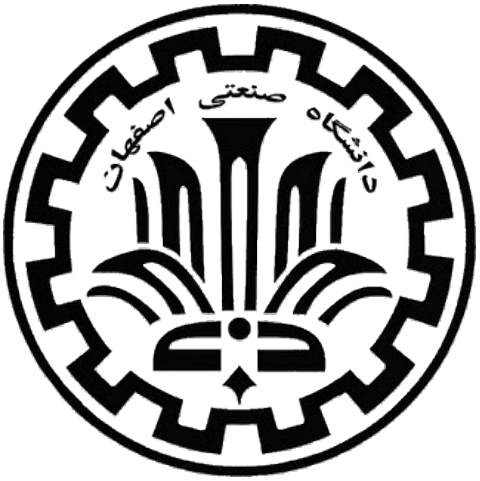

دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

آزمایشگاه سیستم عامل

مدیریت همزمانی در لینوکس

علی فانیان
زینب زالی

تابستان ۱۳۹۸

فراخوانی های سیستمی مدیریت semaphore

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

پس از تعریف سمافور از نوع sem_t، آن را با sem_init مقداردهی اولیه می‌کنند که مقدار موردنظر توسط آرگومان سوم این تابع مشخص می‌شود. اگر آرگومان دوم این تابع صفر باشد، نشان می‌دهد که سمافور بین تردهای یک پروسس استفاده خواهد شد و در صورتی که مقدار آن مثبت باشد، سمافور جهت کنترل همزمانی بین پروسس‌های مختلف قابل استفاده است. در هرحال سمافور باید جایی تعریف شده باشد که قابل دسترس از همه تردها یا همه پروسس‌هایی که به آن نیاز دارند باشد.

در برنامه‌نویسی multithread کافیست این سمافور به صورت global در پروسس والد تعریف شود. اما در برنامه‌نویسی multiprocess، باید به صورت shared_memory تعریف شود.

```
int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);
```

این دو تابع، جهت اخذ و آزادکردن سمافور قبل و بعد از ناحیه بحرانی استفاده می‌شوند.

```
int sem_getvalue(sem_t *sem, int *valp);
int sem_destroy(sem_t *sem);
```

در posix، این امکان وجود دارد که مقدار سمافور را با استفاده از تابع sem_getvalue بازیابی کرد. مقدار بازگشتی در valp در صورتی که سمافور آزاد نباشد، صفر و در غیر این صورت، تعداد نمونه‌های آزاد این سمافور را دربرخواهد داشت.
sem_destroy هم سمافوری که قبلاً init شده است از بین می‌برد. استفاده از سمافور destroyشده منجر به رفتار نامشخصی می‌شود.

مثال ها

Accessing variable “total”, avoiding multiple writes using semaphores

```
/*
- program creates 4 threads, assigns “routine1” as execution routine for each thread
- defines semaphore “sem1” in global space to be accessible by all threads
- each thread before entering its critical section, evaluates the value of “sem1”
- remark: sem_wait decrements semaphore /sem_post increments semaphore
*/
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define THREADS 4

sem_t sem1;
int total=0;

void *routine1(void * id )
{
	int idx=(int)id;
	sem_wait(&sem1);
	//beginning of critical section
	total+=1;
	printf("thread=%d and total=%d \n",idx,total);
	sleep(1);
	//end of critical section
	sem_post(&sem1);
	pthread_exit((void *)idx);
}

int main ()
{
	sem_init(&sem1,0,1);
	pthread_t threads[THREADS];

	for ( int i=0;i<THREADS;i++)
		pthread_create(&threads[i],NULL,routine1,(void *)i);


	for (int i=0; i<THREADS; i++)
		pthread_join(threads[i],NULL);

	return 0;
}
```

An Example of Busy-waiting

```
/*
- this program creates 4 threads and assigns “routine1” as execution routine for each thread
- threads will be synchronized by checking the value of variable “total”
- this method is called “busy-waiting”
*/
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define THREADS 4
#define SIZE 16

sem_t sem1;
int step=0;
int total=0;

void *func1(void * id )
{
   int idx=(int)id;
   //busy wait on step value
   while (step < SIZE)
   {
      while ( step%THREADS != idx );
      //beginning of critical section
      total+=1;
      printf("thread=%d and total=%d \n",idx,total);
      step++;
      sleep(1);
      //end of critical section
   }
   pthread_exit((void *)idx);

}

int main ()
{
   pthread_t threads[THREADS];

   for ( int i=0;i<THREADS;i++)
      pthread_create(&threads[i],NULL,func1,(void *)i);

   for (int i=0; i<THREADS; i++)
      pthread_join(threads[i],NULL);

   return 0;
}
```

این برنامه از سمافور استفاده نمی‌کند و با استفاده از busy waiting عملکرد سمافور را شبیه‌سازی می‌کند. هدف نهایی در این مثال، کنترل همزمانی روی مقدار total است، به صورتی که در هر لحظه فقط به یکی از نخ‌ها اجازه آپدیت این مقدار داده می‌شود. بدین منظور از متغیر سراسری step استفاده شده که نوبت هر نخ را مشخص می‌کند.

دستورکار جلسه هفتم

۱. برنامه ای بنویسید که:

- ضرب داخلی دو آرایه A و B هر یک به اندازه SIZE را محاسبه کند.
- برنامه به تعداد THREADS نخ خواهد داشت به طوریکه : THREADS <= SIZE.
- مقدار محاسبه شده در متغیری با نام product ذخیره خواهد شد.
- ممکن است چند نخ به صورت همزمان بر روی product بنویسند بنابراین از ساز و کاری استفاده کنید که مقدار product به درستی محاسبه شود.

۲. برنامه‌ای بنویسید که مسأله شام فیلسوفان را برای ۵ فیلسوف بدون بن‌بست به شرح زیر اجرا کند:

- ۵ نخ (به ازای هر فیلسوف یک نخ) به صورت همزمان اجرا شوند.
- هر فیلسوف دائماً یا در حال غذاخوردن و یا در حال فکرکردن است.
- اگر فیلسوفی هر دو چوب اطراف ظرفش آماده باشد، به اندازه زمانی تصادفی کمتر از ۳ ثانیه غذا می‌خورد.
- هر فیلسوف بعد از غذاخوردن چوبها را روی میز می‌گذارد و به اندازه زمانی تصادفی کمتر از ۳ ثانیه صبر می‌کند (فکر می‌کند) و سپس دوباره اقدام به غذاخوردن می‌کند.
- جهت جلوگیری از بن‌بست، هر فیلسوف فقط در صورتی شروع به غذاخوردن می‌کند که همزمان دو چوبش آزاد باشد (با کمک busy waiting و چک‌کردن مقدار سمافورها)

نکته: جهت ایجاد رخداد بن‌بست، برای هر فیلسوف، بعد از برداشتن چوب سمت راست، یک ثانیه صبر کنید و سپس چوب سمت چپ را بردارید.