دانشگاه گیلان
نیمسال سوم ۱۴۰۳-۱۴۰۲

بسمه تعالی

تمرینات سری سوم درس سیستم عامل

مهلت ارسل پاسخ: ۱۴۰۲/۱۰/۱۰

۱- معنای اصطلاح انتظار مشغول (Busy Waiting) چیست؟ چه نوع انتظار دیگری در یک سیستم عامل وجود دارد؟ آیا می توان به طور کلی از انتظار مشغول اجتناب کرد؟ پاسخ خود را توضیح دهید.

۲- چرا قفلهای چرخشی (Spin Locks) برای سیستم‌های تک پردازنده‌ای مناسب نیستند، اما اغلب در سیستم‌های چند پردازنده‌ای استفاده می‌شوند؟

۳- نشان دهید که چگونه می توان از یک سمافور باینری برای ایجاد انحصار متقابل در n فرآیند استفاده کرد.

۴- شبه کد شکل ۱ عملیات ()push و ()pop پشته مبتنی بر آرایه را نشان می دهد. با فرض اینکه این الگوریتم می تواند در یک محیط همزمان استفاده شود، به سوالات زیر پاسخ دهید:

الف) چه داده هایی دارای شرایط مسابقه هستند؟

ب) گونه می توان شرایط مسابقه را برطرف کرد؟

```
void bid(double amount) {
   if (amount > highestBid)
      highestBid = amount;
}
```

```
push(item) {
   if (top < SIZE) {
      stack[top] = item;
      top++;
   }
   else
      ERROR
}

pop() {
   if (!is_empty()) {
      top--;
      return stack[top];
   }
   else
      ERROR
}

is_empty() {
   if (top == 0)
      return true;
   else
      return false;
}
```

شگل ۱

۵- یک روش برای استفاده از ()compare_and_swap برای پیاده‌سازی spinlock به شرح زیر است:

```
void lock_spinlock(int *lock) {
   while (compare_and_swap(lock, 0, 1) != 0)
      ; /* spin */
}
```

یک رویکرد جایگزین پیشنهادی، استفاده از اصطلاح "مقایسه و مقایسه و تعویض" است که وضعیت قفل را قبل از فراخوانی عملیات compare_and_swap() بررسی می‌کند. (دلیل این رویکرد این است که فقط در صورتی که قفل در حال حاضر موجود باشد، تابع compare_and_swap() را فراخوانی کنیم.) این استراتژی در زیر نشان داده شده است:

```
void lock_spinlock(int *lock) {
{
   while (true) {
      if (*lock == 0) {
         /* lock appears to be available */

         if (!compare_and_swap(lock, 0, 1))
            break;
      }
   }
}
```

آیا این اصطلاح «مقایسه و مقایسه و تعویض» برای پیاده‌سازی قفل‌های چرخشی مناسب کار می‌کند؟ اگر چنین است توضیح دهید. اگر نه، نشان دهید که چگونه یکپارچگی قفل به خطر افتاده است.

۶- اولین راه حل صحیح نرم افزار شناخته شده برای مسئله بخش بحرانی برای n فرآیند با کران پایین در انتظار n-1 نوبت، توسط آیزنبرگ و مک گوایر (Eisenberg andMcGuire) ارائه شد. فرآیندها متغیرهای زیر را به اشتراک می گذارند:

```
enum pstate {idle, want_in, in_cs};
pstate flag[n];
int turn;
```

تمام عناصر flag در ابتدا بیکار هستند. مقدار اولیه متغیر turn بین ۰ و n-1 است. ساختار فرآیند P_i در شکل ۲ نشان داده شده است. ثابت کنید که الگوریتم هر سه شرط ناحیه بحرانی (انحصار متقابل، انتظار محدود و شرط پیشرفت) را برآورده می کند.

```
while (true) {
   while (true) {
      flag[i] = want_in;
      j = turn;

      while (j != i) {
         if (flag[j] != idle) {
            j = turn;
         else
            j = (j + 1) % n;
      }

      flag[i] = in_cs;
      j = 0;

      while ( (j < n) && (j == i || flag[j] != in_cs))
         j++;

      if ( (j >= n) && (turn == i || flag[turn] == idle))
         break;
   }

      /* critical section */

   j = (turn + 1) % n;

   while (flag[j] == idle)
      j = (j + 1) % n;

   turn = j;
   flag[i] = idle;

      /* remainder section */
}
```

شکل۲

۷- یک وب سرور چند نخی مایل است تعداد درخواست هایی را که به آن سرویس می دهد (که به عنوان بازدید شناخته می شود) پیگیری کند. دو استراتژی زیر را برای جلوگیری از شرایط مسابقه در نظر بگیرید. اولین استراتژی استفاده از قفل اصلی mutex هنگام به‌روزرسانی بازدیدها است:

```
int hits;
mutex_lock hit_lock;

hit_lock.acquire();
hits++;
hit_lock.release();
```

استراتژی دوم استفاده از یک عدد صحیح اتمی است:

```
atomic_t hits;
atomic_inc(&hits);
```

توضیح دهید که کدام یک از این دو استراتژی کارآمدتر است.

۸- فرض کنید val یک عدد صحیح اتمی در یک سیستم لینوکس است. مقدار val پس از تکمیل عملیات زیر چیست؟

```
atomic_set(&val,10);
atomic_sub(8,&val);
atomic_inc(&val);
atomic_inc(&val);
atomic_add(6,&val);
atomic_sub(3,&val);
```

۹- تصویر زیر را در یک لحظه (snapshot) از یک سیستم در نظر بگیرید:

	Allocation	*Max*	*Available*
	A B C D	*A B C D*	*A B C D*
T_0	0 0 1 2	0 0 1 2	1 5 2 0
T_1	1 0 0 0	1 7 5 0	
T_2	1 3 5 4	2 3 5 6	
T_3	0 6 3 2	0 6 5 2	
T_4	0 0 1 4	0 6 5 6	

با استفاده از الگوریتم بانکداری به سوالات زیر پاسخ دهید:

الف) ماتریس Need را بدست آورید.

ب) آیا سیستم در وضعیت امنی است؟ ثابت کنید.

ج) اگر درخواستی از نخ T1 برای (۰،۴،۲،۰) برسد، آیا باید به این درخواست پاسخ مثبت داد؟ ثابت کنید

۱۰- طرح انتظار چرخشی را با طرح های مختلف اجتناب از بن بست (مانند الگوریتم بانکدار) با توجه به مسائل زیر مقایسه کنید:

الف) سربار زمان اجرا

ب) توان عملیاتی سیستم

۱۱- سیستمی متشکل از چهار منبع یکسان را در نظر بگیرید که توسط سه نخ مشترک هستند که هر کدام حداکثر به دو منبع نیاز دارند. نشان دهید که سیستم بدون بن بست است.

۱۲- سیستمی متشکل از m منبع یکسان را در نظر بگیرید که توسط n نخ به اشتراک گذاشته شده است. یک نخ می تواند هر بار فقط یک منبع را درخواست یا منتشر کند. اگر دو شرط زیر وجود داشته باشد، نشان دهید که سیستم بدون بن بست است:

الف) حداکثر نیاز هر نخ بین یک منبع و m منبع است.

ب) مجموع تمام حداکثر نیازها کمتر از m + n است.

۱۳- تصویر زیر را در یک لحظه (snapshot) از یک سیستم در نظر بگیرید:

	Allocation	*Max*
	A B C D	*A B C D*
T_0	1 2 0 2	4 3 1 6
T_1	0 1 1 2	2 4 2 4
T_2	1 2 4 0	3 6 5 1
T_3	1 2 0 1	2 6 2 3
T_4	1 0 0 1	3 1 1 2

با استفاده از الگوریتم بانکدار، تعیین کنید که آیا هر یک از حالت های زیر ناامن است یا خیر. اگر حالت امن است، ترتیب تکمیل نخ ها را نشان دهید. در غیر این صورت، توضیح دهید که چرا حالت ناامن است.

a. ***Available*** = (2, 2, 2, 3)
b. ***Available*** = (4, 4, 1, 1)
c. ***Available*** = (3, 0, 1, 4)
d. ***Available*** = (1, 5, 2, 2)

۱۴- تصویر زیر را در یک لحظه (snapshot) از یک سیستم در نظر بگیرید:

	Allocation	*Max*	*Available*
	A B C D	*A B C D*	*A B C D*
T_0	3 1 4 1	6 4 7 3	2 2 2 4
T_1	2 1 0 2	4 2 3 2	
T_2	2 4 1 3	2 5 3 3	
T_3	4 1 1 0	6 3 3 2	
T_4	2 2 2 1	5 6 7 5	

با استفاده از الگوریتم بانکدار به سوالات زیر پاسخ دهید:

الف) با نشان دادن ترتیبی که در آن نخ ها ممکن است تکمیل شوند، نشان دهید که سیستم در وضعیت امن است.

ب اگر درخواستی از نخ T4 برای (۲، ۲، ۲، ۴) برسد، آیا می توان بلافاصله به درخواست پاسخ داد؟

ج اگر درخواستی از نخ T2 برای (۰، ۱، ۱، ۰) برسد، آیا می توان بلافاصله به درخواست پاسخ داد؟

د اگر درخواستی از نخ T3 برای (۲، ۲، ۱، ۲) برسد، آیا می توان بلافاصله به درخواست پاسخ داد؟

موفق باشید