(یا مرتجی)

پروژه پایانترم سیستم عامل

زهرا عباسقلی ۴۰۰۵۲۲۰۲۲

معصومه غفاری ۴۰۰۵۲۲۰۸۵

فعالیت

به تابع schedule رفته و نحوه کار کد مربوطه را توضیح دهید. ساختار ptable و جزییات داخل تابع که برای شما ناشناخته است را تحقیق کرده و توضیح دهید.

نحوه کار تابع scheduler :

تابع scheduler مسئولیت مدیریت اجرای فرآیندها روی CPU را دارد. این تابع به صورت حلقه بی‌نهایت اجرا می‌شود و فرآیندهایی را که در حالت RUNNABLE(قابل اجرا) هستند، برای اجرا انتخاب می‌کند.

ساختار ptable :

تمام فرآیندها در یک جدول به نام ptable (process table) نگهداری می‌شوند. این جدول شامل آرایه‌ای از ساختارهای proc است که هر کدام اطلاعات مربوط به یک فرآیند را نگهداری می‌کند.

- lock : یک قفل برای جلوگیری از شرایط رقابتی (race conditions) در دسترسی به ptable
- proc[NPROC]: آرایه‌ای از ساختارهای proc که هر کدام نمایانگر یک فرآیند هستند.

  ساختار procشامل اطلاعات مختلفی درباره یک فرآیند است؛ مثل وضعیت(state)، شناسه فرآیند(pid)، محلی که CPU باید برای اجرای این فرآیند به آنجا بپرد(context)، پشته کرنل برای این فرآیند(kstack).

جزئیات داخل تابع:

```
322  void scheduler(void)
323  {
324    struct proc *p;
325    struct cpu *c = mycpu();
326    c->proc = 0;
327    for(;;){
328      // Enable interrupts on this processor.
329      sti();
330
331      // Loop over process table looking for process to run.
332      acquire(&ptable.lock);
333      for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
334        if(p->state != RUNNABLE)
335          continue;
336
337        // Switch to chosen process.  It is the process's job
338        // to release ptable.lock and then reacquire it
339        // before jumping back to us.
340        c->proc = p;
341        switchuvm(p);
342        p->state = RUNNING;
343        swtch(&(c->scheduler), p->context);
344        switchkvm();
345
346        // Process is done running for now.
347        // It should have changed its p->state before coming back.
348        c->proc = 0;
349      }
350      release(&ptable.lock);
351    }
352  }
```

- for(;;) : این حلقه بی‌نهایت است و زمان‌بند دائماً اجرا می‌شود.
- sti() : وقفه‌ها را فعال می‌کند تا CPU بتواند به وقفه‌های سخت‌افزاری پاسخ دهد.
- acquire(&ptable.lock) : قفل جدول فرآیند را می‌گیرد تا از شرایط رقابتی جلوگیری کند.
- for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) : از اولین فرآیند تا آخرین فرآیند در جدول فرآیند پیمایش می‌کند.
- if(p->state != RUNNABLE) continue; : اگر فرآیند قابل اجرا نیست، به فرآیند بعدی می‌رود.
- proc = p : فرآیند انتخاب شده را به عنوان فرآیند فعلی تنظیم می‌کند.
- switchuvm(p) : فضای آدرس فرآیند کاربر را تنظیم می‌کند.
- p->state = RUNNING : وضعیت فرآیند را به RUNNINGتغییر می‌دهد.

- swtch(&cpu->scheduler, proc->context) : به زمینه (context) فرآیند منتخب سوییچ می‌کند. اینجا مکانیزمی است که فرآیند فعلی CPU را به فرآیند منتخب تغییر می‌دهد.
- Switchkvm() : فضای آدرس کرنل را دوباره فعال می‌کند.
- proc = 0 : فرآیند فعلی را به صفر (null) تنظیم می‌کند.
- release(&ptable.lock) : قفل جدول فرآیند را آزاد می‌کند تا سایر بخش‌ها نیز بتوانند به جدول فرآیند دسترسی داشته باشند.

فعالیت

در چه صورت زمان‌بند اجرا می‌شود؟ درباره نحوه قبضه در این سیستم‌عامل تحقیق کنید. نحوه کار کلی سخت‌افزار مربوط به Timer را به طور خلاصه توضیح داده و سعی کنید محل تنظیم این سخت‌افزار را پیدا کنید.

زمان‌بند در شرایط مختلفی اجرا می‌شود. این شرایط شامل تمام شدن برش زمانی، اتمام یا تعلیق فرآیند، وقوع وقفه تایمر و فراخوانی صریح زمان‌بند است. برای درک بهتر می‌توانیم به بررسی توابعی بپردازیم که به نوعی باعث اجرای زمان‌بند می‌شوند.

- تابع scheduler : این تابع وظیفه اجرای زمان‌بند را بر عهده دارد و فرآیند جدیدی را برای اجرا انتخاب می‌کند.
- تابع yield : این تابع فرآیند فعلی را متوقف کرده و به زمان‌بند اجازه می‌دهد فرآیند دیگری را انتخاب کند.
- تابع sleep : این تابع فرآیند فعلی را به حالت خواب (sleep) می‌برد و زمان‌بند را اجرا می‌کند.
- تابع exit : این تابع فرآیند فعلی را خاتمه می‌دهد و زمان‌بند را اجرا می‌کند.
- وقفه تایمر : وقتی وقفه تایمر رخ می‌دهد، زمان‌بند اجرا می‌شود تا فرآیند جدیدی را برای اجرا انتخاب کند.

نحوه قبضه در این سیستم عامل:

قبضه عمدتاً از طریق وقفه‌های تایمر مدیریت می‌شود. تایمر سخت‌افزاری به صورت دوره‌ای وقفه ایجاد می‌کند. این وقفه‌ها باعث می‌شوند که کنترل از فرآیند در حال اجرا به هسته سیستم‌عامل منتقل شود. هنگامی که وقفه تایمر رخ می‌دهد، CPU برای مدیریت وقفه تایمر به تابع trap می‌رود. در داخل مدیریت وقفه، اگر وقفه تایمر باشد، زمان‌بند برای اختصاص CPU به فرآیند دیگری صدا زده می‌شود. این کار از طریق فراخوانی تابع yield انجام می‌شود. تابع yield، وضعیت فرآیند فعلی را به RUNNABLE تغییر می‌دهد و سپس تابع sched را صدا می‌زند که وظیفه انتخاب فرآیند بعدی برای اجرا را دارد.

نحوه کار سخت‌افزار تایمر:

تایمر برای تولید وقفه‌های منظم استفاده می‌شود تا سیستم‌عامل بتواند وظایف زمان‌بندی و قبضه را به درستی انجام دهد. تنظیمات مربوط به تایمر در فایل lapic.c انجام می‌شود. به طور خاص، تابع lapicinit برای تنظیم تایمر استفاده می‌شود. در این تابع، تایمر با استفاده از مقادیر مختلف تنظیم می‌شود تا وقفه‌های منظم تولید کند.

محل تنظیم سخت افزار تایمر:

در تابع main، تنظیمات تایمر و وقفه‌های مربوط به آن انجام می‌شود:

```c
int
main(void)
{
  kinit1(end, P2V(4*1024*1024)); // phys page allocator
  kvmalloc();      // kernel page table
  mpinit();        // detect other processors
  lapicinit();     // interrupt controller
  seginit();       // segment descriptors
  picinit();       // disable pic
  ioapicinit();    // another interrupt controller
  consoleinit();   // console hardware
  uartinit();      // serial port
  pinit();         // process table
  tvinit();        // trap vectors
  binit();         // buffer cache
  fileinit();      // file table
  ideinit();       // disk
  startothers();   // start other processors
  kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must come after startothers()
  userinit();      // first user process
  mpmain();        // finish this processor's setup
}
```

فعالیت

طول هر بازه زمانی را چگونه افزایش می‌دهیم؟ طول برش زمانی مگر یک واحد پیوسته بر حسب ثانیه نیست؟ سیستم‌عامل بر چه حسب این زمان را حساب می‌کند و چه مفهومی دارد؟

برای اضافه کردن قابلیت به زمان بند Round Robin که اگر هیچ فرآیندی در یک دور کارکرد این زمان بند تمام نشده باشد، طول برش زمانی تخصیصی در دوره دوبرابر شود، می‌توان به صورت زیر عمل کرد.

۱. برای هر دور کارکرد زمان بند، یک شمارنده (counter) می سازیم. این شمارنده نشان‌دهنده تعداد فرآیندهایی است که در طول آن دور کارکرد تمام نشده‌اند.
۲. طول برش زمانی را در هر دور به طور معمول تخصیص می دهیم.
۳. هنگامی که یک فرآیند کار خود را تمام کرد، شمارنده را کاهش می دهیم
۴. در آخر دور کارکرد، اگر شمارنده برابر صفر باشد (یعنی هیچ فرآیندی در طول دور کارکرد تمام نشده باشد)، طول برش زمانی را دوبرابر می کنیم.

```c
#include <stdio.h>
#define QUANTUM 1

int calculate_time_slice(int counter)
{
    if (counter == 0)
    {
        return QUANTUM * 2;
    }
    else
    {
        return QUANTUM;
    }
}
```

```c
int main()
{
    int processes[5] = {3, 4, 2, 1, 5};
    int n = sizeof(processes) / sizeof(processes[0]);
    int i, counter = n;
    while (counter > 0)
    {
        for (i = 0; i < n; i++)
        {
            if (processes[i] > 0)
            {
                int time_slice = calculate_time_slice(counter);
                if (processes[i] <= time_slice)
                {
                    printf("Process %d completed in %d unitsn", i, processes[i]);
                    counter--;
                    processes[i] = 0;
                }
                else
                {
                    printf("Process %d executed for %d unitsn", i, time_slice);
                    processes[i] -= time_slice;
                }
            }
        }
    }
    return 0;
}
```

در این کد، فرآیندها به صورت لیست processes و با طول زمان اجرایی خود به طول آرایه نمایش داده شده‌اند. تابع calculate_time_slice براساس شمارنده counter ، مقدار بازه زمانی را محاسبه می‌کند.

سپس در حلقه while وfor ، طول برش زمانی تخصیص داده می‌شود و بسته به طول زمان اجرایی فرآیندها، اجرای آنها انجام می‌شود.

۱. تابع trap:

- این تابع در سیستم‌عامل برای پردازش توقف‌ها (traps) از طریق تراپ‌فریم‌ها (trapframe) استفاده می‌شود.
- تراپ‌فریم‌ها حاوی اطلاعات مربوط به وقوع توقف‌ها (مانند نوع توقف، مقادیر ثبت‌های ماشین و غیره) هستند.
- در زمان وقوع توقف، کنترل از حالت کاربری به حالت هسته‌ای منتقل می‌شود و به تابع تراپ‌هندلر (trap handler) هدایت می‌شود.
- تراپ‌هندلر نوع توقف را بررسی کرده و عملیات مناسب را انجام می‌دهد، مانند خاتمه دادن به برنامه یا انجام عملیات خاص به نمایندگی از برنامه.
- در مورد توقف‌های خاصی مانند تایمر، دستگاه‌های ورودی/خروجی (مثل دستگاه‌های IDE و کیبورد) و خطاهای ناخواسته نیز عملیات مشخصی انجام می‌شود.

۲. متغیر ticks:

- در کد، متغیر ticks به عنوان شمارنده‌ای برای تعداد تیک‌های سیستمی (system ticks) استفاده می‌شود.
- تیک‌های سیستمی به عنوان یک واحد زمانی مشخص (مثلاً میلی‌ثانیه) برای محاسبه زمان‌ها و توقف‌ها در سیستم‌عامل استفاده می‌شوند.
- در تابع trap، هر بار که تایمر تیک می‌زند، مقدار ticks افزایش می‌یابد.
- این متغیر برای محاسبه زمان‌های اجرایی و توقف‌ها در سیستم‌عامل مورد استفاده قرار می‌گیرد.

فعالیت

تحلیل‌هایی روی این معیارها قبل و بعد از اجرای تغییرات ارائه دهید. توضیح دهید برنامه‌های شما چه تاثیری روی این معیارها دارند. گزارشی از معیارها و نحوه تغییر آن‌ها تحت تست‌های مختلف بنویسید.

تأثیرات تغییرات بر عملکرد سیستم

برای ارزیابی تأثیرات تغییرات در زمان‌بند، می‌توان برنامه‌های مختلفی را اجرا کرد و نتایج را قبل و بعد از اعمال تغییرات مقایسه کرد. به عنوان مثال، می‌توان برنامه‌هایی نظیر ضرب ماتریس، حلقه‌های تو در تو، عملیات ورودی و خروجی، و fork را آزمایش کرد. تأثیرات ممکن شامل موارد زیر است:

ضرب ماتریس و حلقه‌های تو در تو:

قبل از تغییرات: زمان‌بند به صورت ثابت برش‌های زمانی را اختصاص می‌دهد. بنابراین، برنامه‌هایی که نیاز به زمان پردازش بیشتری دارند ممکن است به کرات از اجرای مجدد جلوگیری شوند.

بعد از تغییرات: اگر هیچ فرایندی در یک دور به اتمام نرسد، برش زمانی دو برابر می‌شود. این باعث می‌شود برنامه‌هایی که نیاز به زمان پردازش بیشتری دارند (مانند ضرب ماتریس و حلقه‌های تو در تو) بتوانند زمان بیشتری را در هر برش زمانی دریافت کنند، که ممکن است به کاهش زمان کل اجرای برنامه منجر شود.

عملیات ورودی و خروجی:

قبل از تغییرات: برنامه‌هایی که عملیات ورودی و خروجی زیادی دارند ممکن است زمان بیشتری را در حالت انتظار (blocked) سپری کنند.

بعد از تغییرات: زمان‌بند به علت دو برابر شدن برش زمانی، ممکن است در برخی موارد باعث کاهش زمان انتظار و بهبود کارایی شود، اما این به میزان عملیات ورودی و خروجی بستگی دارد.

:Fork

قبل از تغییرات: ایجاد فرآیندهای جدید با fork و مدیریت آنها توسط زمان‌بند با برش‌های زمانی ثابت انجام می‌شود.

بعد از تغییرات: اگر هیچ فرایندی به اتمام نرسد، برش زمانی دو برابر می‌شود. این ممکن است باعث شود فرآیندهای جدید زمان بیشتری برای اجرای اولیه دریافت کنند.