گزارش کار پروژه پنجم آزمایشگاه سیستم عامل

گروه ۱۳:

سیده زینب پیش بین - ۸۱۰۱۹۹۵۹۷

محمد جواد بشارتی - ۸۱۰۱۹۹۳۸۶

آدرس مخزن github:

https://github.com/JavadBesharati/OS-lab-projects-UT-Spring-2023

شناسه آخرین commit:

2a27fd39b0343e1b14f672a5caa4f66892842690

برای تغییر ساختار حافظه و رشد فضای حافظه پشته تغییراتی در فایل‌های زیر اعمال شدند:

defs.h:

```
183   pde_t*          copyuvm(pde_t*, uint, uint);
```

exec.c:

```
22    uint stack_top;
```

```
67    stack_top = KERNBASE - 2 * PGSIZE;
68    if((sp = allocuvm(pgdir, stack_top, KERNBASE)) == 0)
69      goto bad;
70
```

```
100   curproc->stack_top = stack_top;
```

proc.c:

```
192   // Copy process state from proc.
193   if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz, curproc->stack_top)) == 0){
```

```
200   np->stack_top = curproc->stack_top;
```

proc.h:

```
38    struct proc {
39      uint stack_top;
```

syscall.c:

```
17    int
18    fetchint(uint addr, int *ip)
19    {
20      if(addr >= KERNBASE || addr+4 > KERNBASE)
21        return -1;
22      *ip = *(int*)(addr);
23      return 0;
24    }
```

```
29  int
30  fetchstr(uint addr, char **pp)
31  {
32    char *s, *ep;
33
34    if(addr >= KERNBASE)
35      return -1;
36    *pp = (char*)addr;
37    ep = (char*)KERNBASE;
38
39    for(s = *pp; s < ep; s++){
40      if(*s == 0)
41        return s - *pp;
42    }
43    return -1;
44  }
```

```
56  int
57  argptr(int n, char **pp, int size)
58  {
59    int i;
60
61    if(argint(n, &i) < 0)
62      return -1;
63    if(size < 0 || (uint)i >= KERNBASE || (uint)i+size > KERNBASE)
64      return -1;
65    *pp = (char*)i;
66    return 0;
67  }
```

.trap.c

```
17  uint last;
18  uint bot;
19  uint stackTop;
```

```
85    case T_PGFLT:
86      last = rcr2();
87      stackTop = myproc()->stack_top;
88
89      if(last < stackTop && last > (stackTop - PGSIZE))
90      {
91        bot = myproc()->stack_top - PGSIZE;
92        if(allocuvm(myproc()->pgdir, bot, myproc()->stack_top) == 0)
93        {
94          cprintf("Page fault detected... pid %d %s: trap %d err %d on cpu %d "
95                  "eip 0x%x add 0x%x--kill proc,\n top_stack 0x%x\n",
96                  myproc()->pid, myproc()->name, tf->trapno,
97                  tf->err, cpuid(), tf->eip, rcr2(), myproc()->stack_top);
98              myproc()->killed = 1;
99              break;
100       }
101       else
102       {
103         myproc()->stack_top = bot;
104         break;
105       }
106     }
```

:vm.c

```
227     if(newsz > KERNBASE)
```

```
316   copyuvm(pde_t *pgdir, uint sz, uint stack_top)
```

```
341     if(stack_top == 0)
342     {
343       return d;
344     }
345
346     for(i = stack_top; i < KERNBASE; i += PGSIZE)
347     {
348       if((pte = walkpgdir(pgdir, (void*)i, 0)) == 0)
349       {
350         panic("copyuvm: pte should exist 2");
351       }
352       if(!(*pte & PTE_P))
353       {
354         panic("copyuvm: page not present 2");
355       }
356       pa = PTE_ADDR(*pte);
357       flags = PTE_FLAGS(*pte);
358       if((mem = kalloc()) == 0)
359       {
360         goto bad;
361       }
362       memmove(mem, (char*)P2V(pa), PGSIZE);
363       if(mappages(d, (void*)i, PGSIZE, V2P(mem), flags) < 0)
364       {
365         goto bad;
366       }
367     }
```

همچنین فایل test_stack.c برای تست کارکرد برنامه اضافه شد:

test_stack.c > ...

```c
#include "types.h"
#include "stat.h"
#include "user.h"



int test(int n)
{
   //if(n<100)
   //if(n<100)
    test(n + 1);
    return n;
}

int main(int argc, char *argv[])
{
    int pid=0;
    pid = fork();
    if(pid == 0) {
   //int x=1;
  // printf(1, "address %x\n", &x);
    test(1);
   //exec("null",0);
    exit();
    }
    wait();
   //test(1);
    exit();
}

```

پاسخ به سؤالات تشریحی:

۱. در لینوکس، هسته از نواحی virtual memory پیگیری memory mapping های پردازه استفاده میکند؛ مثلا یک process یک VMA برای کدش، یک VMA برای هر نوع دیتا، یک VMA برای هر memory mapping در صورت وجود داشتن، داشته باشد. هر VMA شامل تعدادی page هست که که هر کدام یک entry به page table دارد. اما در xv6، از آدرس های مجازی 32 بیتی استفاده میکند که فضای آدرسی مجازی 4گیگابایتی را ایجاد میکند. همچنین xv6 از ساختار جدول دو سطحی استفاده می کند. xv6 مفهومی از حافظه مجازی ندارد.

---

۲. در ساختار سلسله مراتبی، process ها و task ها به راحتی می‌توانند با به اشتراک گذاشتن کدها و داده‌ها توسط mapping بخش مناسب به صفحات فیزیکی از مصرف اضافی حافظه جلوگیری کنند.

---

۳. در مدخل سطح page directory برای اشاره به سطح بعدی از ۲۰ بیت استفاده می‌شود. همچنین ۱۲ بیت برای سطح دسترسی نگهداری می‌شود. این ۱۲ بیت در هر دو سطح وجود دارد اما در سطح page table از ۲۰ بیت برای آدرس صفحه فیزیکی استفاده می‌شود. در بیت D یعنی بیت dirty باهم تفاوت دارند. در page directory این بیت به این معنا است که صفحه باید در دیسک نوشته شود تا تغییرات اعمال شود، اما در page table این بیت معنایی ندارد.

---

۴. حافظه فیزیکی تخصیص می‌دهد. این کار با صدا زدن تابع kalloc انجام می‌شود.

همچنین این مسئله در کد xv6 برای این تابع ذکر شده:

```
// Allocate one 4096-byte page of physical memory.
```

---

۵. این تابع به منظور اتصال حافظه مجازی به حافظه فیزیکی استفاده می‌شود. همچنین این تابع صفحه جدید را به pgdir اضافه می‌کند.
کد این تابع به صورت زیر است و در توضیحات کامنت شده نیز موضوعات اشاره شده مشخص است:

```
60  static int
61  mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm)
62  {
63    char *a, *last;
64    pte_t *pte;
65
66    a = (char*)PGROUNDDOWN((uint)va);
67    last = (char*)PGROUNDDOWN(((uint)va) + size - 1);
68    for(;;){
69      if((pte = walkpgdir(pgdir, a, 1)) == 0)
70        return -1;
71      if(*pte & PTE_P)
72        panic("remap");
73      *pte = pa | perm | PTE_P;
74      if(a == last)
75        break;
76      a += PGSIZE;
77      pa += PGSIZE;
78    }
79    return 0;
80  }
```

۷. این تابع آدرس PTE موجود در pgdir را بازمی‌گرداند، همچنین اگر لازم باشد جدول مورد نیاز را می‌سازد. این تابع عمل سخت‌افزاری ترجمه آدرس مجازی به فیزیکی را شبیه‌سازی می‌کند. کد این تابع به صورت زیر است:

```
35  static pte_t *
36  walkpgdir(pde_t *pgdir, const void *va, int alloc)
37  {
38    pde_t *pde;
39    pte_t *pgtab;
40
41    pde = &pgdir[PDX(va)];
42    if(*pde & PTE_P){
43      pgtab = (pte_t*)P2V(PTE_ADDR(*pde));
44    } else {
45      if(!alloc || (pgtab = (pte_t*)kalloc()) == 0)
46        return 0;
47      // Make sure all those PTE_P bits are zero.
48      memset(pgtab, 0, PGSIZE);
49      // The permissions here are overly generous, but they can
50      // be further restricted by the permissions in the page table
51      // entries, if necessary.
52      *pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
53    }
54    return &pgtab[PTX(va)];
55  }
```

---

۸. توابع 'allocuvm' و 'mappages' مربوط به سیستم عامل استفاده می‌شوند تا حافظه مجازی را به پردازه‌ها اختصاص دهند و به صورت فیزیکی حافظه مربوطه را نگاه دارند. این توابع اصلی در سیستم عامل‌های مبتنی بر Unix مانند Linux و FreeBSD استفاده می‌شوند. تابع `allocuvm` به پردازه یک ناحیه حافظه مجازی را اختصاص می‌دهد. پردازه‌ها در یک سیستم عامل مبتنی بر Unix ممکن است از حافظه مجازی استفاده کنند که بیانگر مجموعه‌ای از آدرس‌های حافظه است که به صورت فیزیکی تخصیص نیافته‌اند.

تابع `allocuvm` به پردازه یک ناحیه از حافظه مجازی را اختصاص می‌دهد. پردازه‌ها در یک سیستم عامل مبتنی بر Unix ممکن است از حافظه مجازی استفاده کنند که بیانگر مجموعه‌ای از آدرس‌های حافظه است که به صورت فیزیکی تخصیص نیافته‌اند.

با فراخوانی تابع `allocuvm`، سیستم عامل یک ناحیه حافظه مجازی را برای پردازه ایجاد می‌کند و به آن ناحیه حافظه، آدرس‌هایی که ممکن است در آینده توسط پردازه استفاده شوند، اختصاص می‌دهد.

تابع `mappages` نیز برای ارتباط بین حافظه مجازی و صفحات حافظه فیزیکی استفاده می‌شود. زمانی که یک پردازه به یک ناحیه حافظه مجازی دسترسی پیدا می‌کند، سیستم عامل باید مکان صفحات مربوط به آن ناحیه حافظه را در حافظه فیزیکی مشخص کند. تابع `mappages` به سیستم عامل کمک می‌کند تا ناحیه حافظه مجازی را با صفحات حافظه فیزیکی مرتبط کند و نگاه دارد که اگر صفحات مورد نیاز فیزیکی وجود نداشته باشند، آن‌ها را ایجاد کند و به ناحیه مجازی ارتباط دهد.

در کل، توابع `allocuvm` و `mappages` در سیستم عامل برای مدیریت حافظه مجازی پردازه‌ها استفاده می‌شوند و نقش مهمی در اختصاص و مدیریت حافظه برای پردازه‌ها دارند.

---

۹. فراخوانی سیستمی `exec` در سیستم‌های عامل مبتنی بر Unix وظیفهٔ بارگذاری یک برنامه جدید به جای برنامهٔ فعلی را دارد. با استفاده از این فراخوانی، برنامهٔ جاری در حافظهٔ مجازی جایگزین می‌شود و برنامهٔ جدید بارگذاری می‌شود.

فراخوانی سیستمی `exec` در واقع یک سری تابع است که باعث جایگزینی فرآیند فعلی با برنامهٔ جدید می‌شود. توابع اصلی `exec` در زبان برنامه‌نویسی C شامل `execl`، `execvp`، `execve` و `execle` هستند. در این مثال، توضیحات بر اساس تابع `execve` ارائه می‌شوند که یکی از معمول‌ترین توابع `exec` است.

شیوهٔ بارگذاری برنامه با استفاده از فراخوانی سیستمی `execve` عبارت است از:

1. تهیه پارامترهای لازم: برای فراخوانی `execve`، شما نیاز به تهیهٔ سه آرگومان اصلی دارید:

- مسیر فایل برنامهٔ جدید: مسیر فایل برنامه‌ای که می‌خواهید جایگزین برنامهٔ جاری شود.
- آرگومان‌ها (argv): یک آرایه از رشته‌ها که آرگومان‌ها را برای برنامهٔ جدید مشخص می‌کند. آخرین عنصر آرگومان‌ها باید NULL باشد.
- متغیرهای محیطی (envp): یک آرایه از رشته‌ها که متغیرهای محیطی را برای برنامهٔ جدید تعیین می‌کند. آخرین عنصر متغیرهای محیطی باید NULL باشد.

2. فراخوانی `execve`: با استفاده از تابع `execve`، پارامترهای لازم را به عنوان ورودی به آن ارسال کنید.
3. اجرای برنامهٔ جدید: در صورت موفقیت آمیز بودن فراخوانی `execve`، برنامهٔ جدید بر روی فضای حافظهٔ مجازی جایگزین برنامهٔ فعلی می‌شود و اجرا می‌شود.

اهمیت اصلی فراخوانی `exec` در این است که برنامهٔ جدید به جای برنامهٔ قبلی بارگذاری می‌شود، اما فرآیند (PID) و دستهٔ فرآیند (PGID) باقی می‌مانند. بنابراین، برنامهٔ جدید می‌تواند با استفاده از تمام منابع و حقوق دسترسی برنامهٔ قبلی به اجرای خود ادامه دهد.
