دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

آزمایشگاه سیستم عامل

علی فانیان

زینب زالی

تابستان ۹۸

مدیریت پروسس‌ها و برنامه‌های چند پروسه‌ای

وقتی برنامه‌ای اجرا می‌شود، یک فرآیند (پروسس) در سیستم ایجاد می‌شود. اکنون این فرآیند می‌تواند حین اجرا فرآیندهای دیگری نیز ایجاد کند. در این صورت فرآیند اولیه، والد و فرآیندی که توسط فرآیند اول ساخته می‌شود را فرآیند فرزند گویند. همچنین ممکن است هر فرآیند فرزند، فرآیندهای فرزندی برای خود ایجاد کند. در این جلسه، با فراخوانی‌های سیستمی برای ایجاد و مدیریت فرآیندها آشنا می‌شویم.

مثالی از برنامه‌های چندفرآینده (multiprocess) سرویس sshd است که جهت ایجاد ارتباط ssh با ماشین موردنظر استفاده می‌شود. کلاینت‌های متفاوتی ممکن است مایل به برقراری ارتباط ssh با یک ماشین مشخص باشند. بدین منظور sshd روی ماشین موردنظر (سرور) اجرا می‌شود. این فرآیند درخواست‌های کلاینت‌ها برای اتصال ssh را دریافت می‌کند. سپس به ازای هر کلاینت یک پروسس جدید ایجاد می‌کند تا ارتباط او را هندل کند. بدین ترتیب امکان اتصال بیش از یک کلاینت به صورت همزمان فراهم می‌آید و یک جلسه اختصاصی برای هر کلاینت و ماشین هدف (سرور) ایجاد می‌شود.

همچنین وب سرورها هم می‌توانند ساختار مشابهی داشته باشند. هنگامی که یک کلاینت یک صفحه را از یک وب سرور درخواست می‌کند اگر سرور پاسخ آن کلاینت را بدهد و سپس به درخواست کلاینت دیگری گوش کند در آن فاصله زمانی، بسیاری از درخواستها به سرور بی‌پاسخ مانده و شکست می‌خورد. بنابراین اگر سرور بتواند هر درخواستی که دریافت می‌کند را به شکل موازی با درخواست‌های دیگر پاسخ دهد می‌تواند بسیار کاراتر عمل کند و در یک زمان به تعداد بیشتری کلاینت سرویس دهد. اینجاست که می‌توان برای پاسخگویی به درخواست‌های هر کلاینت از یک پروسس جداگانه استفاده کرد (وب سرورها از ترکیب چندنخی و چند پروسسی استفاده می‌کنند)

فراخوانی سیستمی fork

هنگامی که یک برنامه، این تابع را فراخوانی کند، سیستم عامل یک پروسس کاملاً همانند پروسس اول ایجاد می‌کند. اجرای هر دو پروسس از خط بعد از فراخوانی fork ادامه می‌یابد. خروجی این تابع در صورت موفقیت، صفر یا یک عدد مثبت است. با بررسی این خروجی می‌توان فهمید که الان کدام پروسس در حال اجرا است. اگر خروجی این تابع صفر باشد یعنی در پروسس فرزند هستیم و در صورتی که عدد مثبتی باشد یعنی در پروسس والد هستیم و عدد بازگشتی، مقدار شناسه پروسس فرزند ایجادشده است. در صورتی که سیستم نتواند پروسس جدیدی ایجاد کند، fork یک عدد منفی برمی‌گرداند. در ادامه تابع fork و تعدادی دیگر از فراخوانی‌های سیستمی مدیریت پروسس شرح داده می‌شود.

```
pid_t fork(void);
```

یک پروسس همانند پروسس والد ایجاد می‌کند.

```
pid_t waitpid(pid_t pid, int *status, int options);
```

پروسسی که این تابع را فراخوانی می‌کند منتظر پایان پروسس فرزندی با شناسه pid می‌ماند. مقدار status مقداری است که توسط پروسس فرزند بعد از اتمام به پروسس والد برمی‌گردد. در حالتی که آرگومان سوم مقداردهی نشود به صورت پیش‌فرض پروسس تا اتمام پروسس فرزند منتظر می‌ماند (گزینه‌هایی برای آرگومان سوم وجود دارد که حالات دیگری برای پایان انتظار wait در نظر می‌گیرد به man مراجعه کنید)

```
pid_t wait(int *status);
```

پروسسی که این تابع را فراخوانی می‌کند منتظر پایان یکی از پروسس‌های فرزند خود می‌ماند. مقدار بازگشتی این تابع، شناسه پروسس فرزندی است که پایان یافته است.

```
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[], char *const envp[]);
```

هرگاه یکی از این توابع فراخوانی شوند، برنامه‌ای که آدرس آن در path ذکر شده است با پروسس اولیه (پروسس فراخواننده این تابع) جایگزین شده و اجرا می‌شود. برای بررسی دقیق‌تر آرگومان‌های ورودی هر یک از توابع به man مراجعه کنید.

```
exit (status)
```

پروسس جاری را خاتمه داده و status را به پروسس والد برمی‌گرداند (مقداری که توسط wait قابل بازیابی است. به عبارت دقیقتر status&0377 به پروسس والد بازمی‌گردد. برای اطلاعات دقیقتر به man wait مراجعه کنید).

```
pid_t getpid(void);
```

شناسه پروسس جاری (پروسس فراخواننده این تابع) را برمی‌گرداند.

```
pid_t getppid(void);
```

شناسه پروسس والد پروسس جاری را برمی‌گرداند.

```
unsigned int sleep(unsigned int seconds);
```

فراخوانی این تابع، باعث می‌شود پروسس فراخواننده به مدت seconds ثانیه به حالت sleep برود.

فراخوانی های سیستمی برای مدیریت زمان

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

این تابع، مقدار زمان جاری را در آرگومان tv برمی‌گرداند، همچنین tz حاوی timezone سیستم می‌باشد. مقدار tv به صورت timestamp می‌باشد (مقدار زمان گذشته از یک مبدأ زمانی استاندارد معروف به epoch) که حاوی ثانیه و میکروثانیه گذشته از epoch است.

```
time_t time (time_t *seconds)
```

مقدار ثانیه گذشته از epoch را برمی‌گرداند. همچنین اگر آرگومان seconds مقدار NULL نداشته باشد زمان بازگشتی در محل این اشاره‌گر هم ذخیره می‌شود.

در ادامه مثال‌هایی از فراخوانی‌های سیستمی معرفی‌شده آمده است.

مثال‌ها

fork

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
	pid_t pid;
	pid=fork();
	int inChild=0;
	if(pid==0)
	{
		inChild=1;
	}

	while(inChild==0)
	{
		printf("this is Parent\n");
		sleep(1);
	}

	while(inChild==1)
	{
		printf("this is Child\n");
		sleep(1);

	}

	return 0;
}
```

این برنامه را با نام sample_fork.c ذخیره کرده کامپایل (gcc sample_fork.c -o sample_fork) و اجرا کنید. حین اجرای این پروسس، با اجرای ps -aux| grep sample_fork می‌توانید اطلاعات دو پروسس والد و فرزند ایجادشده را مشاهده کنید.

waitpid

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#define MAXCHILD 5
int main(){
	pid_t child [MAXCHILD];
	int inChild=0;
	int status=0;
	for (int i=0;i<MAXCHILD;i++){
		child[i]=fork();
		if(child[i]==0){
			inChild=1;
			break;
		}
	}
	while (inChild==1){
		srand(getpid());
		int r = rand()%10;
		printf("message from child %d : sleeping %d seconds\n",getpid(), r);
		sleep(r);
		inChild=-1;
	}
	while(inChild==0){
		sleep(1);
		for(int i=0;i<MAXCHILD;i++){
			int child_d;
			//**comment from next line
			child_d = wait(&status);
			if (child_d>0)
				printf("child[%d] is dead now \n",child_d);
			else if(child_d==-1)
				printf("no child to wait for \n");
			//**comment till this line
			// child_d = waitpid(child[i],&status,0);
			// if(child_d==0)
			//	printf("child[%d] is still alive\n",child[i]);
			// else if(child_d>0)
			//	printf("child[%d] is dead now \n",child[i]);
			// else
			//	printf("no specified child to wait for \n");
		}
	}
	return 0;
}
```

این برنامه را یک بار به همین صورت کامپایل و اجرا کنید و یک بار هم خطوط مشخص‌شده آخر برنامه را comment و خطوط commentشده قبل را uncomment کرده و کامپایل و اجرا کنید و نحوه اجرای دو برنامه را مقایسه نمایید.

Gettimeofday, time

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/time.h> //gettimeofday
#include <time.h>     //time

int main(int argc,char ** argv)
{
        struct timeval start,stop;
         srand(time(NULL));

        gettimeofday(&start,NULL);

        sleep(rand()%10);

        gettimeofday(&stop,NULL);

        long sec=stop.tv_sec-start.tv_sec;
        float m1=start.tv_usec;
        float m2=stop.tv_usec;

        long elapsed = sec*1000+(m2-m1)/1000;
        printf("%ld\n",elapsed);

        return 0;
}
```