پرسش ۱

به این شکل است که هر فرآیندی که بخواهد وارد ناحیه بحرانی شود میبایست متغیر flag خود را true کند و چنانچه کس دیگری خواستار ورود به ناحیه بحرانی نباشد(flag فرآیندهای دیگر false باشد) وارد ناحیه بحرانی میشد و بعد از خروج از ناحیه بحرانی flag خود را false میکند. اما اگر به هنگام ابراز علاقه برای ورود به ناحیه بحرانی متوجه شود که Flag فرایند دیگر true است. اگر نوبت فرایند دیگر بود flag خود را false میکند چرا که ممکن است دفعه قبل خودش وارد ناحیه بحرانی شده باشد.(نوبت با متغیر turn سنجیده میشود) بنابراین در این روش هم علاقه مورد بررسی قرار میگیرد و هم نوبت. و نوبت تنها در حالتی بررسی میشود که فرایند دیگر هم خواستار وارد شدن به ناحیه بحرانی باشد.

mutual exclusion را شامل میشود , چون در آن واحد تنها یک فرایند هم دارای flag یک است و هم نوبت در دست خودش است بنابراین تنها یک فرایند میتواند وارد ناحیه بحرانی گردد.

شرط progress را دارد زیرا اگر فرایندی از ناحیه بحرانی خارج شود و دیگر نخواهد وارد شود مقدار flag آن false میشود و دیگری هر زمان که بخواهد می تواند وارد شود

اما شرط bounded waiting را ارضا میکند؛ چنانچه فرایندی از ناحیه بحرانی خارج شود. نوبت را به فرایند بعدی میدهد و چنانچه فرایند بعدی خواهان وارد شدن به ناحیه بحرانی را داشته باشد در این حالت فرایند اول نمیتواند مجددا وارد ناحیه بحرانی شود بلکه فرایند دوم وارد ناحیه بحرانی میشود.

پرسش ۲

Responsiveness : نکته این جاست که فرض کنیم که یک برنامه مثلا چند قسمت مهم داشته باشد (که یکی از آن‌ها رابط گرافیکی کاربر است) چنانچه برنامه وارد یکی از این قسمت‌ها شود قسمت‌های دیگر از کار می‌افتد (برای مثال رابط کاربری از کار می‌ایستد و دیگر کاربر با برنامه نمیتواند تعاملی داشته باشد) و این یک مسئله اشتباه است. ولی اگر همه این قسمت‌ها در نخی جداگانه کار کند. همه قسمت‌ها میتوانند کار خود را پیش ببرند حتی اگر یکی از این قسمت‌ها دچار بلاک شود.

Resource Sharing :استفاده مشترک از منابع توسط فرایندهای مختلف کار ساده‌ای نیست. اما نخ‌ها این کار را ساده کرده اند. زیرا نخ‌ها در واقع برای یک برنامه هستند و استفاده از منابع مشترک به همین علت برایشان ساده است.

Economy :برای استفاده از فرایندها میبایست به آن‌ها منابعی داد که هزینه بر است ولی با استفاده از نخ همانطور که در نکته قبلی گفته شد استفاده از منابع مشترک آسان است بنابراین هزینه‌ها کمتر میشود. (چون ساختن فرایند از نخ هزینه بیشتری داد)

Scalability : چندنخی وقتی خیلی خودش را نشان میدهد که با پردازنده‌های چند هسته‌ای کار کند. چون هر کدام از نخ ها میتوانند روی چند هسته کار کنند و سرعت برنامه به طور افزاینده‌ای زیاد شود.

پرسش ۷

asynchronous : در این حالت نخ فرزند و والد به صورت همزمان اجرا میشود و منابع مشترک کمی میتوانند با هم داشته باشند. (بیشتر برای سرور ها استفاده میشود.)

Synchronous : در این حالت پدر ممکن است چند فرزند داشته باشد و فرزندها میتوانند به صورت موازی کار کنند و پدر منتظر اتمام کارشان میماند.

پرسش ۹

هر دو روش برای منابع مشترک از متغیر جهانی استفاده میکند.

هر دو روش نخ ها که میسازیم هر کدام یک تابع را اجرا میکنند.

در هر دو روش تابع ساخت نخ ویژگی های نخ را از طریق آرگومان ورودی میگیرد.

لینوکس , مک از pthread پشتیبانی می کنند ولی windows , thread مخصوص به خود را دارد.

در ویندوز نخ با پوینتر برای صدا زدن تابع استفاده میکند. ولی pthread از اسم تابع ها برای کال کردن تابع ها استفاده میکند.

چنانچه بخواهیم نخ والد منتظر فرزندش باشد در pthreads از تابع pthread_join استفاده میکنیم. و در ویندوز از WaitForSingleObject() بهره میبریم که هر دو یک کار را میکنند.

پرسش ۱۰

مشکلات از این شرحند که اگرچه هزینه تولید نخ از تولید فرایند کمتر است اما همچنان داره هزینه است. و ممکن است با تولید زیادی نخ شاهد افزایش سربار و مشاهده کندی در سیستم باشیم. ضمنا به علت وجود منابع مشترک میبایستی حواسمان به تغییر متغیرها به صورت ناخواسته باشد. (نخ ها بعد از انجام وظیفه خود پاک میشوند.)

روش رفع این مشکل متدی به نام thread pool است. این متد به این شکل است که تعداد مشخصی نخ میتوانند کار کنند و چنانچه تسک جدیدی بیاید و همه نخ ها مشغول باشند میبایستی منتظر بماند (درون یک صف) تا فعالیت این نخ ها تمام شود و نخ جدید تولید گردد و این task را بر عهده بگیرد. البته اگر بخواهیم این متد را بهتر پیاده سازی کنیم بهتر است نخ قبلی را از بین نبریم و task جدید به همان بدهیم تا هزینه سربار تولید نخ جدید و از بین بردن نخ قبلی را نداشته باشیم.

پرسش ۱۱

برای استفاده از چند نخی استانداردی توسط IEEE ایجاد شده است که سیستم های UNIX از آن بهره میبرند. و به واسطه آن توابعی برای استفاده از نخ ها ایجاد شده است.

توابع مربوط به این استاندارد :

- Pthread_create : وظیفه آن تولید نخ جدید، و مقدار return شده آن Thread Identifier است.
- Pthread_exit : برای اتمام یک نخ استفاده میگردد.
- Pthread_join : این تابع به عنوان ورودی نخ دیگری میگرد و نخی که تابع برای آن صدا زده شده است منتظر پایان کار نخ ورودی میماند.
- Pthread_yeild : وقتی نخی بخواهد به نخ دیگری زمان بدهد از این تابع استفاده میشود (علت این موضوع این است که همه نخ ها برای یک برنامه هستند و مثل فرآیندها خودخواهانه عمل نمیکنند.)
- Pthread_attr_init : برای پر کردن ویژگی‌های یک نخ صدا زده میشود و مقادیر اولیه را به صورت پیشفرض پر میکند.
- Pthread_attr_destroy : برای پاک کردن ویژگی های یک نخ استفاده میشود (نخ از بین نمیرود)

_____ پرسش ۱۴ _____

الف)

شرط mutual exclusion را برآورده می کند , زیرا که یک فرایند وقتی وارد ناحیه بحرانی میشود که دیگری مقدار flag خود را صفر کرده باشد. وجود دو شرط کنترلی این شرط را براورده میکند.

شرط progress را نیز دارد زیرا اگر مثلا فرایند دوم نخواهد وارد قسمت بحرانی شود. و flag این فرایند صفر خواهد بود. و به همین دلیل فرایند اول هر وقت بخواهد با تغییر مقدار flag به یک، میتواند وارد ناحیه بحرانی شود.

شرط bounded waiting را ندارد زیرا این امکان وجود ندارد که فرایند بداند الان نوبت خودش است یا دیگری. یعنی اگه پردازنده دست یک فرایند بماند میتواند بدون این که فرایند دیگر وارد ناحیه بحرانی شود بارها وارد ناحیه بحرانی‌اش بشود.

ب)

شرط mutual exclusion را برآورده می کند چون تازمانی که فرایند دیگر مقدار flag را یک نگه داشته است نمیتواند وارد ناحیه بحرانی شود و هر وقت یکی flag را یک کرده، دیگری میبایست flag خود را به صفر تغییر دهد؛ به همین دلیل دو فرایند نمیتوانند وارد ناحیه بحرانی شوند.

شرط progress را دارد چنانچه فرایندی نخواهد وارد ناحیه بحرانی شود با صفر کردن flag این اجازه را به دیگر فرایند ها میدهد که هر هنگام خواستند وارد ناحیه بحرانی شوند.

شرط bounded waiting را ندارد و فرایند از شرایط فرایند دیگر با خبر نیست. به همین دلیل چنانچه پردازنده را در دست داشته باشد. میتواند بارها وارد ناحیه بحرانی شود.

ج)

شرط mutual exclusion را دارد زیرا هر گاه یک فرایند بخواهد وارد ناحیه بحرانی گردد؛ مقدار خود را یکی بیشتر از مقدار فرایند دیگر قرار میدهد.و فقط وقتی وارد میشود که مقدار فرایندیگر صفر شده باشد و یا مقدار flag خودش از بقیه کوچکتر باشد. و به همین دلیل کسی نمیتواند وارد شود زیرا یا Flag بزرگتری دارد یا مقدارش صفر نیست.

شرط progress را ندارد فرض کنیم یک فرایند وارد ناحیه بحرانی شود. بنابراین مقدار غیرصفری خواهد داشت چنانچه فرایند دیگری بخواید وارد ناحیه بحرانی شود میبایست که مقدار خود را بیشتر بگذارد. حال چون مقدارش از دیگری کوچکتر نیست و مقدار دیگر غیر صفر است نمیتواند وارد ناحیه بحرانی شود.

شرط bounded waiting را دارد زیرا که اگر فرایندی بخواهد وارد شود بایستی مقدار خود را بیشتر از دیکری قرار دهد و اگر فرایندی از قبل قصد ورود داشته، پس مقدارش صفر نبوده پس فرایند جدید نمیتواند وارد شود و فرایندی که مدت زمان بیشتری انتظار کشیده است میتواند وارد شود.

د)

شرط Mutual Exclusion برقرار نیست. فرض کنید فرایند اول مقدار فلگ دیگری را چک میکند و چونfalse است وارد ناحیه بحرانی میشود قبل از true شدن فلگ آن context switch رخ میدهد و دیگری نیز وارد ناحیه ی بحرانی میشود.

شرط Progress برقرار است. در صورتی که فرایند دیگر نخواهد وارد ناحیه ی بحرانی شود مقدار flag آن false است بنابراین فرایند اول میتوانند هرچقدر که بخواهد وارد ناحیه بحرانی خود شود.

شرط Bounded Waiting برقرار نیست. چون توجهی به نوبت فرایند ها نمیشوند فرایند اول وارد میشود و خارج میشود چون چیزی تغییر نمیکند دوباره میتواند وارد شود حتی اگر فرایند دیگر مدت ها منتظر بوده باشد.

ه)

شرط Mutual Exclusion برقرار نیست. هرگاه فرایندی بخواهد وارد شود مقدار فلگ خود را true میکند و اگر نوبتش باشد وارد ناحیه ی بحرانی میشود حال context-switch رخ میدهد و فرایند دوم میخواهد وارد ناحیه ی بحرانیش شود مقدار فلگش را true میکند ولی چون نوبتش نیست وارد while میشود و در داخل منتظر میماند تا فرایند اول از ناحیه ی بحرانی خارج شود حال فرایند اول از ناحیه ی بحرانی خارج میشود بنابراین فرایند دوم از while خارج میشود اما هنوز turn را تغییر نداده دوباره context-switch اتفاق میفتد و چون هنوز نوبت فرایند اول است فرایند اول وارد ناحیه بحرانی میشود در این بین فرایند دوم ادامه میدهد چون ازwhile خارج شده turn را تغییر میدهد از while بیرونی نیز خارج میشود و وارد ناحیه بحرانی میشود حال هر دو در ناحیه ی بحرانی قرار دارند.

شرط Progress برقرار است. زیرا وقتی یکی از دوفرآیند نخواهد وارد ناحیه بحرانی شود Flag خود را false قرار میدهد و بنابراین چنانچه دیگری بخواهد وارد شود مقدار flag را یک میکند و حتی اگر turn برابر با i نباشد بعد بعد وارد حلقه شدن مقدار turn تغییر میکند و نوبت به همان فرایند میرسد. و وارد ناحیه بحرانی میشود. از آن جا که نوبت در دست خودش است از این به بعد هربار بخواهد وارد ناحیه بحرانی خودش شود میتواند این کار را بکند.

شرط Bounded Waiting برقرار نیست. فرض کنید فرایند صفر در ناحیه ی بحرانی است و فرایند ۱ میخواهد وارد ناحیه ی بحرانی اش شود بنابراین فرایند ۱ در خط ;(flag[1-i]) while منتظر میماند تا فرایند صفر از ناحیه بحرانی خارج شود زمانی که فرایند صفر از ناحیه ی بحرانی خارج می شود اگر cpu به فرایند ۱ داده شود این فرایند وارد ناحیه ی بحرانی خود میشود)تا اینجا درست(اما اگر cpu به فرایند صفر داده شود این فرایند میتواند بدون توجه به فرایند دیگر دوباره وارد ناحیه بحرانی خود شود بنابراین این شرط برقرار نیست.

و)

شرط Mutual Exclusion :فرض کنید فرایند A بخواهد وارد شود اگر فرایند دیگری نخواهد وارد شود بدون مشکل وارد میشود و چون از قبل مقدار flag را true کرده فرایند دیگری نمیتواند وارد شود.

شرط Progress :اگر فرایندی از ناحیه بحرانی خارج شود و دیگر نخواهد وارد شود مقدار flag آن false میشود و دیگری هر زمان که بخواهد می تواند وارد شود.

شرط Bounded Waiting : هرگاه فرایندی از ناحیه ی بحرانی خود خارج شود نوبت را به دیگری میدهد بنابراین اگر دیگری بخواهد وارد شود فرایند فعلی نمیتواند دوباره وارد شود.

ز)

شرط Mutual Exclusion :برقرار است. هرگاه فرایندی وارد ناحیه ی بحرانی شود یعنی مقدار هر دو سمافور ۱ بوده اند و آن ها را ۰ کرده است بنابراین فرایند دیگر نمیتواند وارد شود و زمانی که خارج میشود مقدار هر دو سمافور را یک میکند.

شرط Progress :زمانی که یکی از ناحیه بحرانی خارج شود و دیگر نخواهد وارد شود مقدار سمافور ها را ۱ میکند و دیگری میتواند هرچند بار که بخواهد وارد شود.

شرط Bounded Waiting : این شرط ارضا می شود زیرا، اگر فرایندی بخواهد وارد ناحیه ی بحرانی اش شود ولی فرایند دیگری در حال حاضر در ناحیه ی بحرانی باشد در دستور wait بلاک میشود و در صف مربوط به فرایند های بلاک شده در آن سمافور قرار میگیرد به محض اینکه دستور سیگنالی روی آن سمافور اجرا شد.)زمان خروج فرایند دیگر از ناحیه ی بحرانی(فرایند فعلی از بلاک در می آید و وارد ناحیه ی بحرانی میشود بنابراین امکان ندارد فرایندی بتواند بدون توجه به دیگران مرتبا وارد ناحیه ی بحرانی اش شود. اما ممکن است دچار ددلاک شوند فرایند اول سمافور A را صفر کند و فرایند دوم سمافور B را در این شرایط هر دو فرایند پشت ناحیه بحرانی خواهند ماند.