# دستور کار آزمایشگاه سیستم های عامل


## آزمایشگاه سیستم های عامل

آرین محسنی

کاوه احمدی

آزمایش ۸ و ۹

آذر 1403

# دستور کار آزمایشگاه سیستم های عامل

در بخش اول این آزمایش میخواهیم ادامه دستور کار 8 که در آن الگوریتم cfs توضیح داده شده است را بررسی کنیم.

برای این کار از لینک داده شده ریپازیتوری را کلون میکنیم و سپس تغییرات گفته شده در فایل دستور کار را در بخش sfc_scheduler.c اعمال میکنیم. سپس با اجرای برنامه خروجی زیر را مشاهده میکنیم.

```
process array filled
tree created
inserted processes to rbtree
current_tick: 0
process 2000 is running current vruntime: 100   current residual_duration: 3
after running for one tick:     vruntime: 101   residual_duration: 2
current_tick: 1
process 2001 is running current vruntime: 101   current residual_duration: 3
after running for one tick:     vruntime: 102   residual_duration: 2
current_tick: 2
process 2000 is running current vruntime: 101   current residual_duration: 2
after running for one tick:     vruntime: 102   residual_duration: 1
current_tick: 3
process 2002 is running current vruntime: 102   current residual_duration: 3
after running for one tick:     vruntime: 103   residual_duration: 2
current_tick: 4
process 2001 is running current vruntime: 102   current residual_duration: 2
after running for one tick:     vruntime: 103   residual_duration: 1
current_tick: 5
process 2000 is running current vruntime: 102   current residual_duration: 1
after running for one tick:     vruntime: 103   residual_duration: 0
process 2000 terminated.
current_tick: 6
process 2002 is running current vruntime: 103   current residual_duration: 2
after running for one tick:     vruntime: 104   residual_duration: 1
current_tick: 7
process 2001 is running current vruntime: 103   current residual_duration: 1
after running for one tick:     vruntime: 104   residual_duration: 0
process 2001 terminated.
current_tick: 8
process 2002 is running current vruntime: 104   current residual_duration: 1
after running for one tick:     vruntime: 105   residual_duration: 0
process 2002 terminated.

real    0m0.003s
user    0m0.002s
sys     0m0.001s
arian@arian-VirtualBox:~/Desktop/University/OS/OS_Lab/Lab8/completely-fair-scheduler$
```

# دستور کار آزمایشگاه سیستم های عامل

## آزمایش جلسه 9)

در این بخش میخواهیم با سمافور آشنا شویم.

در سمافور دو تابع وجود دارد، sem_post و sem_wait که اولی نقش signal و دومی نقش wait را دارد که یعنی منابع را آزاد میکند و منابع را استفاده میکند.

```c
#include <pthread.h>

int shared_mem= 1;
sem_t s;

void* f1(){
    int x;
    sem_post(&s);
    sem_wait(&s);
    x= shared_mem;
    printf("f1, %d\n", x);
    x++;
    printf("f1 updated\n");
    sleep(1);
    shared_mem= x;
    printf("new val: %d\n", shared_mem);
    sem_post(&s);
}
void* f2(){
    int x;
    sem_wait(&s);
    x= shared_mem;
    printf("f2, %d\n", x);
    x--;
    printf("f2 updated\n");
    sleep(1);
    shared_mem= x;
    printf("new val: %d\n", shared_mem);
    sem_post(&s);
}

int main(){
    pthread_t t1, t2;
    pthread_create(&t1, NULL, f1, NULL);
    pthread_create(&t2, NULL, f1, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    printf("final: %d\n", shared_mem);

}
```

# دستور کار آزمایشگاه سیستم های عامل

در این کد ابتدا سمافور s را به تابع sem_post پاس میدهیم تا بتوانیم مقدار متغیر را زیاد کنیم وگرنه قفل میشود و برنامه متوقف میشود.

دو نخ ایجاد میکنیم و هرکدام را به یک تابع مقدار دهی میکنیم. در ابتدا سمافور روی تابع اولی زیاد میشود و در تابع دومی مقدارش کم میشود و قفل تابع دوم باز میشود تا بتواند ادامه روندش را طی کند. با چاپ کردن مقادیر تغییر یافته توسط دو تابع، خروجی مورد نظر به این شکل میباشد

```
f1, 1
f1 updated
new val f1: 2
f2, 2
f2 updated
new val f2: 1
final: 1
```