به نام خدا

محمدمهدی آقاجانی

تمرین سوم سیستم عامل

استاد طاهری جوان

تمرین اول :

در این روش ارضای شرط انحصار متقابل وجود دارد زیرا تا وقتی که یک فرآیند در ناحیه بحرانی خودش قرار دارد flag خود را true نگه داشته بنابراین فرآیند دیگر در حلقه while گیر افتاده است و هنگامی که از ناحیه بحرانی خارج میشود flag خود را برابر false میکند و دیگری را آزاد مینماید.

شرط پیشرفت نیز ارضا میشود زیرا وقتی یک فرآیند قصد ورود به ناحیه بحرانی دارد دیگری در صورتی که در ناحیه بحرانی نباشد جلوی او را نمیگیرد.

شر انتظار محدود نیز برقرا است زیرا دستیابی به ناحیه بحرانی به صورت تصادفی نیست

اما این روش مشکل انتظار مشغول را دارد زیرا وقتی یکی از فرآیند ها در ناحیه بحرانی است دیگری بیکار است و سیکل های پردازنده را هدر میدهد.

تمرین دوم :

سمافور قوی به سمافوری گفته میشود که خروج از صف انتظار آن منوط به ترتیب ورود آن باشد اما سمافور ضعیف همچین ترتیبی را تضمین نمیکند . از مزایای سمافور قوی این است که عدم گرسنگی را تضمین میکنند اما از معایب سمافور ضعیف این است که میتوانند دچار گرسنگی شوند

تمرین سوم:

متغیر های شرطی cwait **و** csignal **است.**

Cwait : **چنانچه فرآیندی** cwait(x) **را صدا کند پشت شرط** x **به خواب میرود**

Csignal : **چنانچه فرآیندی** csignal(x) **را صدا بزند یک پیغام بیدار باش برای فرآیند هایی که پشت** x **به خواب رفته اند ارسال میکند و یکی از آنها برای احیا شدن انتخاب میشود.**

تمرین چهارم :

الف)

شرط انحصار متقابل را ارضا میکند زیرا در هنگامی که یکی در ناحیه بحرانی خود است فلگ خود را یک کرده است و دیگری قبل از ورود فلگ او را چک میکند.

شرط پیشرفت نیز ارضا میشود زیرا هنگام ورود دیگرانی که متقاضی ناحیه بحرانی نیستند مزاحم آن نمیشوند و میتواند به تناوب وارد ناحیه بحرانی بشود

شرط انتظار محدود را ارضا نمیکند زیرا اگر فرآیندی در ناحیه بحرانی باشد و زمانی کافی به ان بدهیم دوباره وارد ناحیه بحرانی میشود سپس در وسط ناحیه بحرانی پردازده را به دیگری بدهیم چون هنوز یک فرآیند در ناحیه بحرانی است پس مدام صبر میکند تا پردازنده از او گرفته شود و وقتی پردازنده را به فرآیند دیگر میدهیم و ناحیه بحرانی خود را به اتمام میرساند اگر وقت کافی داشته باشد دوباره میتواند به ناحیه بحرانی وارد شده و دیگری را قفل نگه دارد.

این روش مشکل بن بست نیز دارد یه این صورت که اگر خطوط را یکی در میان اجرا کنیم هیچ یک وارد ناحیه بحرانی نمیشوند

ب)

احصار متقابل را ارضا میکند زیرا اگر فرآیند ۱ در ناحیه بحرانی باشد قطعا فلگ مربوط به خود را ۱ کرده است و فرآیند ۲ همواره بر روی حلقه میماند تا مقدار فلگ ۱ برابر صفر شود

به وضوح شرط پیشرفت را نیز ارضا میکند

با توجه به رندم بودن تابع pause میتوان گفت که انتظار محدود ارضا میشود زیرا به صورت تصادفی احتمالا pause مقادیری به تاخیر میاندازد که گاهی قبل و گاهی بعد از آن پردازنده به فرآیند دیگر تخصیص می یابد.

ج)

انصار متقابل را ارضا میکند زیرا وقتی یک فرآیند در ناحیه بحرانی است دیگری در حلقه و پشت شرط دوم آن میماند

همچنین به علت وجود شرط اول حلقه شرط پیشرفت نیز ارضا میگردد.

انتظار محدود را نیز ارضا میکند زیرا با n۱=n۲+۱ باعث میشود فرآیندی که تازه در ناحیه بحرانی بوده پشت حلقه گیر کند و با سوییچ کردن بر روی فرآیند دیگری آن فرآیند منتظر از حلقه بیرون آید.

د)

مشکل انحصار متقابل دارد زیرا اگر در ابتدای کار یک فرآیند به حلقه بر بخورد چون دیگری false است از حلقه رد میشود و اگر در ابتدای حلقه پردازنده را از او بگیریم و به دیگری بدهیم چون هنوز فرآیند اول مهلت نکرده بود تا فلگ خود ار true کند پس دیگری هم از حلقه رد شده و هر دو با هم میتوانند وارد ناحیه بحرانی بشوند

شرط پیشرفت را به وضوح ارضا میکند

همچنین انتظار محدود را نیز ارضا نمیکند

ه)

انحصار متقابل را ارضا میکند با توجه به اینکه مقدار turn در ابتدا برابر یکی از فرآیندهاست پس دیگری در پشت حلقه درونی گرفتار میشود تا وقتی که دیگری از ناحیه بحرانی خارج شده و مقدار فلگ خود را برابر false کند

با توجه به اینکه در ابتدا مقدار فلگ برابر false است پس شرط پیشرفت را نیز ارضا میکند زیرا اگر turn برابر خودش بود که به راحتی وارد ناحیه بحرانی میشود و در غیر این صورت وارد حلقه میشود سپس در آنجا وارد پشت حلقه نمیماند و turn را برابر خود کرده و وارد ناحیه بحرانی میشود

شرط انتظار محدود را ارضا نمیکند زیرا به دیگر فرآیند های در حال انتظار توجه نمیکند.

و)

شرط انحصار متقابل را ارضا میکند بدین صورت که اگر یک فرایند در ناحیه بحرانی خود باشد دیگری یا در حلقه اول گیر میافتد و یا پشت حلقه دوم به خاطر متغیر turn گیر میافتد .

به وضوح پیشرفت را هم ارضا میکند

انتظار محدود را برآورده میکند زیرا اگر فرآیند صفر از ناحیه بحرانی بیرون بیاید مقدار turn را برابر یک میکند سپس اگر وقت کافی داشته باشد دوباره وارد ناحیه بحرانی میشود با این تفاوت که فرآیند دیگر از داخل حلقه درونی با true کردن فلگش خارج شده و در حلقه بیرونی گیر میافتد و فرآیند صفر این بار که از ناحیه بحرانی خود خارج شود وارد حلقه بیرونی شده و وارد if میشود و پشت حلقه درونی میماند تا دیگری وارد ناحیه بحرانی گردد.

ز)

انحصار متقابل را برآورده میکند زیرا وقتی یکی وارد ناحیه بحرانی مشود با WAIT کردن سمافور ها نمیگذارد دیگران وارد شوند

همچنین شرط پیشرفت نیز که به راحتی ارضا میشود زیرا میتواند سمافور ها را wait کرده و وارد نواحی بحرانی گردد

انتظار محدود را برآورده نمیکند زیرا اگر از ناحیه بحرانی خارج شود و بعد فرصت داشته باشد با وجود اینکه سیگنال میزند ولی چون هنوز ردازنده در اختیار دیگری قرار نگرفته میتواند ادامه دهد و دوباره wait زدن را شروع کرده و وارد ناحیه بحرانی شود.(اگر پردازش موازی وجود داشته باشد مشکل انتظار محدود هم حل میشود زیرا به محض signal زدن فرآیندی که روی wait خوابیده بیدار میشود)

این روش مشکل بن بست هم دارد زیرا اگر فرآیند صفر روی a عمل wait را انجام دهد و بعد فرآیند یک روی b عمل wait را انجام دهد سپس دوباره فرآیند صفر wait بر روی b بزند بر روی آن میخوابد و فرآیند یک هم بر روی a میماند و هر دو wait میشوند.

ح) همان سوال ز است.

تمرین پنجم :

الف)

Producer:

While(true){

Produce_item(item);

Put_buffer(item);

Signal(s);

}

Consumer:

While(true){

Wait(s);

Get_buffer(item);

Consume_item(item);

}

ب)

S2=0 , s1=1

Producer:

While(true){

Produce_item(item);

Wait(s1);

Put_buffer(item);

Signal(s2);

}

Consumer:

While(true){

Wait(s2);

```
Get_buffer(item);
Signal(s1);
Consume_item(item)
}
```

ج)

```
Writer
While(true){
Wait(mutex)
Wc++;
If(wc==1)
Wait(s);
Signal(mutex);
Wait(s2);
Write_to_DB;
Signal(s2);
Wait(mutex);
Wc--;
If(wc==0)
Signal(s)
Signal(mutex)
}

Reader
While(true){
Wait(s);
Read_from_DB;
Signal(s);
}
```

د)

```
Reader
While(true){
Wait(s)
Signal(s)
Wait(mutex)
Rc++;
If(rc == 1)
        Wait(s)
Signal(mutex);
Read_from_DB;
Wait(mutex2)
Rc—
If(rc==0)
        Signal(s)
Signal(mutex2)

Writer
While(true){
Wait(s)
Write_to_DB;
Signal(s);
}
```

تمرین ششم :

۱)

```
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

۲)

sz : میزان حافظه تخصیصی داده شده به فرآیند مورد نظر

state : وضعیت فرآیند از UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE

context : به این منظور استفاده می‌شود که رجیستر های ضروری برای فرآیند را ذخیره کند و هنگام switching این اطلاعات از بین میرود. context

ofile : فایل‌های باز شده را نگه میدارد

killed : اگر غیر صفر باشد فرآیند جاری از بین رفته است.