به نام خدا

گزارش پروژه پنجم آزمایشگاه سیستم عامل

پریا خوش‌تاب ۸۱۰۱۹۸۳۸۷ - علی اخگری ۸۱۰۱۹۸۳۴۱ - پرنیان فاضل ۸۱۰۱۹۸۵۱۶

۱- چرا ساختار سلسله مراتبی منجر به کاهش مصرف حافظه می گردد؟

در ساختار سلسله مراتبی page table، در صورت وجود مناطق بزرگی از حافظه که بلا استفاده هستند، page table می تواند فضای بسیار کمتری را مصرف کند.

با توجه به اینکه هر منطقه ای از حافظه که برای ناحیه ای که توسط شماره صفحه سطح بالا پوشش داده نشده است، نیازی به تخصیص جدول صفحه سطح پایین تر ندارد، از این رو از اختصاص جدول صفحه سطح پایین برای آن ناحیه جلوگیری می کند و در نتیجه منجر به کاهش حافظه می شود.

۲- روشی برای تخمین فرکانس دسترسی به صفحه های حافظه به کمک بیت های مدخل جدول صفحه ارائه دهید.

در صورتی که پردازنده به یک مکان حافظه در صفحه دسترسی داشته باشد، MMU بیت های قابل دسترسی را در page descriptor به روز می کند. بیت قابل دسترسی در page descriptor

می تواند برای شناسایی "سن" ورودی page table استفاده شود. فرآیند تعویض سیستم عامل به صورت دوره ای بیت های دسترسی به جدول های صفحه را بازنشانی می کند. سپس هنگامی که یک فرآیند به یک صفحه دسترسی پیدا می کند، پردازنده می تواند تشخیص دهد که از آخرین بازنشانی مقادیر از کدام صفحات استفاده شده است. در لینوکس، این اساس الگوریتم حداقل استفاده شده (LRU) را تشکیل می دهد که صفحات نامزدی را که ممکن است به mass storage مبادله شوند انتخاب می کند.

۳ـ تابع kalloc چه نوع حافظه ای تخصیص می دهد؟

حافظه فیزیکی تخصیص می دهد، به این صورت که یک صفحه ۴۰۹۶ بایتی از حافظه فیزیکی تخصیص می دهد. در صورتی که بتواند این مقدار حافظه را تخصیص دهد، یک اشاره گر بر می گرداند که کرنل بتواند استفاده کند. در غیر این صورت 0 را برمیگرداند که به این معنی است که امکان تخصیص حافظه وجود ندارد.

```
79  // Allocate one 4096-byte page of physical memory.
80  // Returns a pointer that the kernel can use.
81  // Returns 0 if the memory cannot be allocated.
82  char*
83  kalloc(void)
84  {
85    struct run *r;
86
87    if(kmem.use_lock)
88      acquire(&kmem.lock);
89    r = kmem.freelist;
90    if(r)
91      kmem.freelist = r->next;
92    if(kmem.use_lock)
93      release(&kmem.lock);
94    return (char*)r;
95  }
96
97
```

۴- تابع mappages چه کاربردی دارد؟

این تابع یک حافظه مجازی (va) را به حافظه فیزیکی (pa) در page table موجود (pgdir) نگاشت میکند. در واقع این تابع PTE ها را برای آدرس های مجازی تشکیل می دهد و صفحه جدید را به pgdir اضافه میکند. اگر این نگاشت موفقیت آمیز باشد، این تابع 0 و در غیر اینصورت -1 برمیگرداند.

```
57  // Create PTEs for virtual addresses starting at va that refer to
58  // physical addresses starting at pa. va and size might not
59  // be page-aligned.
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  }
```

۵ـ راجع به تابع walkpgdir توضیح دهید. این تابع چه عمل سخت افزاری را شبیه سازی می کند؟

این تابع به آدرس مجازی که در آرگومان va است نگاه می کند و جایی که باید آن آدرس مپ شود را طبق pgdir، مشخص می کند و سپس آدرس PTE موجود در pgdir را باز میگرداند. همچنین در صورت لزوم، page table لازم را می سازد.

این تابع عمل سخت افزاری ترجمه آدرس مجازی به فیزیکی را شبیه سازی می کند.

```
32  // Return the address of the PTE in page table pgdir
33  // that corresponds to virtual address va.  If alloc!=0,
34  // create any required page table pages.
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  }
```

۶ـ دو نقص نگاشت فایل در حافظه نسبت به خواندن عادی در فایل ها را بیان کنید.

1- یک خطای ورودی/خروجی در یک فایل نگاشت شده با حافظه را نمی توان توسط SQLite شناسایی و برطرف کرد. در عوض، خطای I/O باعث ایجاد سیگنالی می‌شود که اگر توسط برنامه شناسایی نشود، منجر به خرابی برنامه می‌شود.

2- سیستم عامل باید یک حافظه پنهان بافر یکپارچه داشته باشد تا پسوند I/O نگاشت شده با حافظه به درستی کار کند، به خصوص در شرایطی که دو پردازش به یک فایل پایگاه داده دسترسی دارند و یکی از آنها از I/O نگاشت شده با حافظه استفاده می کند و دیگری نیست. همه سیستم عامل ها دارای حافظه پنهان بافر یکپارچه نیستند. در برخی از سیستم‌عامل‌هایی که ادعا

می‌کنند یک حافظه پنهان بافر یکپارچه دارند، پیاده‌سازی باگ است و می‌تواند منجر به خراب شدن پایگاه‌های داده شود.

3- عملکرد همیشه با ورودی/خروجی نقشه‌برداری شده با حافظه افزایش نمی‌یابد. در واقع، می توان موارد آزمایشی را ساخت که در آن عملکرد با استفاده از I/O نگاشت شده با حافظه کاهش می یابد.