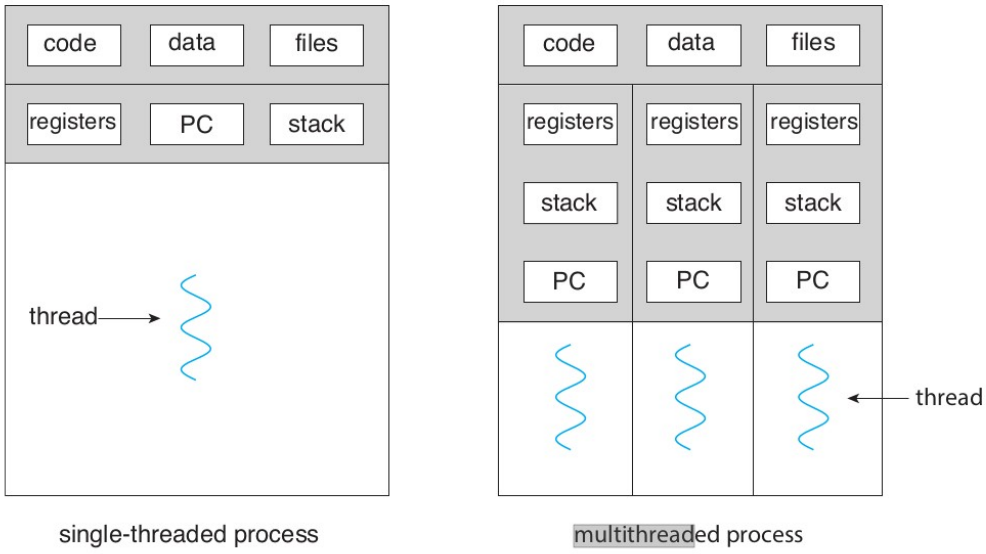
مریم سعیدمهر
ش.د. :۹۶۲۹۳۷۳
استاد مربوطه : دکتر زالی

تکلیف سری دوم سیستم عامل
دانشگاه صنعتی اصفهان
ترم اول سال تحصیلی ۹۸ – ۹۹

.1

(a) با رسم شکل ، تفاوت فرایند تک نخی و چند نخی در حافظه اصلی کامپیوتر را نشان دهید.



(b) PCB در عملیات تعویض متن چه نقشی دارد ؟ در لینوکس برای هر پروسس یا هر نخ یک PCB در نظر گرفته میشود ؟ بررسی کنید اسم ساختمان داده مربوطه در کرنل لینوکس چیست و در چه فایلی ذخیره شده ؟

نقش PCB در عملیات context switch به این شکل است که ، وقتی یک interrupt یا system call میرسد سیستم عامل موظف است تا پروسسی که در حال حاضر در CPU است را از آن خارج کرده و پروسس جدید را به CPU بفرستد تا اجرا شود اما برای اینکه بعداً دوباره بتواند اجرای پروسه ی ناتمام قبلی را ادامه دهد باید بداند که اجرای آن تا کجا پیش رفته پس باید یک سری اطلاعات از آن پروسه را ذخیره کند، ساختاری که برای این کار مناسب است همان PCB میباشد.

در لینوکس ،برای هر پروسس یک PCB در نظر گرفته میشود و نخ های آن پروسس در ساختاری با عنوان TCB ذخیره میشوند که از PCB لینک میشوند.

ساختاری که پروسس ها با آن در کرنل مشخص میشوند یک ساختار پیاده شده با زبان C است

به نام task_struct میباشد که درفایل <include/linux/sched.h> در دایرکتوری سورس کد کرنل موجود است(در سیستم من : /usr/src/linux-source-5.0.0/include/linux/sched.h) همچنین در کرنل لینوکس همه ی پروسس های فعال در یک لینک لیست دوطرفه ای از task_struct تعریف میشوند و کرنل پوینتری به پروسس در استیت رانینگ دارد که به PCB پروسس فعلی موجود در CPU اشاره میکند همچنین این لینک لیست به process table معروف است و در proc/ قرار دارد.

(c) تفاوت ها و شباهت های سیگنال و اینتراپت :

ماهیت هر دویشان تقریباً یکسان است از این لحاظ که هردو اطلاع رسانی انجام میدهند! تفاوتشان در این است که سیگنال در حقیقت یک اینتراپت بین پروسه ای(یا بین نخ های یک پروسه)است یعنی در سطح یوزر خواهد بود و صرفاً یکی از روش های IPC است درحالی که اینتراپت از سطح کاربر به کرنل فرستاده میشود(با سیستم کال) یا در سطح سخت افزار(از I/O به CPU) است.

(d)چه عاملی باعث تغییر وضعیت

i. مستقیم از آماده به اجرا : scheduler dispatch (short-term scheduler تصمیم میگیرد که این تغییر وضعیت رخ دهد)

ii. از اجرا به انتظار : اگر I/O request اتفاق بیوفته یا پروسه ی فرزندی تولید شده باشد و این پروسه والد ملزم به صبر کردن برای اتمام اجرای فرزندش باشد یا در این پروسه منتظر اینتراپتی باشیم و در کل منتظر رخدادی باشیم ، از موضع اجرا به انتظار میرویم.

iii. انتظار به معلق : اگر waiting queue پر باشد آنگاه mid-term scheduler تصمیم میگیرد تا عملیات swapping را انجام دهد و درجه ی multiprogramming را کاهش دهد یعنی PCB یک پروسس را از مموری(RAM) خارج کرده و به disk منتقل کند.

2. در مورد IPC :

(a) با رسم شکل تفاوت شان را مشخص کرده و توضیح دهید.

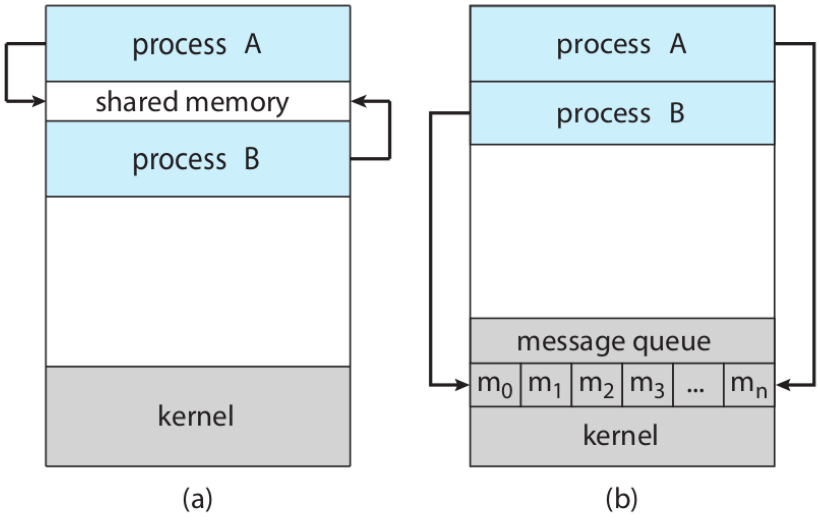


Figure 3.11 Communications models. (a) Shared memory. (b) Message passing

مهم ترین تفاوت اینها در دخالت کرنل موقع انجام تبادلاتشان است. در مدل حافظه مشترک ، همانطور که در شکل نیز پیداست ، دو پروسه برای تبادلاتشان نیازی به کمک کرنل ندارند و فقط برای ایجاد این حافظه مشاع از سیستم کال و در واقع کرنل کمک میگیرند در حالی که در مدل انتقال پیام ، برای تبادل هر پیام باید سیستم کال مربوطه فراخوانی شود و این عملیات در کرنل انجام میشود.

(b) مزایا و معایب هر کدام ؟

مدل message passing برای تبادل اطلاعات کوچک تر مناسب تر است همچنین پیاده سازی آن راحت تر میباشد به این دلیل که پیچیدگی کمتری دارد و عملیات کنترلی عمدتاً به عهده ی کرنل است و نه کاربر در حالی که در مدل دوم ، عملیات کنترلی برعهده ی کاربر است.

اما مدل shared memory سریع تر است زیرا در تبادل هر پیام باید از سیستم کال استفاده شود که خود برای سیستم سربار به همراه خواهد داشت(به دلیل اجرای سیستم کال ها در کرنل) ولی در این مدل ، فقط در ایجاد حافظه مشترک از سیستم کال ها استفاده میشود و بعد از آن همگی پروسه های مذکور میتوانند از آن استفاده کنند و نیازی به کمک کرنل نیست.

3. از برنامه زیر چه نتیجه ای میگیرید.

```
int main()
{
        pid_t pid;
        pid = fork();
        int var = 0;
        if (pid == 0) { /* child process */
                printf("Start of the child\n");
                var = 10;
                printf("var address and value in child process: 0x%x: %d\n ", &var, var);
                printf("End of the child\n");
                exit(0);
        }
        else { /* parent process */
                printf("Start of the Parrent Process\n");
                var = 20;
                printf("var address and value in parrent process: 0x%x: %d\n ", &var, var);
                wait(NULL);
                printf("End of the Parrent Process\n");
        }
        return 0;
}
```

آنچه مشاهده شد

Start of the parent
Var address and value in parent process: 0xd2be53b0 : 20
Start of the child
Var address and value in child process: 0xd2be53b0 : 10
End of the child
End of the parent

نتیجه : پروسه فرزند ، در حقیقت یک کپی از فضای آدرس پروسه پدر است، برای همین است که آدرس متغیر var در هر دو یکسان است اما چون اجرای پروسه ها مستقل است یعنی فضای حافظه شان مشترک نیست (فقط کپی یکدیگرند ولی محتوای متغیر ها در هر کدام متفاوت است البته از بعد از فراخوانی fork)پس مقدار متغیر در هرکدام متفاوت است.

4. برای هریک از موارد زیر کدام روش IPC را پیشنهاد میدهید؟

(a) ارسال/دریافت اطلاعات به/از سرویس جستجوی گوگل در برنامه نویسی سایت ها برای افزودن امکان جستجو بکار میرود. استفاده از روش socket معقول تر است به دلیل وجود بستر اینترنت و پروتکل های UDP/TCP

(b) ارسال نتیجه عملیات پروسه پدر به پروسه های فرزند استفاده از pipe در این حالت متداول تر

است زیرا کار با آن ساده تر است .

5. در مورد Amazon Web Service

(a) در وبسایت آمازون از thread pool استفاده شده. مزیت استفاده از thread pool به جای thread چیست ؟

اولاً با ساخت هر نخ ، سیستم متحمل سربار خواهد شد و این زمان اجرای فرایند را تحت تاثیر میگذارد ولی اگر به اندازه ظرفیت سیستم ،در همان ابتدای راه اندازی سیستم ، نخ بسازیم یک بار متحمل این سربار میشویم و مهم تر اینکه این سربار از دید کاربر مخفی خواهد ماند(دقیقاً کاری که thread pool میکند)

در ثانی ، در استفاده از نخ(به صورت خالص) در واقع با رسیدن هر درخواست باید یک نخ ساخته شود ، حال در شرایطی که نخ های سیستم تمام شده ، ناچار به miss کردن درخواست ها میشویم در حالی که thread pool با فراهم کردن یک صف برای ذخیره درخواست ها در شرایطی که نخ آزاد ندارد ، تا حد خوبی از miss شدن درخواست ها جلوگیری کرده است.

(b) چه اتفاقی ممکن است برای سیستم رخ داده باشد که با وجود thread pool باز هم یک سری درخواست ها از بین رفتند ؟

همانطور که در قسمت (a) اشاره شد ، ایده ی thread pool برای جلوگیری از miss شدن درخواست ها ، استفاده از یک صف بود . حال اگر این صف هم پُر شود دیگر راهکاری وجود ندارد و سیستم ناچار میشود درخواست های جدید را miss کند.

6. با توجه به کدهای p1.c و p2.c پاسخ دهید:

(a) کامل کنید.

P1.c

```
int m1[100];
int m2[100];
int m_reply[100];

f1();
read_matrix(char* matrix_name);
print_matrix(int* m);

int main(int argc, char** argv){
        pthread_t t1, t2;
        read_matrix(argv[1]);
        read_matrix(argv[2]);

        pthread_attr_t attr;
        pthread_attr_init(&attr);

        pthread_create(&t1, &attr, f1, args1);//args1={i:0,j:4,k:0,l:4}
        pthread_create(&t2, &attr, f1, args2);//args2={i:5,j:9,k:5,l:9}

        pthread_join(t1, NULL);
        pthread_join(t2, NULL);

        print_matrix(&m_reply);

        return 0;
}
```

P2.c

```
int main(){
        pid_t pid1, pid2;
        pid1 = fork()
        if ( pid1 == 0 ){// in child
                execl("./p1","m1.txt","m2.txt");
                exit(0);}

        else{// in parent
                pid2 = fork()
                if ( pid2 == 0 ){// in child
                        execl("./p1","mr.txt","m3.txt");
                        exit(0);}
                wait(NULL);
        }

        show_m("mr.txt");
        return 0;
}
```

(b) توضیح دهید در هر بخش از کدام یک از موازی سازی عملیات یا موازی سازی داده استفاده شده است؟

در قطعه کد P1.c از موازی سازی داده استفاده شده زیرا برای محاسبه ی حاصل ضرب دو ماتریس از دو نخ استفاده شده یعنی قرار است یک task (محاسبه حاصل ضرب دو ماتریس) انجام شود و چیزی که بین نخ ها تقسیم میشه درواقع همون داده میباشد.

(c) به طور کلی توضیح دهید استفاده از pipe چه کمکی در بهبود کارایی این برنامه میتوانست بکند؟

استفاده از pipe ازاین جهت میتوانست کارا باشد که دیگر نیاز به ذخیره سازی حاصل ضرب m1 و m2 در فایل جداگانه و سپس باز کردن مجددفایل حاصل عملیات قبل برای انجام عملیات بعدی نبود و میتوانستیم همزمان با هر مرحله تولید حاصل عملیات اول ، عملیات دوم را ادامه دهیم.