گزارشِ تحقیق درباره‌ی روش‌های مختلفِ کنترلِ همروندی در سیستم‌عامل - مقدمه

تهیه و تنظیم: مبین خیری

شماره دانشجویی: 994421017

استاد راهنما: دکتر لیلا شریفی

کنترل هم‌روندی (Concurrency control) چیست و چگونه پیاده‌سازی می‌شود؟

سیستم‌های کامپیوتری، چه نرم‌افزاری و چه سخت‌افزاری، شامل ماژول‌ها یا مؤلفه‌هایی هستند. هر مؤلفه به گونه‌ای طراحی شده که به‌طرز صحیحی عمل کند؛ یعنی، از قوانین سازگاری به‌خصوصی تبعیت نموده یا با آن‌ها تطابق داشته باشد. هنگامی که مؤلفه‌های هم‌روند با یک‌دیگر به وسیله پیام‌دهی یا اشتراک‌گذاری داده‌های دسترسی شده برهم‌کنش کنند (در حافظه یا محل ذخیره‌سازی)، هم‌روندی برخی از مؤلفه‌ها ممکن است توسط مؤلفه‌های دیگری مورد تخطی قرار می‌گیرند. هم‌روندی به ارائه قواعد، روش‌ها، روش‌شناسی‌های طراحی و نظریاتی می‌پردازد که به حفظ سازگاری مؤلفه‌های هم‌روندی، طی اجرای عملیات کمک کند تا سازگاری و صحت کل سیستم حفظ شود.

معرفی کنترل هم‌روندی (Concurrency control) به یک سامانه، به معنای اعمال محدودیت‌های عملیاتی است که اغلب منجر به کاهش برخی از کارایی‌های سامانه می‌گردد. دستیابی به سازگاری عملیاتی و صحت باید تا حد امکان همراه با بهره‌وری باشد، به گونه‌ای که کارایی از حد معقول خاصی کمتر نشود. ممکن است کنترل هم‌روندی در مقایسه با الگوریتم‌های سری (Sequential Algorithms) که ساده‌تر هستند نیازمند افزودن به پیچیدگی و سربار در قالب یک الگوریتم هم‌روندی باشد. به عنوان مثال، شکست در کنترل هم‌روندی، ممکن است منجر به خراب‌شدن داده‌ها شود که ناشی از گسست عملیات خواندن یا نوشتن است. در سیستم‌ها چند وظیفه همزمان انجام می‌شوند. اگر این وظایف مستقل از هم باشند اجرای آن‌ها ساده است اما درصورتی‌که درگیر باشند، مثلاً نوشتن هم‌روند بر روی یک فایل، برای انجام درست وظایف نیاز به کنترل هم‌روندی است وگرنه ممکن است منجر به نتایج ناخواسته شوند. در همین ارتباط ،مفهومی به‌نام سمافور توسط متخصصان طراحی شده است. سمافور (Semaphore) به متغیری گفته می‌شود که در محیط‌های هم‌روند برای کنترل دسترسی فرایندها به منابع مشترک به‌کار می‌رود. سمافور می‌تواند به دو صورت دودویی (که تنها دو مقدار صحیح و غلط را دارا است) یا شمارنده اعداد صحیح باشد. از سمافور برای جلوگیری از ایجاد وضعیت رقابتی میان فرایندها استفاده می‌گردد. به این ترتیب، اطمینان حاصل می‌شود که در هر لحظه تنها یک فرایند به منبع مشترک دسترسی دارد و می‌تواند از آن بخواند یا بنویسد.

سمافورها اولین بار به‌وسیله دانشمند علوم رایانه هلندی، ادسخر دیکسترا معرفی شدند و امروزه به‌طور گسترده‌ای در سیستم‌عامل‌ها مورد استفاده قرار می‌گیرند.

اصل اساسی این است که دو یا چند فرایند می‌توانند به وسیله سیگنال‌های ساده با یک‌دیگر همکاری کنند. هر فرایند را می‌توان در نقطه خاصی از اجرا متوقف نموده و تا رسیدن سیگنال خاصی از اجرای آن جلوگیری نمود. برای ایجاد این اثر، از متغیرهای خاصی به‌نام سمافور استفاده می‌گردد. هر فرایندی که بخواهد به منبع مشترک دسترسی داشته باشد، اعمال زیر را انجام خواهد داد:

1. مقدار سمافور را بررسی می‌کند.
2. در صورتی که مقدار سمافور مثبت باشد، فرایند می‌تواند از منبع مشترک استفاده کند. در این صورت، فرایند یک واحد از سمافور می‌کاهد تا نشان دهد که یک واحد از منبع مشترک را استفاده کرده است.
3. در صورتی که مقدار سمافور صفر یا کوچک‌تر از صفر باشد، فرایند به خواب می‌رود تا زمانی که سمافور مقداری مثبت به خود بگیرد. در این حالت فرایند از خواب بیدار شده و از مرحله یک شروع می‌کند.
4. هنگامی که فرایند کار خود را با منبع تمام کرد، یک واحد به سمافور اضافه می‌گردد .هر زمان که مقدار سمافور به صفر یا بیشتر برسد، یکی از فرایند(هایی) که به خواب رفته به صورت تصادفی یا به روش FIFO توسط سیستم‌عامل بیدار می‌شود. در این حالت، بلافاصله فرایند بیدار شده، منبع را در دست می‌گیرد و مجدداً پس از اتمام کار یک واحد از سمافور کم می‌شود. اگر مقدار سمافوری صفر باشد و چند فرایند بلوکه شده در آن وجود داشته باشد، با افزایش یک واحدی سمافور، مقدار سمافور همچنان صفر باقی می‌ماند اما یکی از فرایندهای بلوکه شده، آزاد می‌شود.

کنترل هم‌روندی برچسب زمان

کنترل هم‌روندی برچسب زمان (Timestamp-based concurrency control) در علوم کامپیوتر یکی از الگوریتم‌های کنترل هم‌روندی بدون قفل است که برای کنترل هم‌روندی تراکنش‌ها در برخی از پایگاه‌های داده، توسط زدن برچسب زمان استفاده می‌شود. این الگوریتم عدم وجود بن‌بست را تضمین می‌کند.

کنترل هم‌روندی چندنسخه‌ای

کنترل هم‌روندی چندنسخه‌ای (Multiversion Control Concurrency) در زمینه پایگاه داده علوم رایانه، یک روش کنترل هم‌روندی است که معمولاً توسط سامانه‌های مدیریت پایگاه داده برای ارائه دسترسی هم‌روند به پایگاه داده استفاده می‌شود. همچنین در زبان‌های برنامه‌نویسی برای پیاده‌سازی حافظه تراکنشی به کار می‌رود.

بدون کنترل هم‌روندی، اگر کسی در حال خواندن از یک پایگاه داده باشد و همزمان شخص دیگری در آن بنویسد، ممکن است خواننده یک قطعه داده‌ای که کامل نوشته نشده یا متناقض است را ببیند. به‌طور مثال، هنگام انتقال داده بین دو حساب بانکی اگر خواننده، زمانی که پول از حساب اصلی حذف شده و قبل از ذخیره شدن در حساب مقصد، میانگین حساب را از بانك بخواند، به نظر می‌رسد که پول در بانك

ناپدید شده است. انزوا (isolation) یک ویژگی است که دسترسی‌های هم‌روند به داده را تضمین می‌کند. انزوا با استفاده از معنا و مفهوم پروتکل‌های کنترل هم‌روندی پیاده‌سازی می‌شود. ساده‌ترین راه این است که همه خوانندگان منتظر بمانند تا نوشتن انجام شود که به عنوان یک قفل خواندن - نوشتن شناخته می‌شود. قفل‌ها باعث ایجاد درگیری می‌شوند، به‌خصوص بین تراکنش‌های خواندن طولانی و تراکنش‌های به‌روزرساني. هدف MVCC، حل مشکل با نگه داشتن چندین نسخه از هر یک از داده‌ها است. بدین ترتیب، هر کاربری که به پایگاه داده متصل است، یک تصویر (snapshot) از پایگاه داده را در یک لحظه خاص در زمان می‌گیرد. هر گونه تغییری که توسط یک نویسنده ایجاد شده است، توسط سایر کاربران پایگاه داده تا زمانی که تغییرات تکمیل نشده باشد (یا در شرایط پایگاه داده، تا زمانی که تراکنش کامل شود) مشاهده نمی‌شود.

هنگامی که یک پایگاه داده MVCC یک قطعه از داده را به‌روزرسانی می‌کند، داده اصلي را با داده جدید جایگزین نخواهد کرد بلکه یک نسخه جدید از آن داده ایجاد می’کند. بنابراین چندین نسخه از داده ذخیره می‌شود. نسخه‌ای که هر تراکنش مشاهده می‌کند به سطح انزوای اجرا شده (isolation level) بستگی دارد. شایع‌ترین سطح جداسازی با MVCC، جداسازی فوری (snapshot isolation) است. با سطح جداسازی فوری، یک تراکنش وضعیت داده را به‌عنوان زمان انجام تراکنش مشاهده می‌کند.
MVCC چالش چگونگی حذف نسخه‌هایی را که منسوخ شده و هرگز خوانده نخواهد شد را معرفی می‌کند. در بعضی موارد، یک فرآیند به صورت دوره‌ای از طریق حذف نسخه‌های منسوخ اجرا می‌شود. این اغلب یک فرآیند توقف کلي است که یک جدول کامل را پردازش می‌کند و آن را با آخرین نسخه هر یک از آیتم‌های داده بازنویسی می‌کند. PostgreSQL با استفاده از فرآیند VACUUM حذف نسخه‌های منسوخ شده را انجام می‌دهد.

پایگاه داده‌های دیگر، بلوک‌های ذخیره‌سازی را به دو قسمت تقسیم می‌کنند: بخش داده و Undo log. بخش داده همیشه آخرین نسخه کامل شده را نگه می‌دارد. Undo log امکان بازسازی نسخه‌های قدیمی‌تر داده‌ها را فراهم می‌کند. محدودیت اصلی ذاتی رویکرد دوم این است که وقتی بار کاری به‌روزرسانی بیشتری وجود دارد، بخشی از undo log نمی‌تواند اجرا شود و پس از آن، تراکنش‌ها به دلیل عدم توانايي در گرفتن تصویر از پایگاه داده قطع می‌شوند. برای یک پایگاه داده مبتنی بر سند، همچنین اجازه می‌دهد تا سیستم به منظور بهینه‌سازی اسناد با نوشتن تمام اسناد به بخش‌های مجاور دیسک، هنگامی که به‌روز می‌شود، کل سند قابل بازنویسی شود؛ به جای آن‌که بیت‌ها و تکه‌ها جدا شوند یا در ساختار پایگاه داده پیوسته مرتبط نگه‌داری شوند.

MVCC دیدگاه‌های سازگار با زمان لحظه‌ای را فراهم می‌کند. تراکنش‌های خواندن تحت MVCC برای تعیین وضعیتی از بانک اطلاعاتی که باید خوانده شود، به‌طور معمول از یک نشان‌گر زمان (time stamp) یا شناسه تراکنش استفاده می‌کنند و این نسخه از داده‌ها را می‌خوانند. بنابراین، تراکنش‌های خواندن و نوشتن بدون نیاز به قفل شدن از یک‌دیگر جدا می‌شوند. با این حال، علی‌رغم ضروری نبودن قفل، در بعضی از پایگاه‌های داده MVCC مانند اوراکل استفاده می‌شود. نوشتن، یک نسخه جدیدتر ایجاد می‌کند در حالی که خواندن همزمان با آن، به یک نسخه قدیمی دسترسی پیدا می‌کند.

الگوریتم بانکدار

الگوریتم بانکدار یک الگوریتم تخصیص منابع و اجتناب از بن‌بست است که توسط ادسخر دیسترا توسعه یافته که امنیت آن به وسیله شبیه‌سازی تخصیص بیشترین مقدار ممکن از تمام منابع آزمایش شده به‌طوری که یک s-state ایجاد می‌کند تا برای همه فرایندهای در حال انتظار تمام شرایط بن‌بست را قبل از تصمیم‌گیری و اجازه تخصیص منبع بررسی کند. الگوریتم بانکدار هر زمان که یک فرایند درخواست منبع کند، توسط سیستم‌عامل اجرا می‌شود. الگوریتم به وسیله انکار یا تعویق درخواست از بن‌بست جلوگیری می‌کند. این در صورتی است که اگر تعیین شود که پذیرش درخواست می‌تواند سیستم را به حالت ناامن ببرد. (حالتی که بن‌بست می‌تواند رخ دهد). هنگامی که یک فرایند جدید وارد یک سیستم می‌شود باید حداکثر تعداد درخواست از هر یک از منابع را اعلام کند که البته نباید از تعداد کل منابع در سیستم تجاوز کند. همچنین هنگامی که یک فرایند همه منابع درخواستی را تحویل می‌گیرد باید آن‌ها را پس از اتمام عملیاتش، بازگرداند.

الگوریتم بانکدار برای انجام کار نیاز به دانستن سه مفهوم زیر دارد:

- هر فرایند چه مقدار از هر نوع منبع را درخواست کرده است.
- هر فرایند چه مقدار از هر نوع منبع را در اختیار دارد.
- چه تعدادی از هر منبع موجود است.

الگوریتم پترسون

الگوریتم پترسون یک الگوریتم برنامه‌نویسی همزمان برای انحصار متقابل است که به دو فرایند اجازه می‌دهد تا از یک منبع مشترک بدون هیچ تعارضی استفاده کنند و از حافظه مشترک تنها برای ارتباطات بهره ببرند. این الگوریتم توسط گری ال پترسون در سال ۱۹۸۱ طراحی شد. از آنجایی که الگوریتم اصلی پترسون برای تنها دو فرایند قابل اجرا است، الگوریتم را می‌توان به‌صورت زیر برای بیش از دو فرایند تعمیم داد. پیاده‌سازی الگوریتم پترسون و دیگر الگوریتم‌های وابسته به آن، در فرایندهایی که خواهان دسترسی مرتب به حافظه هستند، نیازمند عملیاتی دقیق برای نظارت بر اجرای درست و به‌ترتیب فرایندها است.

الگوریتم دکر

الگوریتم دکر (Dekker's algorithm) یک الگوریتم برنامه‌نویسی همزمان برای انحصار متقابل است که به دو فرایند اجازه می‌دهد تا از یک منبع مشترک بدون هیچ تعارضی استفاده کنند و از حافظه مشترک تنها برای ارتباطات بهره ببرند. این الگوریتم توسط ادسخر دیکسترا طراحی شده‌است.

الگوریتم نانوایی

الگوریتم نانوایی (Bakery algorithm)، الگوریتمی رایانه‌ای است که توسط لزلی لمپورت، دانشمند علوم کامپیوتر ابداع شده‌است. این الگوریتم، با استفاده از انحصار متقابل، ایمنی استفاده از منابع مشترک توسط ریسمان (Thread) که به‌طور همزمان اجرا می‌شوند را بهبود می‌بخشد. در مسائل مربوط

به علوم کامپیوتر، در بسیاری از اوقات چندین ریسه به‌طور همزمان سعی در دستیابی به منبع مشترکی را دارند. این منبع مشترک می‌تواند یک شمارش‌گر، محلی از حافظه، قطعه‌ای از کد برنامه، یا هر منبع دیگری باشد. اگر دو یا چند ریسه به‌طور هم‌زمان بر روی بخشی از حافظه بنویسند یا یکی قبل از آن‌که دیگری فرایند نوشتن را تمام کرده باشد، همان حافظه را بخواند، اصطلاحاً خرابی داده (Data corruption) اتفاق می‌افتد. الگوریتم نانوایی لمپورت یکی از چندین الگوریتم کامپیوتری است که با استفاده از انحصار متقابل، از ورود ریسه‌های همزمان به بخش‌های بحرانی کد و در نتیجه خرابی داده، جلوگیری می‌کند.

الگوریتم‌های غیرمسدودکننده

در علوم رایانه، به یک الگوریتم غیرمسدودکننده می‌گویند، اگر از کار افتادن یا توقف هر ریسه (رایانه) باعث از کار افتادن یا توقف یک ریسه دیگر نشود. برای بعضی عملیات‌ها، این الگوریتم‌ها جایگزین مناسبی برای پیاده‌سازی‌های مسدودکننده رایج هستند. اگر یک الگوریتم غیرمسدودکننده، پیشروی در سطح سیستم را تضمین کند، به آن «بدون قفل» یا «آزاد از قفل» می‌گویند. اگر یک الگوریتم غیرمسدودکننده، پیشروی در سطح ریسه را هم تضمین کند، به آن «بدون انتظار» یا «آزاد از انتظار» می‌گویند.

قفل چرخشی

قفل چرخشی (spinlock) قفلی است که باعث می‌شود ریسمانی برای به‌دست آوردن آن در یک حلقه منتظر بماند (چرخش می کند) و باید بارها و بارها چک کند که آیا قفل آزاد شده یا خیر. از آن‌جا که ریسمان همچنان مشغول است اما کار مفیدی را انجام نمی‌دهد، استفاده از چنین قفلی، نوعی انتظار مشغول است. قفل‌های چرخشی پس از آنکه به‌دست آورده شدند، معمولاً تا زمانی که به‌طور واضح آزاد نشوند، نگه داشته می‌شوند؛ اگرچه در برخی پیاده‌سازی‌ها، در صورت مسدود شدن ریسمان (ریسمانی که قفل را نگه می‌دارد) یا به خواب رفتن آن، ممکن است قفل به‌طور خودکار آزاد شود. از آن‌جا که قفل‌های چرخشی از سربار ناشی از زمان‌بندی مجدد فرایندها توسط سیستم‌عامل یا تعویض زمینه جلوگیری می‌کنند، فقط در صورتی کارآمد هستند که ریسمان‌ها احتمالاً فقط برای مدت کوتاهی مسدود شوند. به همین دلیل، هسته‌های سیستم‌عامل اغلب از قفل‌های چرخشی استفاده می‌کنند. با این حال، قفل‌های چرخشی اگر برای مدت طولانی نگه داشته شوند، بیهوده هستند، زیرا ممکن است از اجرای سایر ریسمان‌ها جلوگیری کرده و نیاز به زمان‌بندی مجدد داشته باشند. هرچه ریسمان بیشتر قفل را نگه دارد، خطر ایجاد وقفه در ریسمان توسط زمانبند سیستم‌عامل در حین نگه داشتن قفل، بیشتر خواهد بود. اگر این اتفاق بیفتد، ریسمان‌های دیگر «در حال چرخش» باقی می‌مانند (یعنی به‌طور مکرر سعی در به‌دست آوردن قفل دارند)، در حالی که ریسمان نگه‌دارنده قفل پیشرفتی در جهت آزاد کردن قفل ندارد. نتیجه این وضعیت، یک تأخیر نامحدود است تا زمانی که ریسمان نگه‌دارنده قفل بتواند کار خود را تمام کرده و قفل را آزاد کند. این امر به‌ویژه در سیستم‌های تک پردازنده‌ای صادق است که در آن هر ریسمان در حال انتظار با اولویت مشابه احتمالاً سهم زمانی خود را (زمان اختصاص داده شده که در این بازه یک ریسمان می‌تواند اجرا شود) در حال چرخش تلف می‌کند تا سرانجام ریسمانی که قفل را نگه داشته، به پایان برسد.

پیاده‌سازی صحیح قفل چرخشی چالش‌برانگیز است، زیرا برنامه‌نویسان باید امکان دسترسی همزمان به قفل را در نظر بگیرند که این امر می‌تواند باعث ایجاد شرایط مسابقه‌ای شود. به‌طور کلی، پیاده‌سازی قفل چرخشی فقط به‌وسیله دستورالعمل‌های خاص زبان اسمبلی، مانند عملیات یک‌جای تست و ست، امکان‌پذیر است و در زبان‌های برنامه‌نویسی که از عملیات یک‌جای واقعی پشتیبانی نمی‌کنند، به‌راحتی قابل اجرا نیست. در معماری‌هایی که چنین عملیاتی ندارند یا در صورت نیاز به پیاده‌سازی زبان سطح بالا، ممکن است از یک الگوریتم قفل‌کننده غیریک‌جا استفاده شود، به‌عنوان مثال الگوریتم پیترسون. با این حال، چنین پیاده‌سازی‌ای ممکن است به حافظه بیشتری نسبت به قفل چرخشی نیاز داشته باشد، برای امکان پیشرفت پس از باز کردن قفل کندتر باشد و زبان سطح بالا، در صورت مجاز بودن اجرای خارج از دستور، ممکن است قابل اجرا نباشد.