

دانشگاه صنعتی اصفهان دانشکده برق و کامپیوتر آزمایشگاه سیستم عامل

# پیشگزارش جلسه چهارم

پاییز ۱٤٠٢



آزمایشگاه سیستم عامل دانشکده برق و کامپیوتر — دانشگاه صنعتی اصفهان پاییز ۱۴۰۲

# فهرست مطالب

ىدىرىت پروسسھا و برنامەھاى چند پروسەاى	۲
راخوانی های سیستمی برای مدیریت زمان	۴
شالهاشالها	۵
fork	
waitpid	۶
Status	Υ
Gettimeofday, time	



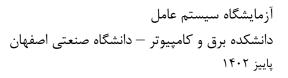
### مدیریت پروسسها و برنامههای چند پروسهای

وقتی برنامه ای اجرا می شود، یک فرآیند (پروسس) در سیستم ایجاد می شود. اکنون این فرآیند می تواند حین اجرا فرآیندهای دیگری نیز ایجاد کند. در این صورت فرآیند اولیه، والد و فرآیندی که توسط فرآیند اول ساخته می شود را فرآیند فرزند گویند. همچنین ممکن است هر فرآیند فرزند، فرآیندهای فرزندی برای خود ایجاد کند. در این جلسه، با فراخوانی های سیستمی برای ایجاد و مدیریت فرآیندها آشنا می شویم.

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

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

## فراخواني سيستمي fork





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

pid\_t fork(void);

يك يروسس همانند يروسس والد ايجاد مي كند.

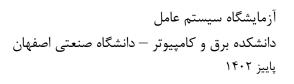
pid\_t waitpid(pid\_t pid, int \*status, int options);

پروسسی که این تابع را فراخوانی می کند منتظر پایان پروسس فرزندی با شناسه pid می ماند. مقدار status مقداری است که توسط پروسس فرزند بعد از اتمام به پروسس والد برمی گردد. در حالتی که آرگومان سوم مقداردهی نشود به صورت پیش فرض پروسس تا اتمام پروسس فرزند منتظر می ماند (گزینه هایی برای آرگومان سوم وجود دارد که حالات دیگری برای پایان انتظار wait در نظر می گیرد به man مراجعه کنید)

pid t wait(int \*status);

پروسسی که این تابع را فراخوانی می کند 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 در فضای آدرس پروسس اولیه (پروسس فراخواننده این تابع) جایگزین شده و اجرا می شود. برای بررسی دقیق تر آرگومان های ورودی هر یک از توابع به man مراجعه کنید.

#### exit (status)

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

pid\_t getpid(void);

شناسه پروسس جاری (پروسس فراخواننده این تابع) را برمی گرداند.

pid\_t getppid(void);

شناسه پروسس والد پروسس جاری را برمی گرداند.

unsigned int sleep(unsigned int seconds);

فراخوانی این تابع، باعث می شود 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 timezone 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 برمی گرداند. همچنین اگر آر گومان seconds مقدار NULL نداشته باشد زمان باز گشتی در محل این اشاره گر هم ذخیره می شود.

در ادامه مثالهایی از فراخوانیهای سیستمی معرفی شده آمده است.



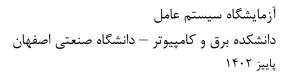
```
آزمایشگاه سیستم عامل
دانشکده برق و کامپیوتر — دانشگاه صنعتی اصفهان
پاییز ۱۴۰۲
```

#### 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 و اجرا کنید. حین اجرای این پروسس، با اجرای ps -aux | grep فردند ایجادشده را مشاهده کنید.





با استفاده از دستور زیر می توانید به اجرای یک پروسس مشخص با داشتن pid آن پایان دهید(kill کنید). این دستور در حقیقت سیگنالی به پروسس ارسال کرده و اجرای آن را پایان می بخشد (در جلسات آینده با این دستور بیشتر آشنا خواهید شد).

kill -9 <PID>

#### 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)
```



```
آزمایشگاه سیستم عامل
دانشکده برق و کامپیوتر — دانشگاه صنعتی اصفهان
پاییز ۱۴۰۲
```

```
// 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 مشخص شده آخر برنامه را comment و خطوط comment شده قبل را uncomment کرده و کامپایل و اجرا کنید و نحوه اجرای دو برنامه را مقایسه نمایید.

#### **Status**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
void task();
int main()
          pid_t pid;
          int status;
          while(1){
                     pid = fork();
                     if(pid == 0) {
                                task();
                     wait(&status);
                     printf("returned status : %d, real status %d \n", status, WEXITSTATUS(status));
          }
          return 0;
void task(){
          srand(time(NULL) + getpid());
          int r = rand()\%5;
          sleep(r);
          printf("%d : my random is %d\n", getpid(),r);
          exit (r);
```



```
آزمایشگاه سیستم عامل
دانشکده برق و کامپیوتر — دانشگاه صنعتی اصفهان
پاییز ۱۴۰۲
```

این برنامه با استفاده از تابع WEXITSTATUS مقدار درست status دریافتی از پروسس فرزند را نمایش میدهد.

#### 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;
```