آریا خلیق
۹۵۲۴۰۱۴
اواخر بهار ۹۸
تمرین شماره ۸ سیستم‌عامل!

۱)

الف)

بله بن‌بست رخ داده زیرا هر کدام از ماشین‌ها منتظر منبعی(ادامه خیابان) ای هستند که هر کدام در دست دیگری است و در گره گیر کرده‌اند.

ب)

شرط اول: صدق می‌کند زیرا در هر لحظه تنها یک ماشین می‌تواند از تقاطع خیابان به جلو حرکت کند.

شرط دوم: صدق می‌کند زیرا هر ماشین یک بخش از خیابان را در اختیار خود گرفته و بخش دیگری را می‌خواهد و آن را پس نمی‌دهد.

شرط سوم: صدق می‌کند زیرا یک فرایند نمی‌تواند یک منبع را داوطلبانه پس دهد و به زور هم نمی‌توان گرفت.

شرط چهارم: این شرط هم صادق است. یک حلقه رخ داده(در بالا توضیح داده شد)

ج)

از چراغ راهنما قرمز در تقاطع استفاده می‌کنیم با این فرض اضافه که اگر ماشینی وارد وسط تقاطع شود و به دلیل این که یک ماشین دیگر جلوی آن قرار دارد در وسط چهارراه قرار گیرد به هیچ عنوان وارد نشود که تقاطع فقط برای ثانیه‌هایی مشغول باشد و هیچ دو ماشینی هم‌زمان خواستار ورود به بخش وسطی تقاطع نباشند و این‌گونه چند ماشین درخواست یک منبع تقاطع را نخواهند کرد.

پیاده‌سازی آن بسیار راحت است(همان طور که در دنیای واقعی شده).

بهره‌وری آن نیز بالا می‌باشد در صورتی که زمان‌های چراغ قرمز به صورت هوشمند و بر اساس میزان ماشین‌های منتظر مشخص شود. اگر حالت عادی باشد حداکثر و در بدترین حالت، بازدهی نصف خواهد شد ولی هرگز بن‌بست نخواهیم داشت.

۲)
الف)
قسمت ۱)
نکته بسیار مهم: فرض کردم پردازنده زمانی کارایی ندارد که مشغول نیست(منتظر انجام کار یک resource است) و یا دارد فعالیتی غیر از پیش‌برد فرایندها انجام می‌دهد(مثلا انجام محاسبات برای این که آیا بن‌بست رخ می‌دهد یا نه، یا اینکه در الگوریتم بانک‌دار مشغول حساب کردن امن بودن است)

ترتیب از بهترین به بدترین:
رزرو تمامی منابع قبل از شروع ← بسیار خوب می‌باشد ولی چون تعداد کم‌تری فرایند وارد می‌شوند ممکن است همه آن‌ها به حالت wait بروند و CPU میزان بسیار زیادی بی‌کار باشد.

الگوریتم بانک‌دار ← محاسبات نسبتاً سنگینی دارد ولی در عوض بن‌بست رخ نداده و فرایندها عموما در حال فعالیت هستند.

شروع مجدد فرآیند (و آزاد کردن منابع) در صورتی که فرآیند منتظر یک منبع است. ← بسیار بد زیرا ممکن است قبل از رخ‌دادن بن‌بست فرایند را restart کنیم و دوباره فرآیند را اجرا کنیم که کار تکراری است. در عوض محاسبات سنگینی نداریم.

دادن شماره ترتیبی به منابع و ... ← به دلیل این روش ممکن است فرآیندها wait شوند و CPU بی‌کار بماند ولی در عوض محاسبات آن کم‌تر است.

کشف بن‌بست و از بین بردن کامل فرایندهای دخیل ← کارایی بسیار پایین چون در حال محاسبات بسیار سنگین پیدا کردن بن‌بست است.

کشف بن‌بست و به عقب برگرداندن فرآیندها (فرض شده منظور همه فرایندها باشند) ← مانند حالت بالا است فقط تعداد بیش‌تری فرایند به عقب برگردانده می‌شوند و دوباره ممکن است همان حالت رخ دهد.

قسمت ۲)
ترتیب از بهترین به بدترین:

کشف بن‌بست و از بین بردن کامل فرایندهای دخیل ← بسیار عالی می‌باشد هر چند باعث دوباره کاری می‌شود ولی تعداد زیادی فرآیند با هم در حال فعالیت هستند.

کشف بن‌بست و به عقب برگرداندن فرآیندها (فرض شده منظور همه فرایندها باشند) ← مانند بالایی است ولی تعداد بیشتری wait می‌شوند به همین دلیل چند پردازندگی کم‌تری دارد.

شروع مجدد فرآیند (و آزاد کردن منابع) در صورتی که فرآیند منتظر یک منبع است. ← تعداد زیادی فرایند در حال فعالیت هم‌زمان هستند(هر چند دوباره کاری) و از این نظر بسیار خوب است.

الگوریتم بانک‌دار ← یک safe zone برای خود نگه می‌دارد به همین دلیل باعث می‌شود فرایندهای کم‌تری پیش‌روی کنند(منتظر منبع هستند) و فرایندهای کم‌تری نیز وارد شوند.

دادن شماره ترتیبی به منابع و ... ← باعث می‌شود تعداد زیادی فرآیند منتظر دادن منبع شوند و به همین دلیل پیشرفت هم‌زمان نداشته باشند.

رزرو تمامی منابع قبل از شروع ← باعث می‌شود کم‌ترین میزان فرایندها وارد شوند پس کم‌ترین تعداد برنامه هم‌زمان را دارد.

ب)

خیر من در نظر نگرفتم. علت این بود که کارآیی پردازنده را میزان مشغول بودن پردازنده به فعالیت کردم و دوباره کاری پردازنده را به عنوان نکته منفی در کارآیی پردازنده در نظر نگرفتم. اجرای موازی هم هم‌چنین.

اگر در نظر می‌گرفتم باید با یک ضریبی این رتبه بندی تغییر می‌کرد و برای مثال بانک‌دار هیچ‌گاه به بن‌بست نمی‌خورد و اول می‌شود.

۳)

تحلیل از دیدگاه بن‌بست:

تحت شرایطی که توضیح داده شده ندارد.
شرط کاف‌من به دادمان می‌رسد. شرط Hold & Wait را نقض می‌کند یعنی اگر چنگال سمت راست را بردارد و چنگال سمت چپ موجود نباشد منبع قدیمی خود(سمت راست) را هم رها می‌کند.

نکته بسیار مهم: در کتاب در مورد است حالت خاص بحث شده که همه فیلسوف‌ها اول سمت راست را بردارند و چک کنند و چون همه سمت چپ آن‌ها مشغول است دوباره همه چنگال‌ها را زمین بگذارند و این چرخه بارها و بارها ادامه پیدا کند در این صورت live lock رخ می‌دهد که با dead lock متفاوت است. اگر فرض کنیم در چک کردن سمت چپ و گذاشتن آن روی میز atomic باشد که البته منطقی هم هست این حالت دیگر رخ نخواهد داد.

تحلیل از دیدگاه گرسنگی:

دارد.
بیایید فیلسوف شماره ۳ را از گرسنگی بکشیم. فرض کنیم ۴ سمت راست ۳ و ۲ سمت چپ آن است. فرض می‌کنیم ۲ در حال خوردن است و ۴ هم در حال خورد است. فیلسوف ۳ قصه ما چپ و راست خود را چک می‌کند که در آن لحظه هر دو در حال خوردن هستند و اصلاً نمی‌تواند سمت راست را بردارد. ۲ و ۴ از خوردن بازمی‌ایستند، فکر می‌کنند و دوباره شروع به خوردن می‌کنند و CPU به دست ۳ می‌رسد و دوباره چیزی بر نمی‌دارد و این قصه تا ابد ادامه پیدا می‌کند. علت آن این است که هیچ فیلسوفی تکنیکی برای رزرو کردن چنگال‌ها ندارد و stateless است.

۴)

منطقاً نمی‌تواند تشخیص دهد چون اگر ۱۰۰۰ سال بعد ۱۰ فرایند را چک کنیم و هنوز پیشرفت نکرده باشند ممکن است همه آن‌ها یک ثانیه بعد تمام شوند(بر اثر اتفاقی برای یک فرایند دیگر) به همین دلیل اگر اطلاعاتی از آینده نداشته باشیم این کار ممکن نیست.

برای رفع آن، اگر برای ما رفع گرسنگی حیاتی باشد و بتوانیم الگوریتم آن را بنویسیم از الگوریتم‌های مناسب استفاده کنیم که بن‌بست گرسنگی هیچ‌گاه رخ ندهد. اگر این الگوریتم‌ها را نداشته باشیم و یا به هر دلیلی از آن‌ها استفاده نکنیم، بهتر است بعد از یک مدت زمان مشخص و طبق نیازمندی کاربر برنامه را kill و از اول شروع کنیم. مهم است که برخی از فرایندها با در اختیار گرفتن پردازنده هم حتی پیشرفت نمی‌کنند (مثلاً حالت گرسنگی دار مساله فیلسوفان) به همین دلیل عملاً مجبور به kill کردن آن‌ها یا صبر بیش‌تر می‌شویم.

۵)

بله. بانک‌های کم‌تر ریسک‌کن از همین ایده استفاده می‌کنند. اصلاً ایده استفاده از اسکناس به جای مواد ارزشمند در حالتی که به اسکناس ارزش بیش‌تری از این مواد بدهیم از این الگوریتم تبعیت می‌کند.

در بانک‌های امروزی اگر همه افراد دارایی خود را در یک لحظه طلب کنند احتمالاً بانک‌ها ورشکسته می‌شوند چون نسبت به پول نقد خود از این الگوریتم استفاده نکرده‌اند.

مثال روزمره‌ای که در زندگی خود استفاده می‌کنیم قرض دادن درآمد حساب بانکی‌مان به یک دوست باشد که قرار است یک‌ماه دیگر پول را برگرداند و در بدترین شرایط ممکن هزینه‌های زندگی خود را حساب می‌کنیم که اگر بعد از دادن پول از پس آن‌ها تا حقوق بعدی برآمدیم پول را به دوست خوبمان بدهیم.