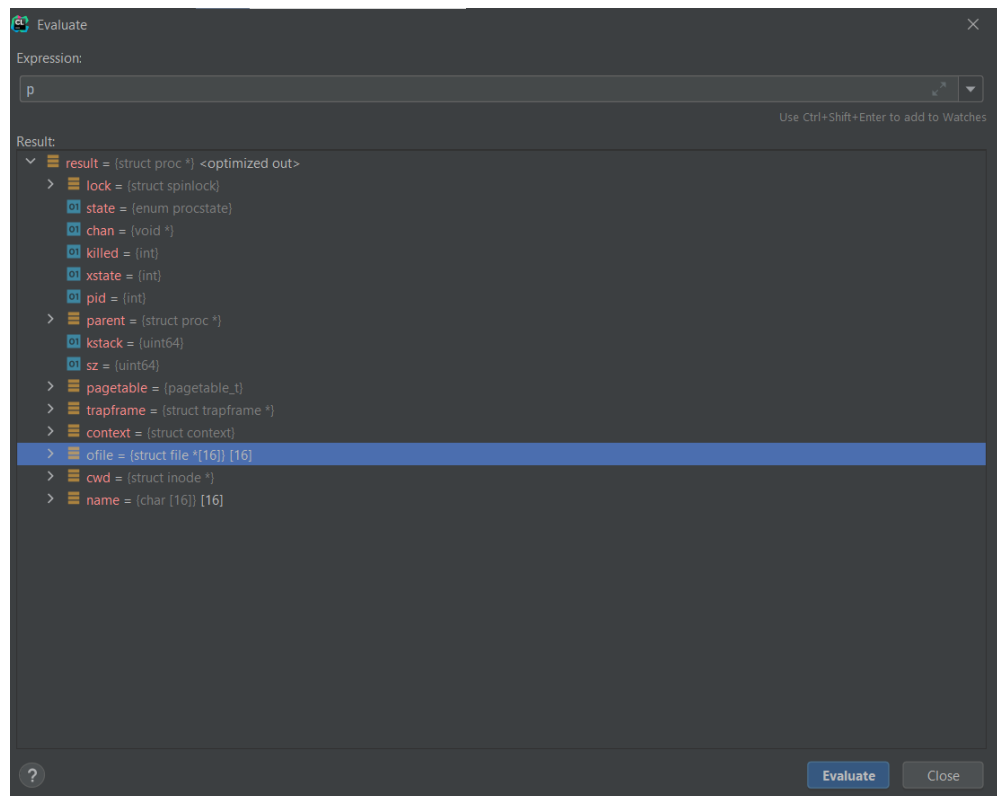
struct proc {
    struct spinlock lock;

    // p->lock must be held when using these:
    enum procstate state;           // Process state
    void *chan;                     // If non-zero, sleeping on chan
    int killed;                     // If non-zero, have been killed
    int xstate;                     // Exit status to be returned to parent's wait
    int pid;                         // Process ID

    // wait_lock must be held when using this:
    struct proc *parent;            // Parent process

    // these are private to the process, so p->lock need not be held.
    uint64 kstack;                  // Virtual address of kernel stack
    uint64 sz;                      // Size of process memory (bytes)
    pagetable_t pagetable;          // User page table
    struct trapframe *trapframe;    // data page for trampoline.S
    struct context context;          // swtch() here to run process
    struct file *ofile[NOFILE];     // Open files
    struct inode *cwd;               // Current directory
    char name[16];                  // Process name (debugging)
};

```



سپس متد allocproc را صدا میزنیم که در این متد میره سراغ پردازش های که UNUSED هستند را پیدا میکند و اگر پیدا شدن آنها را در حالت اجرا برای هسته در میآورد
متد allocpid یک pid جدید ست میکند

```

// Look in the process table for an UNUSED proc.
// If found, initialize state required to run in the kernel,
// and return with p->lock held.
// If there are no free procs, or a memory allocation fails, return
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;
}

```

سپس بادوتا if بعدی به ترتیب یک trap frame و بعد یک (page table) pcb برای پراسه درست میکنیم
و بعد به اندازه این اطلاعات در مموری با memset حافظه ذخیره میکنیم


```

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;
}

```

اگره پرداز از قبل page table یا trap frame داشته باشد با متد freeproc مقدار آنها را بای دیفالت میکنیم

```
// free a proc structure and the data hanging from it,  
// including user pages.  
// p->lock must be held.  
static void  
freeproc(struct proc *p)  
{  
    if(p->trapframe)  
        kfree((void*)p->trapframe);  
    p->trapframe = 0;  
    if(p->pagetable)  
        proc_freepagetable(p->pagetable, p->sz);  
    p->pagetable = 0;  
    p->sz = 0;  
    p->pid = 0;  
    p->parent = 0;  
    p->name[0] = 0;  
    p->chan = 0;  
    p->killed = 0;  
    p->xstate = 0;  
    p->state = UNUSED;  
}
```

```
Evaluate
Expression:
p

Result:
result = {struct proc * | 0x80010f60} 0x80010f60
  lock = {struct spinlock}
  state = {enum procstate} USED
  chan = {void * | 0x0} NULL
  killed = {int} 0
  xstate = {int} 0
  pid = {int} 1
  parent = {struct proc * | 0x0} NULL
  kstack = {uint64} 274877894656
  sz = {uint64} 0
  pagetable = {pagetable_t | 0x87f73000} 0x87f73000
  trapframe = {struct trapframe * | 0x87f74000} 0x87f74000
  context = {struct context}
  ofile = {struct file *[16]}
  cwd = {struct inode * | 0x0} NULL
  name = {char [16]}
```

با استفاده از `uvmfirst` با اندازه PGSIZE در مموری حافظه میگیریم برای پردازش

```

// for the very first process.
// sz must be less than a page.
void
uvmfirst(pagetable_t pagetable, uchar *src, uint sz)  pagetable: 0x87f73000    src: "\027\005"    sz: 52
{
    char *mem;    mem: ""

    if(sz >= PGSIZE)
        panic("uvmfirst: more than a page");
    mem = kalloc();
    memset(mem, 0, PGSIZE);
    mappages(pagetable, 0, PGSIZE, (uint64)mem, perm: PTE_W|PTE_R|PTE_X|PTE_U);  pagetable: 0x87f73000
    memmove(mem, src, sz);  sz: 52    src: "\027\005"    mem: ""
}


```

```

// Like strcpy, but guaranteed to NOT terminate.
char*
safestrcpy(char *s, const char *t, int n)  s: ""    t: ""    n: <optimized out>
{
    char *os;    os: "initcode"

    os = s;
    if(n <= 0)
        return os;    os: "initcode"
    while(--n > 0 && (*s++ = *t++) != 0)  t: ""    n: <optimized out>
        ;
    *s = 0;    s: ""
    return os;
}









```

 Evaluate

Expression:

p|

Result:

```
▼  result = {struct proc * | 0x80010f60} 0x80010f60
  >  lock = {struct spinlock}
    01 state = {enum procstate} RUNNABLE
    01 chan = {void * | 0x0} NULL
    01 killed = {int} 0
    01 xstate = {int} 0
    01 pid = {int} 1
    01 parent = {struct proc * | 0x0} NULL
    01 kstack = {uint64} 274877894656
    01 sz = {uint64} 4096
  >  pagetable = {pagetable_t | 0x87f73000} 0x87f73000
  >  trapframe = {struct trapframe * | 0x87f74000} 0x87f74000
  >  context = {struct context}
  >  ofile = {struct file *[16]}
  >  cwd = {struct inode * | 0x8001f070} 0x8001f070
  >  name = {char [16]}
```

scheduler میره و پراسه که الان باید cpu اجرا کند را مشخص میکند

```
scheduler(void)
{
    struct proc *p;
    struct cpu *c = mycpu();

    c->proc = 0;
    for(;;){
        // Avoid deadlock by ensuring that devices can interrupt.
        intr_on();

        for(p = proc; p < &proc[NPROC]; p++) {
            acquire(&p->lock);
            if(p->state == RUNNABLE) {
                // Switch to chosen process. It is the process's job
                // to release its lock and then reacquire it
                // before jumping back to us.
                p->state = RUNNING;
                c->proc = p;
                swtch(&c->context, &p->context);

                // Process is done running for now.
                // It should have changed its p->state before coming back.
                c->proc = 0;
            }
            release(&p->lock);
        }
    }
}
```

برای درست کردن یک سیستم کال ابتدا باید فایل syscall.h را عوض کنیم به صورت زیر

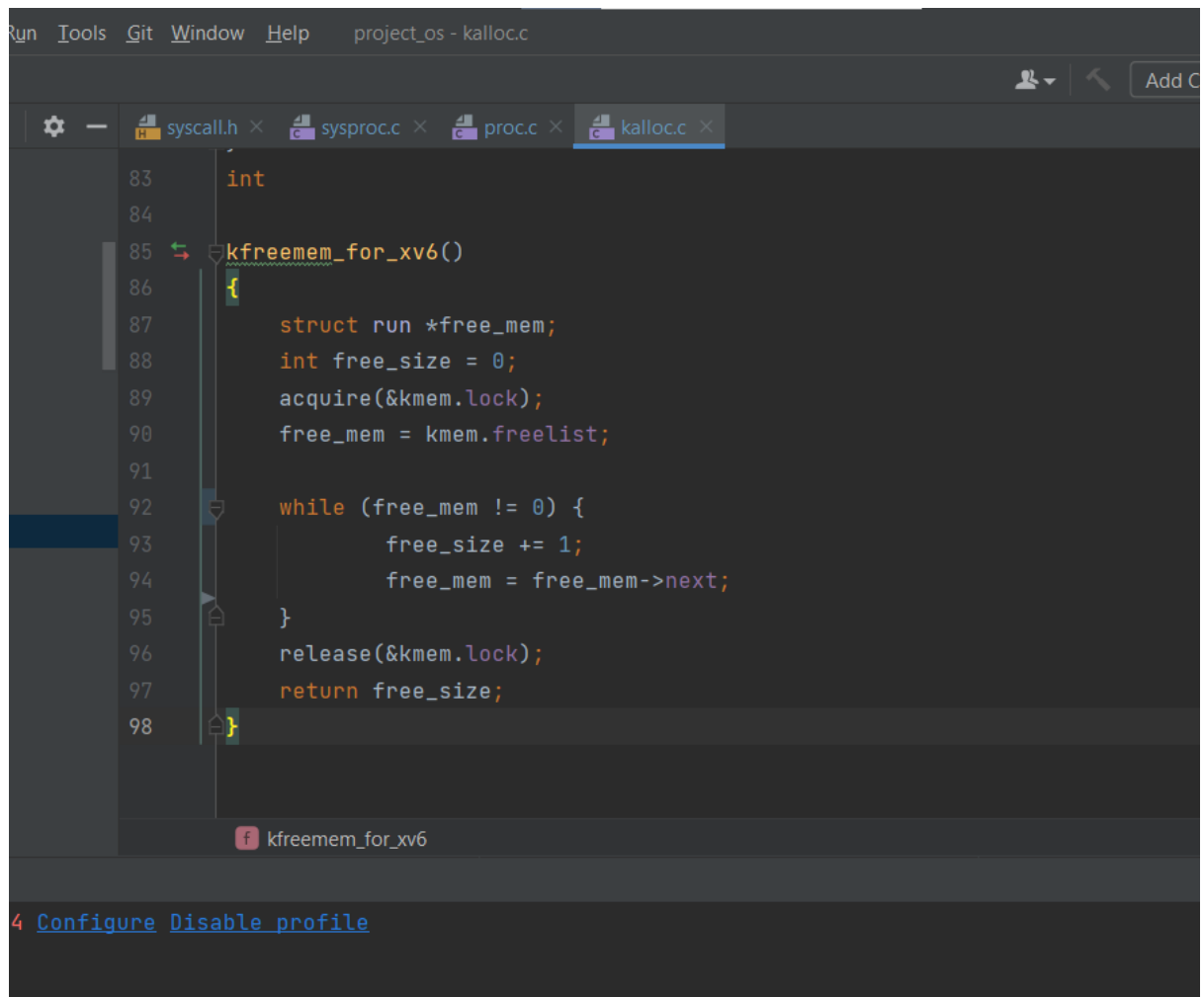
```
user.c x  syscall.h x  proc.c x  proc.h x  usys.d x  usys.pl x  kalloc.c x  user.h x
4  #define SYS_wait 3
5  #define SYS_pipe 4
6  #define SYS_read 5
7  #define SYS_kill 6
8  #define SYS_exec 7
9  #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define
12 #define
13 #define
14 #define
15 #define #define SYS_chdir 9
16 #define
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 #define SYS_used_mem 23
24 #define SYS_kfreemem_for_xv6 24
25
```

سپس در فایل sysproc.c متد هامون رو تعریف میکنیم

```
81 // since start.
82 uint64
83 sys_uptime(void)
84 {
85     uint xticks;
86
87     acquire(&tickslock);
88     xticks = ticks;
89     release(&tickslock);
90     return xticks;
91 }
92
93 uint64
94 sys_used_mem(void)
95 {
96     return used_mem();
97 }
98
99 uint64
100 sys_kfreemem_for_xv6(void)
101 {
102     return kfreemem_for_xv6();
103 }
```

در فایل `proc.c` متد مربوط با پراسه ها رو که میرود کل حافظه اشغال شده توسط پراسه رو جمع میکند مینویسیم و در فایل `kalloc.c` مریم و با استفاده از `kmem.freelist` مقدار فضای خالی سیستم را به دست می آوریم


```
syscall.h x sysproc.c x proc.c x
83     }
84     uint64
85     used_mem(void)
86     {
87         struct proc *p;
88         uint64 size_used_memory = 0 ;
89         uint64 size_proce = 0 ;
90         for(p = proc; p < &proc[NPROC]; p++){
91             size_proce=p->sz;
92             size_used_memory = size_used_memory + size_proce;
93         }
94         return size_used_memory;
95     }
```



The screenshot shows a code editor window titled "project_os - kalloc.c". The editor has tabs for "syscall.h", "sysproc.c", "proc.c", and "kalloc.c". The "kalloc.c" tab is active, showing the following code:

```
83     int
84
85     kfreemem_for_xv6()
86     {
87         struct run *free_mem;
88         int free_size = 0;
89         acquire(&kmem.lock);
90         free_mem = kmem.freelist;
91
92         while (free_mem != 0) {
93             free_size += 1;
94             free_mem = free_mem->next;
95         }
96         release(&kmem.lock);
97         return free_size;
98     }
```

Below the code editor, there is a status bar showing a red icon and the text "kfreemem_for_xv6". At the bottom of the window, there is a text area containing the text "4 [Configure](#) [Disable profile](#)".

حالا باید سیستم کال هامون رو در فایل `syscall.c` تعریف کنیم تا قابل استفاده باشند

```
usertest.c x syscall.h x sysproc.c x proc.c x proc.h x usys.d x usys.pl x kalloc.c x syscall.c
8      argaddr(n, &addr);
9      return fetchstr(addr, buf, max);
10  }
1
2      // Prototypes for the functions that handle system calls.
3      extern uint64 sys_fork(void);
4      extern uint64 sys_exit(void);
5      extern uint64 sys_wait(void);
6      extern uint64 sys_pipe(void);
7      extern uint64 sys_read(void);
8      extern uint64 sys_kill(void);
9      extern uint64 sys_exec(void);
10     extern uint64 sys_fstat(void);
11     extern uint64 sys_chdir(void);
12     extern uint64 sys_dup(void);
13     extern uint64 sys_getpid(void);
14     extern uint64 sys_sbrk(void);
15     extern uint64 sys_sleep(void);
16     extern uint64 sys_uptime(void);
17     extern uint64 sys_open(void);
18     extern uint64 sys_write(void);
19     extern uint64 sys_mknod(void);
20     extern uint64 sys_unlink(void);
21     extern uint64 sys_link(void);
22     extern uint64 sys_mkdir(void);
23     extern uint64 sys_close(void);
24     extern uint64 sys_used_mem(void);
25     extern uint64 sys_kfreemem_for_xv6(void);
```

```
usertest.c x syscall.h x sysproc.c x proc.c x proc.h x usys.d x usys.pl x kalloc.c x syscall.c x user.h x
88     // to the function that handles the system call.
89     static uint64 (*syscalls[])(void) = {
90         [SYS_fork] sys_fork,
91         [SYS_exit] sys_exit,
92         [SYS_wait] sys_wait,
93         [SYS_pipe] sys_pipe,
94         [SYS_read] sys_read,
95         [SYS_kill] sys_kill,
96         [SYS_exec] sys_exec,
97         [SYS_fstat] sys_fstat,
98         [SYS_chdir] sys_chdir,
99         [SYS_dup] sys_dup,
100        [SYS_getpid] sys_getpid,
101        [SYS_sbrk] sys_sbrk,
102        [SYS_sleep] sys_sleep,
103        [SYS_uptime] sys_uptime,
104        [SYS_open] sys_open,
105        [SYS_write] sys_write,
106        [SYS_mknod] sys_mknod,
107        [SYS_unlink] sys_unlink,
108        [SYS_link] sys_link,
109        [SYS_mkdir] sys_mkdir,
110        [SYS_close] sys_close,
111        [SYS_kfreemem_for_xv6] sys_kfreemem_for_xv6,
112        [SYS_used_mem] sys_used_mem,
113    };
114
115     void
116     syscalls(void)
```

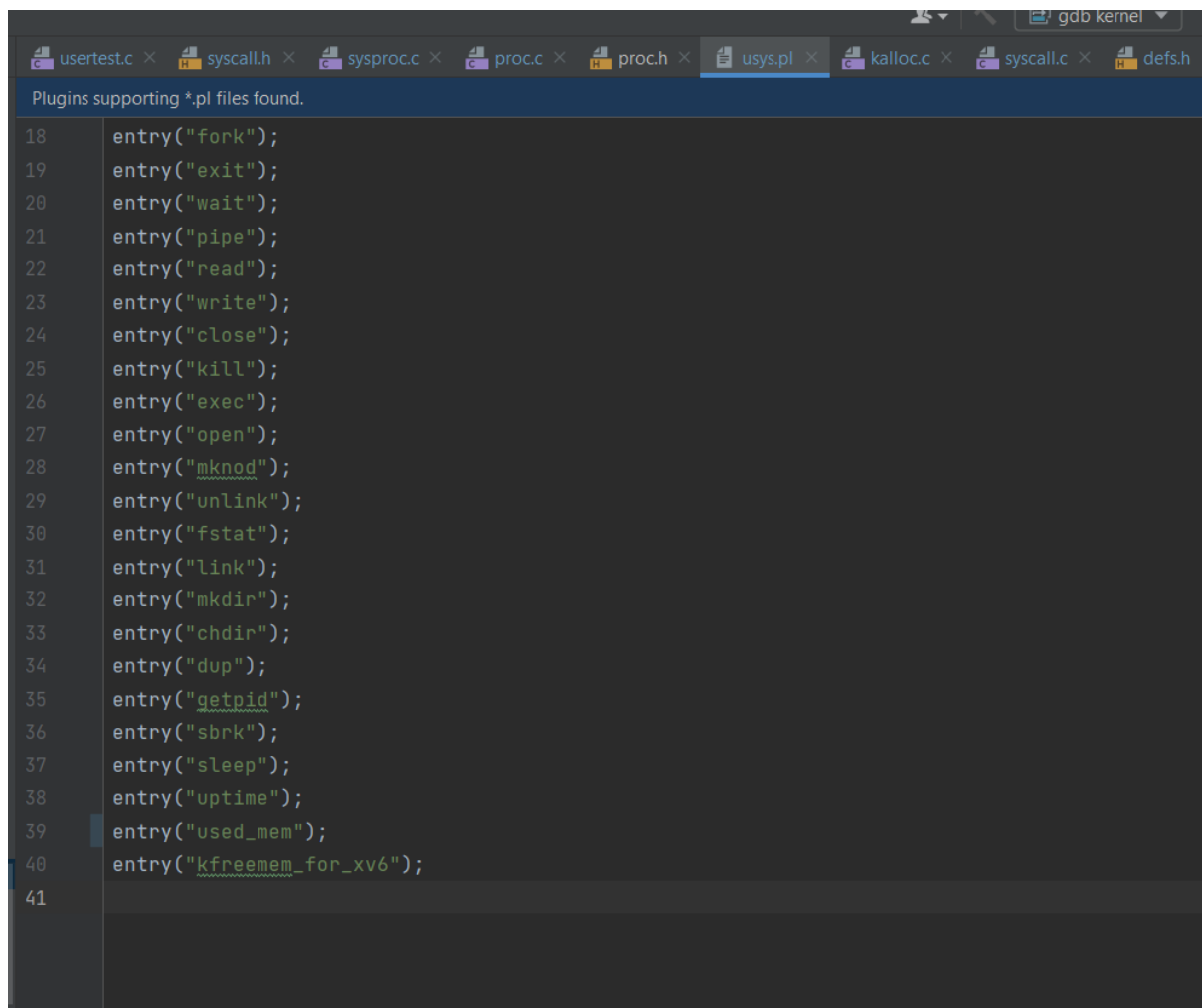
برای آنکه توابع `sys_kfreemem_for_xv6` , `sys_used_mem` بتوان استفاده کرد و بشه شناخت باید داخل فایل `defs.h` نیز اینها تعریف شود

```
userstest.c x syscall.h x sysproc.c x proc.c x proc.h x usys.pl x kalloc.c x syscall.c x
90 void proc_mapstacks(pagetable_t);
91 → pagetable_t proc_pagetable(struct proc *);
92 void proc_freepagetable(pagetable_t, uint64);
93 → int kill(int);
94 → int killed(struct proc*);
95 → void setkilled(struct proc*);
96 → struct cpu* mycpu(void);
97 → struct cpu* getmycpu(void);
98 → struct proc* myproc();
99 → void procinit(void);
100 → void scheduler(void) __attribute__((noreturn));
101 → void sched(void);
102 → void sleep(void*, struct spinlock*);
103 → void userinit(void);
104 int wait(uint64);
105 → void wakeup(void*);
106 → void yield(void);
107 → int either_copyout(int user_dst, uint64 dst, void *src, uint64 len);
108 → int either_copyin(void *dst, int user_src, uint64 src, uint64 len);
109 → void procdump(void);
110 → uint64 used_mem(void);
111
112 // swtch.S
113 void swtch(struct context*, struct context*);
114
115 // spinlock.c
116 void acquire(struct spinlock*);
117 int holding(struct spinlock*);
118 void initlock(struct spinlock*, char*);
f kfreemem_for_xv6
```

```
userstest.c x syscall.h x sysproc.c x proc.c x proc.h x usys.pl x kalloc.c x syscall.c x defs.h
51 struct inode* nameiparent(char*, char*);
52 int readi(struct inode*, int, uint64, uint, uint);
53 void stati(struct inode*, struct stat*);
54 int writei(struct inode*, int, uint64, uint, uint);
55 void itrunc(struct inode*);
56
57 // ramdisk.c
58 void ramdiskinit(void);
59 void ramdiskintr(void);
60 void ramdiskrw(struct buf*);
61
62 // kalloc.c
63 void* kalloc(void);
64 void kfree(void *);
65 void kinit(void);
66 int kfreemem_for_xv6(void);
67
68 // log.c
69 void initlog(int, struct superblock*);
70 void log_write(struct buf*);
71 void begin_op(void);
72 void end_op(void);
73
74 // pipe.c
75 int pipealloc(struct file**, struct file**);
76 void pipeclose(struct pipe*, int);
77 int piperead(struct pipe*, uint64, int);
78 int pipewrite(struct pipe*, uint64, int);
79
```

برای اینکه بتوان از این سیستم کال ها در پکیج یوزر نیز استفاده کرد باید در دو فایل `usys.pl` و فایل `user.h` نیز اینها رو اد کنیم

```
usertest.c x syscall.h x sysproc.c x proc.c x proc.h x usys.pl x kalloc.c x syscall.c x defs.h x user.h x
int close(int);
int kill(int);
int exec(const char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
uint64 FreeMemory(void);
int kfreemem_for_xv6(void);
int used_mem(void);
// ulib.c
int stat(const char*, struct stat*);
char* strcpy(char*, const char*);
void *memmove(void*, const void*, int);
char* strchr(const char*, char c);
int strcmp(const char*, const char*);
void fprintf(int, const char*, ...);
void printf(const char*, ...);
char* gets(char*, int max);
uint strlen(const char*);
void* memset(void*, int, uint);
```



The screenshot shows a GDB kernel window with a list of system call entries. The window title is "gdb kernel". The tabs at the top include "usertest.c", "syscall.h", "sysproc.c", "proc.c", "proc.h", "usys.pl", "kalloc.c", "syscall.c", and "defs.h". The "usys.pl" tab is active, displaying a list of system call entries. The text "Plugins supporting *.pl files found." is visible at the top of the window. The list of entries is as follows:

```
18 entry("fork");
19 entry("exit");
20 entry("wait");
21 entry("pipe");
22 entry("read");
23 entry("write");
24 entry("close");
25 entry("kill");
26 entry("exec");
27 entry("open");
28 entry("mknod");
29 entry("unlink");
30 entry("fstat");
31 entry("link");
32 entry("mkdir");
33 entry("chdir");
34 entry("dup");
35 entry("getpid");
36 entry("sbrk");
37 entry("sleep");
38 entry("uptime");
39 entry("used_mem");
40 entry("kfreemem_for_xv6");
41
```

برای تست کردن پروژه در دوتا سیستم کال را صدا میزنیم و دیگری یک فایل
usertest.c در پکیج یوزر میسازیم تا تست کنیم

```
syscall.h x sysproc.c x proc.c x kalloc.c x syscall.c x defs.h x user.h x usys.pl x main.c x
if(cpuid() == 0){
    consoleinit();
    printfinit();
    printf("\n");
    printf("xv6 kernel is booting\n");
    printf("\n");
    kinit();          // physical page allocator
    kvminit();        // create kernel page table
    kvmminithart();   // turn on paging
    procinit();       // process table
    trapinit();       // trap vectors
    trapminithart();  // install kernel trap vector
    plicinit();       // set up interrupt controller
    plicminithart();  // ask PLIC for device interrupts
    binit();          // buffer cache
    iinit();          // inode table
    fileinit();       // file table
    virtio_disk_init(); // emulated hard disk
    printf("free memory ,find with kmem freelist is : %d\n", kfreemem_for_xv6());
    userinit();       // first user process
    printf("free memory ,find with proc is : %d\n", used_mem());
    __sync_synchronize();
    started = 1;
} else {
    while(started == 0)
        ;
    __sync_synchronize();
    printf("hart %d starting\n", cpuid());
}
```

```
syscall.h x sysproc.c x proc.c x kalloc.c x syscall.c x defs.h x user.h x usys.pl x main.c x usertest.c x
#include "kernel/types.h"
#include "kernel/stat.h"
#include "user/user.h"

void
main(void){
    int used_memory = used_mem();
    int free_memory = kfreemem_for_xv6();
    int system_memory = used_memory+free_memory;
    //print
    printf("free memory of xv6: %d\n", free_memory);
    printf("used memory of xv6: %d\n", used_memory);
    printf("system memory of xv6: %d\n", system_memory);
}
```



```

xv6@c5d2b94cc38c: ~/xv6-riscv
riscv64-unknown-elf-objdump -t user/_grind | sed '1,/SYMBOL TABLE/d; s/ .* / /; /\$/d' > user/grind.sym
riscv64-unknown-elf-gcc -wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/wc.o user/wc.c
riscv64-unknown-elf-lld -Z max-page-size=4096 -T user/user.ld -o user/_wc user/wc.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-unknown-elf-objdump -S user/_wc > user/wc.asm
riscv64-unknown-elf-objdump -t user/_wc | sed '1,/SYMBOL TABLE/d; s/ .* / /; /\$/d' > user/wc.sym
riscv64-unknown-elf-gcc -wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/zombie.o user/zombie.c
riscv64-unknown-elf-lld -Z max-page-size=4096 -T user/user.ld -o user/_zombie user/zombie.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-unknown-elf-objdump -S user/_zombie > user/zombie.asm
riscv64-unknown-elf-objdump -t user/_zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /\$/d' > user/zombie.sym
riscv64-unknown-elf-gcc -wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/usertest.o user/usertest.c
riscv64-unknown-elf-lld -Z max-page-size=4096 -T user/user.ld -o user/_usertest user/usertest.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-unknown-elf-objdump -S user/_usertest > user/usertest.asm
riscv64-unknown-elf-objdump -t user/_usertest | sed '1,/SYMBOL TABLE/d; s/ .* / /; /\$/d' > user/usertest.sym
mkfs/mkfs fs.img README user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user/_usertests user/_grind user/_wc user/_zombie user/_usertest
mmeta 46 (boot, super, log blocks 30 inode blocks 13, Bitmap blocks 1) blocks 1954 total 2000
ballocc: first 777 blocks have been allocated
ballocc: write bitmap block at sector 45
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

free memory ,find with kmem freelist is : 32595
free memory ,find with proc is : 4096
hart 2 starting
hart 1 starting
init: starting sh
$ usertest
exec usertest failed
$ usertest
free memory of xv6: 32564
used memory of xv6: 53248
system memory of xv6: 85812
$ -

```

```

xv6@c5d2b94cc38c: ~/xv6-riscv
riscv64-unknown-elf-objdump -t user/_usertests | sed '1,/SYMBOL TABLE/d; s/ .* / /; /\$/d' > user/usertests.sym
riscv64-unknown-elf-gcc -wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/grind.o user/grind.c
riscv64-unknown-elf-lld -Z max-page-size=4096 -T user/user.ld -o user/_grind user/grind.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-unknown-elf-objdump -S user/_grind > user/grind.asm
riscv64-unknown-elf-objdump -t user/_grind | sed '1,/SYMBOL TABLE/d; s/ .* / /; /\$/d' > user/grind.sym
riscv64-unknown-elf-gcc -wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/wc.o user/wc.c
riscv64-unknown-elf-lld -Z max-page-size=4096 -T user/user.ld -o user/_wc user/wc.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-unknown-elf-objdump -S user/_wc > user/wc.asm
riscv64-unknown-elf-objdump -t user/_wc | sed '1,/SYMBOL TABLE/d; s/ .* / /; /\$/d' > user/wc.sym
riscv64-unknown-elf-gcc -wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/zombie.o user/zombie.c
riscv64-unknown-elf-lld -Z max-page-size=4096 -T user/user.ld -o user/_zombie user/zombie.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-unknown-elf-objdump -S user/_zombie > user/zombie.asm
riscv64-unknown-elf-objdump -t user/_zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /\$/d' > user/zombie.sym
riscv64-unknown-elf-gcc -wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/usertest.o user/usertest.c
riscv64-unknown-elf-lld -Z max-page-size=4096 -T user/user.ld -o user/_usertest user/usertest.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-unknown-elf-objdump -S user/_usertest > user/usertest.asm
riscv64-unknown-elf-objdump -t user/_usertest | sed '1,/SYMBOL TABLE/d; s/ .* / /; /\$/d' > user/usertest.sym
mkfs/mkfs fs.img README user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user/_usertests user/_grind user/_wc user/_zombie user/_usertest
mmeta 46 (boot, super, log blocks 30 inode blocks 13, Bitmap blocks 1) blocks 1954 total 2000
ballocc: first 777 blocks have been allocated
ballocc: write bitmap block at sector 45
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

free memory ,find with kmem freelist is : 32595
free memory ,find with proc is : 4096
hart 2 starting
hart 1 starting
init: starting sh
$

```