بسمه تعالی

گزارش کار پروژه پنجم آزمایشگاه سیستم عامل

امیرحسین راحتی : ۸۱۰۱۰۰۱۴۴

سید احمد رکنی حسینی : ۸۱۰۱۰۰۱۵۴

محمدعلی شاهین فر : ۸۱۰۱۰۰۱۶۹

سوال ۱

راجع به مفهوم ناحیه مجازی در لینوکس به طور مختصر توضیح داده و آن را با xv6 مقایسه کنید

.

مفهوم حافظه مجازی به تمام حافظه ای که در جایگاه مموری استفاده می شوند اطلاق می شود.
می تواند این حافظه تنها شامل ram باشد یا می تواند شامل بخشی از حافظه دیسک (SSD, HDD) یا هچیز دیگری باشد.
در کنار حافظه فیزیکی همواره یک واحد memory management unit (MMU) نیز وجود دارد که آدرس مجازی را به آدرس فیزیکی تبدیل می کند تا بتوان به حافظه فیزیکی دسترسی داشت.

تفاوت vma در لینوکس و xv6 :
- از آنجا که xv6 یک سیستم ساده است و با هدف آموزش ساخته شده ، به همین دلیل در آن از روش های ساده برای مدیریت مموری و paging استفاده شده
از طرف دیگر فیچرهایی مانند memory-mapped segment و shared memory هم در آن پیاده سازی نشده
اما در لینوکس که یک سیستم عامل general purpos است هدف بهره وری بیشتر بوده نه سادگی ، به همین دلیل در آن فیچرهای زیادی از جمله فیچرهایی که اشاره شد موجود است و مموری و paging نیز بهتر و پیچیده تر مدیریت می شوند

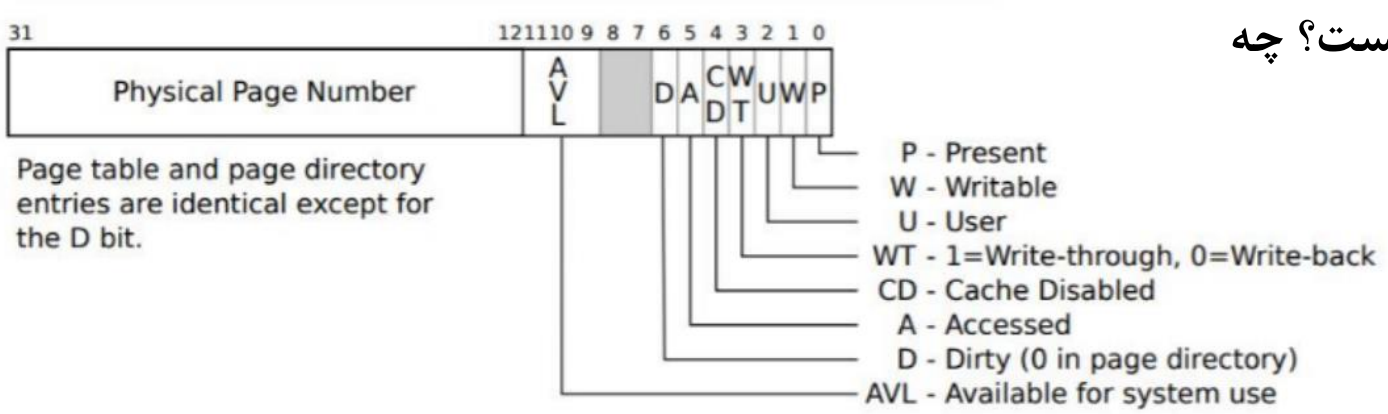
سوال ۲

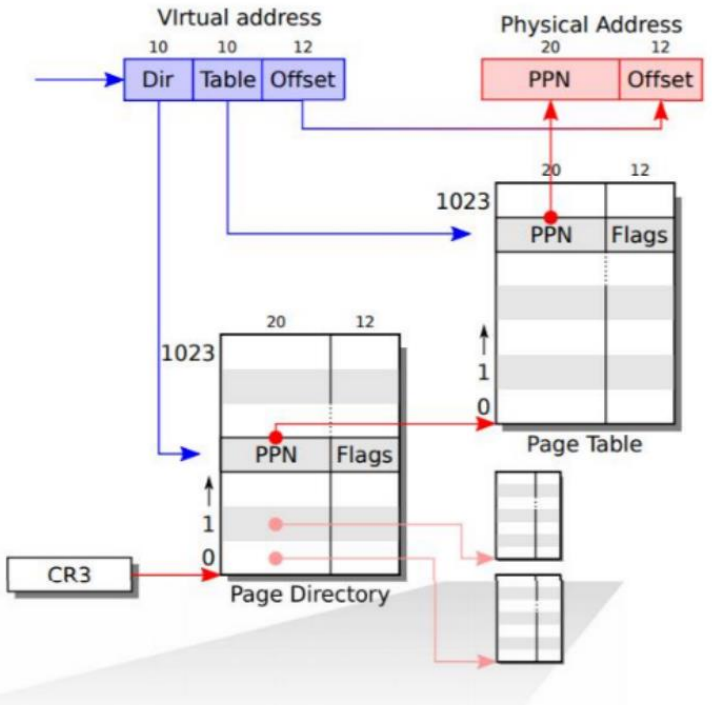
چرا ساختار سلسله مراتبی منجر به کاهش مصرف حافظه میگردد؟

در این ساختار با توجه به اینکه process ها می توانند کدها و داده ها را بین همدیگر توسط mapping به اشتراک بگذارند می توان از مصرف بیشتر مموری جلوگیری کرد

سوال ۳

محتوای هر بیت یک مدخل (۳۲ بیتی) در هر سطح چیست؟ چه تفاوتی میان آنها وجود دارد؟





همانطور که در شکل رو به رو هم مشخص در مدخل سطح page directory ۲۰ بیت برای اشاره به آدرس لایه بعدی استفاده می شود و ۱۲ بیت برای سطوح دسترسی
در page table نیز ۲۰ بیت برای اشاره به آدرس فیزیکی و ۱۲ بیت هم برای سطوح دسترسی نگه داشته می شود.

تنها تفاوت آنها در بیت Dirrty (D) است (در شکل رو به رو هم اشاره شده است)
در page table این بیت معنایی ندارد
اما در page directory به این معنی است که صفحه برای اعمال تغییرات باید در دیسک نوشته شود

سوال ۴

تابع kalloc چه نوع حافظه ای تخصیص میدهد؟ (فیزیکی یا مجازی)

```
// Allocate one 4096-byte page of physical memory.
// Returns a pointer that the kernel can use.
// Returns 0 if the memory cannot be allocated.
char*
kalloc(void)
{
  struct run *r;

  if(kmem.use_lock)
    acquire(&kmem.lock);
  r = kmem.freelist;
  if(r)
    kmem.freelist = r->next;
  if(kmem.use_lock)
    release(&kmem.lock);
  return (char*)r;
}
```

به اندازه ۴۰۹۶ بایت حافظه فیزیکی می گیرد
در خود فایل kalloc.c هم این مورد ذکر شده است.

سوال ۵

تابع mappages چه کاربردی دارد؟

```
// Create PTEs for virtual addresses starting at va that refer to
// physical addresses starting at pa. va and size might not
// be page-aligned.
int
mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm)      Robert
{
  char *a, *last;
  pte_t *pte;

  a = (char*)PGROUNDDOWN((uint)va);
  last = (char*)PGROUNDDOWN(((uint)va) + size - 1);
  for(;;){
    if((pte = walkpgdir(pgdir, a, 1)) == 0)
      return -1;
    if(*pte & PTE_P)
      panic("remap");
    *pte = pa | perm | PTE_P;
    if(a == last)
      break;
    a += PGSIZE;
    pa += PGSIZE;
  }
  return 0;
}
```

در کد خود xv6 توضیحات لازم آمده است :
این تابع آدرس virtual را به آدرس فیزیکی تبدیل می کند (در فضای آدرس پروسس)
از طرفی صفحه جدید را نیز به pgdir اضافه می کند

سوال ۷

راجع به تابع walkpgdir() توضیح دهید. این تابع چه عمل سخت افزاری را شبیه سازی میکند؟

```
// Return the address of the PTE in page table pgdir
// that corresponds to virtual address va.  If alloc!=0,
// create any required page table pages.
 pte_t *
walkpgdir(pde_t *pgdir, const void *va, int alloc)
{
  pde_t *pde;
  pte_t *pgtab;

  pde = &pgdir[PDX(va)];
  if(*pde & PTE_P){
    pgtab = (pte_t*)P2V(PTE_ADDR(*pde));
  } else {
    if(!alloc || (pgtab = (pte_t*)kalloc()) == 0)
      return 0;
    // Make sure all those PTE_P bits are zero.
    memset(pgtab, 0, PGSIZE);
    // The permissions here are overly generous, but they can
    // be further restricted by the permissions in the page table
    // entries, if necessary.
    *pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
  }
  return &pgtab[PTX(va)];
}
```

باز هم به کد xv6 مراجعه می کنیم
همانطور که توضیح داده شده این تابع آدرس PTE در pgdir مشخص شده را برمی گرداند و هر page table ای که لازم داشته باشد را می سازد
این کار معادل عملیات سخت افزاری ترجمه آدرس مجازی به فیزیکی است

سوال ۸

توابع allocuvm و mappages که در ارتباط با حافظه ی مجازی هستند را توضیح دهید.

```
// Allocate page tables and physical memory to grow process from oldsz to
// newsz, which need not be page aligned.  Returns new size or 0 on error.
int
allocuvm(pde_t *pgdir, uint oldsz, uint newsz)      Austin Clements, 13 years ag
{
  char *mem;
  uint a;
```

Allocuvm : این تابع ازبخش حافظه فیزیکی user برای یک پروسس ، حافظه می گیرد. سه پارامتر pgdir که page directory آن و اندازه قبلی و جدید آن را می گیرد و حافظه پروسس را از اندازه قبلی به اندازه جدید افزایش می دهد و مقادیر بخش جدید را صفر می کند.

Mappages : در سوال ۵ نیز توضیح داده شد ، کاربرد آن نگاشت کردن آدرس مجازی به آدرس فیزیکی است

این دو تابع با یکدیگر سبب می شوند که ممووری در لایه پروسس به خوبی مدیریت شود ، allocuvm حافظه را به خوبی اختصاص می دهد (پیج هایی در یوزر اسپیس) و mappages آن ها را نگاشت می کند

سوال ۹

شیوه ی بارگذاری برنامه در حافظه توسط فراخوانی سیستمی exec را شرح دهید.

```
// Allocate two pages at the next page boundary.
// Make the first inaccessible.  Use the second as the user stack.
sz = PGROUNDUP(sz);
if((sz = allocuvm(pgdir, sz, sz + 2*PGSIZE)) == 0)
  goto bad;
clearpteu(pgdir, (char*)(sz - 2*PGSIZE));
sp = sz;

// Push argument strings, prepare rest of stack in ustack.
for(argc = 0; argv[argc]; argc++) {
  if(argc >= MAXARG)
    goto bad;
  sp = (sp - (strlen(argv[argc]) + 1)) & ~3;
  if(copyout(pgdir, sp, argv[argc], strlen(argv[argc]) + 1) < 0)
    goto bad;
  ustack[3+argc] = sp;
}
ustack[3+argc] = 0;

ustack[0] = 0xffffffff;  // fake return PC
ustack[1] = argc;
ustack[2] = sp - (argc+1)*4;  // argv pointer

sp -= (3+argc+1) * 4;
if(copyout(pgdir, sp, ustack, (3+argc+1)*4) < 0)
  goto bad;
```

برای این کار پس از اینکه برنامه را برای لود شدن در حافظه و اجرا آماده کرد چند کار انجام می دهد :

ابتدا آدرس فعلی حافظه را آزاد می کند و سپس یک آدرسی از حافظه را allocate می کند تا در آن code, data, stack و برنامه جدید را بنویسد،

برای این کار لازم است حافظه را چند قسمت کند که هر کدام در بخش خود قرار بگیرند.

در مرحله بعدی فایل برنامه جدید را می خواند code, data برنامه جدید را در حافظه لود می کند.

در مرحله بعد استک را می سازد و stack pointer و command-line arguments و متغیرهای محیطی را می خواند و در آن قرار می دهد

در مرحله آخر هم تمام متغیرهای مرتبط با پروسس جدید مانند pc , stack pointer برای برنامه جدید تنظیم می شوند و برای پیج های تخصیص داده شده page table را می سازد و به page directory اضافه می کند

پروژه

فایل proc.c و پیاده سازی های انجام شده برای shared memory

sharedmem.c M | proc.c M | defs.h M | sysproc.c M

proc.c > edit_sharedmem(int, int)

```
627   {
628     if (sharedMems[id].linkedPids[i] == proc->pid)
629       sharedMems[id].linkedPids[i] = -1;
630   }
631   if (!sharedMems[id].linkedProcs)
632   {
633     kfree(sharedMems[id].addr);
634   }
635   mem_release(&sharedMems[id].lock);
636   return 1;
637 }
638
639 int
640 edit_sharedmem(int id , int number)
641 {
642   mem_acquire(&sharedMems[id].lock);
643   for(int i = 0 ; i < MAX_PROCS ; i++)
644   {
645     if(myproc()->pid == sharedMems[id].linkedPids[i])
646     {
647       sharedMems[id].addr[0] += number;
648       break;
649     }
650   }
651   mem_release(&sharedMems[id].lock);
652   return sharedMems[id].addr[0];
653 }
654
```

sharedmem.c M | proc.c M | sysproc.c | proc.h M

proc.c > open_sharedmem(int)

```
548 }
549
550 void init_sharedmem(void)
551 {
552
553   for (int i = 0 ; i < MAX_SHARED_MEM ; i++)
554   {
555     sharedMems[i].linkedProcs = 0;
556     for(int j = 0 ; j < MAX_PROCS ; j++)
557     {
558       sharedMems[i].linkedPids[j] = -1;
559       sharedMems[i].lock = 0;
560     }
561   }
562   flag = 1;
563 }
564
565 void mem_acquire(int * lock)
566 {
567   while(*lock);;
568   *lock = 1;
569 }
570
571 void mem_release(int * lock)
572 {
573   *lock = 0;
574 }
575
576 char *
```

nal Help

sharedmem.c M | proc.c M | proc.h M

proc.c > ...

```
1  #include "types.h"
2  #include "defs.h"
3  #include "param.h"
4  #include "memlayout.h"
5  #include "mmu.h"
6  #include "x86.h"
7  #include "proc.h"
8  #include "spinlock.h"
9
10 #define MAX_PROCS 10
11 #define MAX_SHARED_MEM 10
12
13 struct SharedMemory
14 {
15   char* addr;
16   int linkedProcs;
17   int lock;
18   int attached_processes[MAX_PROCS];
19 };
20
21 struct SharedMemory sharedMems[MAX_SHARED_MEM];
22 int flag = 0;
23
24 struct {
25   struct spinlock lock;
26   struct proc proc[NPROC];
27 } ptable;
28
29 static struct proc *initproc;
```

0

پروژه

فایل proc.c و پیاده سازی های توابع
open_sharedmem , close_sharedmem

sharedmem.c M | proc.c M | defs.h M | sysproc.c M | user.h M | usys.S | proc.h M

proc.c > open_sharedmem(int)

```
577  open_sharedmem(int id)
578  {
579    if(!flag)
580      init_sharedmem();
581    // cprintf("ref : %d\n",sharedMems[id].linkedProcs);
582    struct proc* proc = myproc();
583    if (sharedMems[id].linkedProcs == 0)
584    {
585      char* mem = (char *)kalloc();
586      memset(mem, 0, PGSIZE);
587      mem_acquire(&sharedMems[id].lock);
588      myproc()->sz+= PGSIZE;
589      if(mappages(proc->pgdir, (char *)myproc()->sz, PGSIZE, V2P(mem), PTE_W|PTE_U))
590        return 0;
591      sharedMems[id].addr = mem;
592      sharedMems[id].linkedProcs +=1;
593      mem_release(&sharedMems[id].lock);
594    }
595    else
596    {
597      mem_acquire(&sharedMems[id].lock);
598      myproc()->sz+=PGSIZE;
599      if(mappages(proc->pgdir, (char *)myproc()->sz, PGSIZE, V2P(sharedMems[id].addr), PTE_W|PTE_U))
600        return 0;
601      sharedMems[id].linkedProcs++;
602      mem_release(&sharedMems[id].lock);
603    }
604      mem_acquire(&sharedMems[id].lock);
605    for (int i = 0; i < MAX_PROCS; i++)
```

sharedmem.c M | proc.c M | sysproc.c | proc.h M

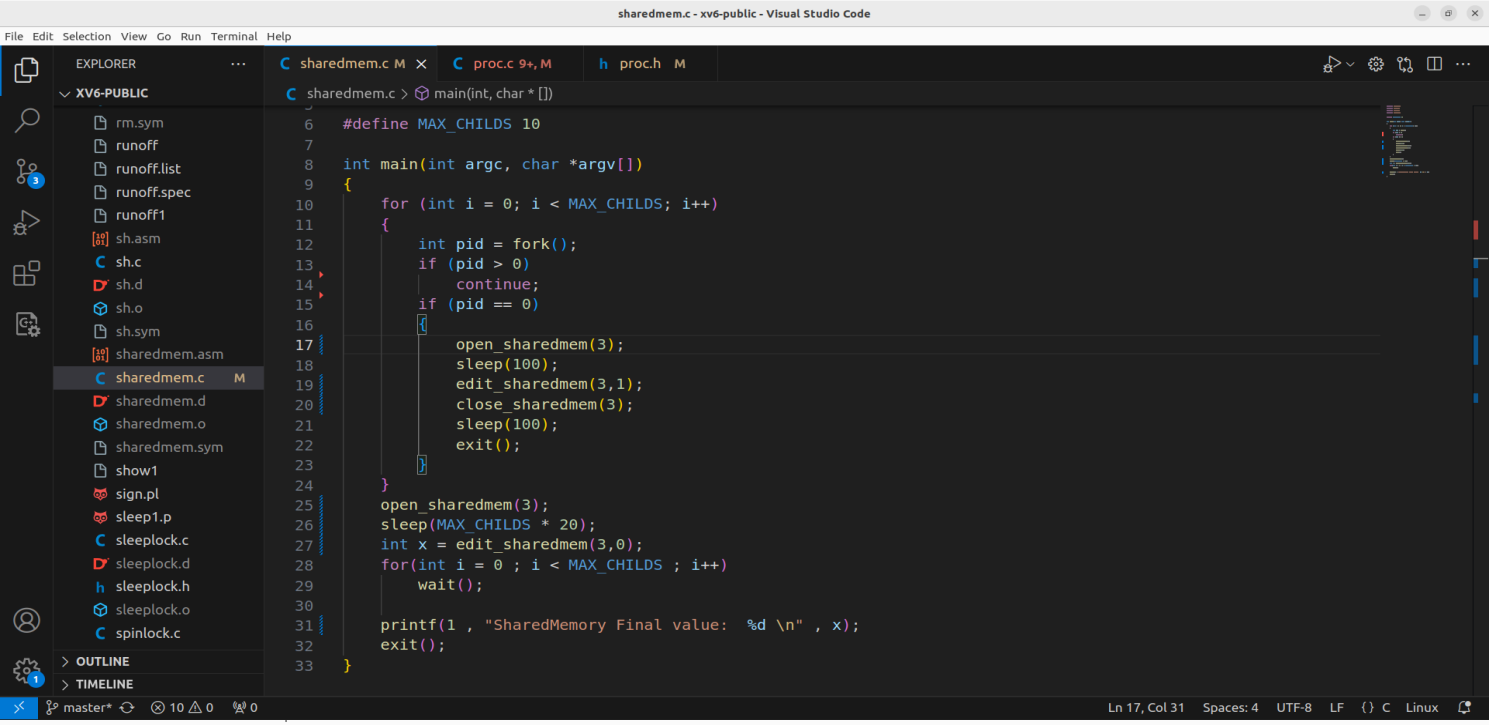
proc.c > edit_sharedmem(int, int)

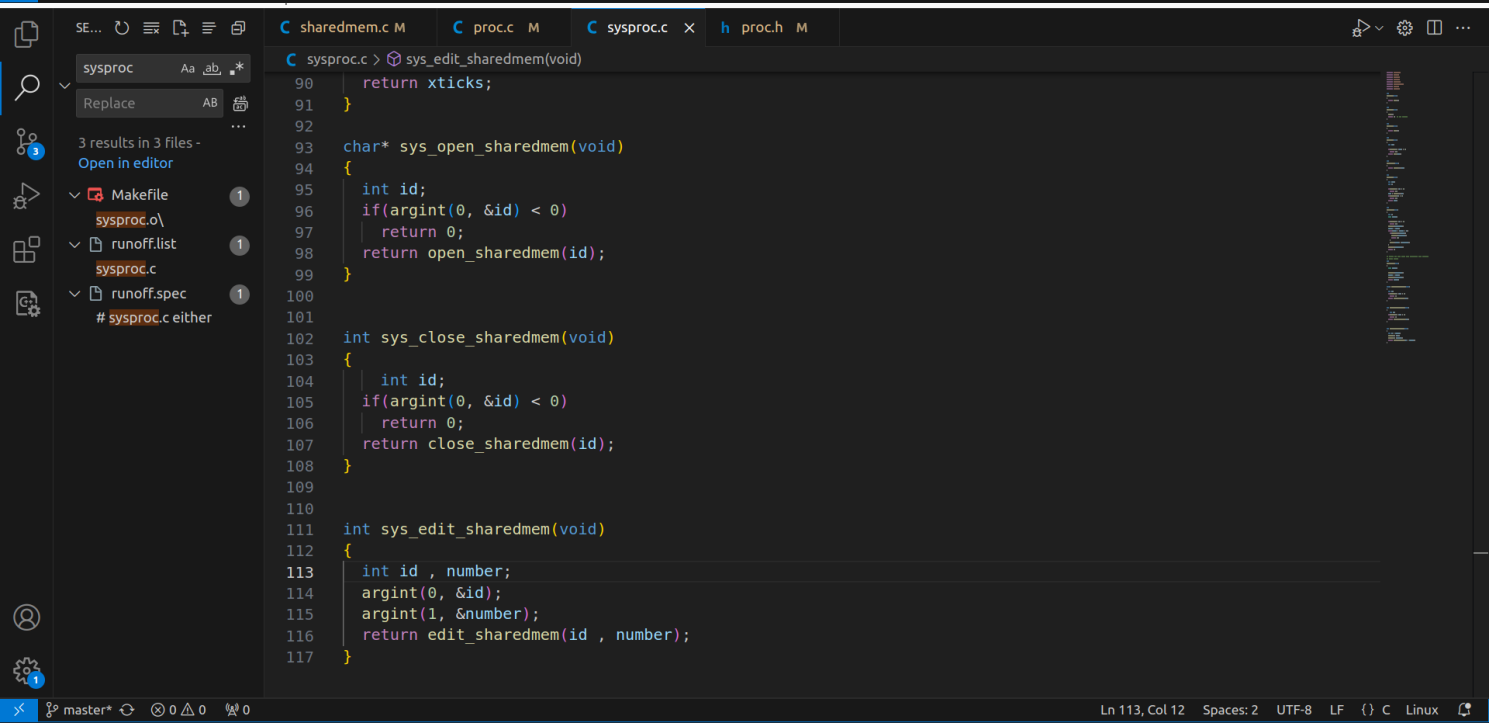
```
615    int
616    close_sharedmem(int id)
617    {
618      struct proc* proc = myproc();
619      mem_acquire(&sharedMems[id].lock);
620      sharedMems[id].linkedProcs--;
621          myproc()->sz-=PGSIZE;
622
623      for (int i = 0; i < MAX_PROCS; i++)
624      {
625        if (sharedMems[id].linkedPids[i] == proc->pid)
626          sharedMems[id].linkedPids[i] = -1;
627      }
628      if (!sharedMems[id].linkedProcs)
629      {
630          kfree(sharedMems[id].addr);
631      }
632      mem_release(&sharedMems[id].lock);
633      return 1;
634    }
635
```

sharedmem.c M | proc.c M | defs.h M | sysproc.c M | user.h M | usys.S | proc.h M

proc.c > open_sharedmem(int)

```
589      if(mappages(proc->pgdir, (char *)myproc()->sz, PGSIZE, V2P(mem), PTE_W|PTE_U))
590        return 0;
591      sharedMems[id].addr = mem;
592      sharedMems[id].linkedProcs +=1;
593      mem_release(&sharedMems[id].lock);
594    }
595    else
596    {
597      mem_acquire(&sharedMems[id].lock);
598      myproc()->sz+=PGSIZE;
599      if(mappages(proc->pgdir, (char *)myproc()->sz, PGSIZE, V2P(sharedMems[id].addr), PTE_W|PTE_U))
600        return 0;
601      sharedMems[id].linkedProcs++;
602      mem_release(&sharedMems[id].lock);
603    }
604      mem_acquire(&sharedMems[id].lock);
605    for (int i = 0; i < MAX_PROCS; i++)
606    {
607      if (sharedMems[id].linkedPids[i] == -1)
608      {
609        sharedMems[id].linkedPids[i] = proc->pid;
610        break;
611      }
612    }
613      mem_release(&sharedMems[id].lock);
614      return 1;
615  }
616
```





پروژه

برنامه سطح یوزر sharedmem و تغییرات فایل sys_proc برای سیستم کال های اضافه شده

شناسه آخرین کامیت :

https://github.com/AmirhosseinRHT/os-lab5/commit/a44fc65f47e883e4fd095198e7fd443c9590f57d