شروع سیستم عامل حقیقت:

**ویدیو اول:**

سیستم عامل از جنس نرم افزار هست، فقط BIOS در ROM اجرا میشود بقیه در رم هستند، امروزه همه جا DRAM قرار دارد. فرآیند یک برنامه در حال اجرا هست، یک برنامه شامل چند بخش هست یکیش تکست هست که تمام کد ها در آن هست، بخش دیگر داده هست و بخش دیگر استک یا پشته هست، دقت کن درسته برنامه اسمبلی چند تا برنامه هست یا چند خط ولی وقتی اسمبلر تبدیل میکند فقط یک برنام هست. فرمان ها به shell یا command interpreter میدهیم و فراخوانی سیستم ها را به کرنل میدهیم سیستم عامل یک کرنل و یک پوسته دارد.

یک برنامه در سطح یوزر خودش اجازه دارد فراخوانی سیستم انجام بدهد در مد کاربر، فراخوانی سیستم یعنی درخواست را مستقیم به کرنل بدهیم. برنامه ها دو نوع هستند برنامه های کاربردی و برنامه های سیستمی و برنامه های سیستمی با جزئیات سخت افزار سر و کار دارند. سیستمی ترین برنامه دنیا کرنل هست. سیستم عامل از نوع برنامه سیستمی هست. سیستم عامل سیستم ما را boot یا راه اندازی میکند. کی برنامه ها را میگذارد تا در حافظه تا اجرا شود؟ سیستم عامل، حالا کی خود سیستم عامل را اجرا میکند؟ خودش، BIOS فقط میاد آماده بوت میکنه بوت کردن را خود سیستم عامل انجام میدهد. اکثر برنامه های سیستمی در مد کاربر هستند، تنها چیزی که تو مد هسته هست، هسته هست. یعنی مثلا اختیارات کامپایلر با فتوشاپ یکی هست ولی اون یکی بیشتر باید بلد باشه اجزا رو.

یک فرآیند در قالب سیستم کال درخواست یک منبع را از سیستم عامل میکند. مثل دعا کردن در مقابل خداوند. حالا اگه داشت میدهد اگه نداشت میگه وایستا تو صف.

اگر سیستم عامل مدیریت منبع نکند هرج و مرج میشود. باعث هدر رفتن منابع میشود. در واقع کاربر نیاز های خودش را به برنامه میگوید و برنامه با استفاده از سیستم کال آن را کار را انجام میدهند.

به صورت اشتراک زمانی در هر تایم اسلایس یک برنامه را اجرا میکند، کی میاد سوییچ میکند بین برنامه ها؟ سیستم عامل، این وسط هم یک سری سیستم کال هست وسط برنامه ها که سیستم عامل اجرا میشود در واقع نوعی وقفه رخ داده هست یعنی لا به لای برنامه ها بار ها و بارها سیستم عامل اجرا میشود هر موقع وقفه بیاد. چه چیزی باعث سوییچ میشود؟ وقفه.

جزوه دوم:

سیستم عامل و سایر فرآیند ها حین اجرای یک برنامه اصلا نمیتوانند وقفه بدهند چون برنامه هستند و CPU ندارند که وقفه بدهند ( حتی اگه چند تا پردازنده باشد هر برنامه واسه CPU خودش میتواند وقفه بدهد) به وقفه نرم افزاری تله میگوییم.

وقفه سخت افزاری: به این دلیل میگوییم آسنکرون یا ناهمگام چون از قبل مشخص نیست کجا وقفه رخ میدهد هر سری جای جدید هست کلا وقفه های خارجی چون تصادفی هستند به این شکل هست. ولی در وقفه های نرم افزاری جاش معلوم هست از قبل یعنی همگام یا سنکرون هست. چون خود برنامه دارد وقفه میدهد یعنی برنامه نویس گفته تو خط 20 ام وقفه بده ده بار دیگه هم اجرا کنی همان هست. تایمر در سیستم های اشتراک زمانی نقش کلیدی دارد کی اعلام میکند به CPU ده ثانیه تموم شد؟ تایمر کی تعیین میکند وقتی تموم شد وقفه بدهد؟ سیستم عامل.

چرا فتوشاپ مثلا سیستم کال میکند برای یک منبع؟ حفاظت و پیچیدگی چون دسترسی ندارد. سیستم عامل باید یک محافظتی برای منابع انجام دهد تا هر کسی نتواند سراغ آن برود.

Exception یا استثنا: هر گاه فرآیند جاری خلاف بکند یعنی کاری بکند که اجازه اجرای آن را ندارد.

تا دقیقه 34 از قسمت سوم دیده شد.

موقع اجرای یک فرآیند سیستم عامل چون یک برنامه هست اصلا اجرا نمیشود واسه همین هیچ غلطی نمیتواند بکند. وقتی یک برنامه اجرا میشود هیچ شخصی جز CPU و فرآیند نیست هیچکس دقت کن. پس هر چیزی خطا اجرا میشود از جنس سخت افزار هست. اون مدیریت حافظه هست یا MMU که باعث میشود. یعنی وقتی که یک برنامه ای مثلا از آدرس شروع 120 تا 130 هست و بخواد به خانه 131 دسترسی پیدا کند MMU یا واحد مدیریت حافظه CPU جلوی آن را میگیرد. و MMU باعث وقفه دادن نرم افزاری و exception میدهد یعنی استثنا شما نمیتونی این کار را بکنی. کی اون عدد 120 تا 130 را گذاشته؟ قطعا سیستم عامل گذاشته ولی وقتی خطا میشود که سیستم عامل نیست که بفهمد.

سیگنال از طرف فرآیند پدر به فرزند میرود و آن را از بین میبرد. سیستم کال و وقفه سخت افزاری و تایمر و استثنا این ها مهم ترین وقفه ها هستند.

پردازش وقفه: وقتی وقفه بیاد چیکار میکنیم؟ کی به وقفه رسیدگی میکنیم؟ در پایان دستورالعمل جاری رسیدگی میکنیم همین دستورالعمل. از کجا میفهمیم چه نوع وقفه ای رخ داده هست؟ یا به جدول مخصوص آن نگاه میکنیم مثلا از کدوم پایه وارد میشود اشکال آن بود تعداد دستگاه ها محدود میشد چون نمیشد هعی پایه اضافه کنیم که کلا 5 تا بود. روش دیگر این بود هر کی با CPU کار داشت سیگنال درخواست وقفه بدهد. در این روش وقفه میگوید چه نوع وقفه ای هست چون با یک شماره میدهد و ما میدانیم در جدول به کدام خانه برویم و آنجا میرویم و سرویس وقفه مخصوص آن را اجرا میکنیم. بردار وقفه در حافظه هست. 256 خونه 4 بایتی برای وقفه داریم که 1 کیلوبایت براش در نظر میگیریم و در آن هر کدام آدرس سرویس وقفه هست.

PSW شامل همه فلگ های مهم در برنامه هست. CPU آدرس برگشت از وقفه را در پشته گذاشت و آدرس برگشت را ذخیره کرد. از هشت به بعد سیستم عامل اجرا میشود. PSW در استک ذخیره میشود PC هم که آدرس بازگشت را دارد در استک هست. CPU وقتی وقفه رخ میدهد میرود داخل مد هسته. وقفه های تو در تو داریم یعنی مثلا یک وقفه دارد اجرا میشود یک وقفه دیگر بیاد. اگر وقفه جدید اولویت پایین تری داشت باید صبر کند سرویس دهی وقفه فعلی تمام شود. یا اولویت بالاتری دارد که دوباره وقفه میخورد سرویس وقفه فعلی و اون را اجرا میکنیم.

تا 40 دقیقه ویدیو چهارم.

بن بست:  
بن بست یعنی یک سیکل انتظار ابدی یعنی وقتی که فرآیند ها منتظر همدیگر باشند، یعنی اینکه یک گروهی منتظر هم هستند الزاما دو تا نیستند، تعداد فرآیند ها در این سیکل میشود فرد هم باشد شاید دو تا فرآیند منتظر یکی دیگر باشد، ممکن هست یکی منتظر دیگری باشد ولی کسی دیگری منتظرش نباشد، یعنی همشون منتظر هستند هیچ کاری هم نمیکنند، چرا یک فرآیند منتظر هست اصلا؟ تا الان هر چی بن بست دیدیم بن بست همگامی بود یعنی منتظر پیام بودیم جفتمون receive کردیم و بعد خوابیدیم یا در synchronization در سمافور در انتظار سیگنال هستیم، راه حلی هم نداریم فقط به برنامه نویس ربط دارد اما در این فصل فرآیند منتظر منبع هستیم که این منبع در اختیار فرآیند دیگر است ولی ما اون فرآیند را نمیشناسیم صرفا دنبال منبع هستیم، در واقع دنبال فرآیندی هستیم که اصلا ارتباطی باهاش نداریم و پیام ندیم و این مورد را نمیتوان مثل مورد قبلی در تولید کننده و مصرف کننده با حواس جمعی برنامه نویس حل کرد ما یک سری فرآیند داریم که هیچ ارتباطی با هم ندارند صرفا دنبال منابع مثل مانیتور یا پرینتر هستند و برنامه نویس نمیتواند حل کند، منبع میتواند هر چیزی باشد، دو دسته هستند مثل منابع منطقی مثل فایل یا ساختمان داده یا منابع فیزیکی مثل حافظه یا ورودی خروجی و CPU، سر فایل یا ساختمان داده ای میشود دچار بن بست شویم، در هر دو منبع، ما در این فصل شکست میخوریم یعنی مسئله ای را مطرح میکنیم که نمیتوانیم حل کنیم و سیستم عامل باید یک نوعی حل کند که نمیشود، گام اول برای درخواست یک منبع با سیستم کال یک request میزند مثل درخواست خواندن یا نوشتن در فایل، دو حالت داریم بعدش یا wait میکنیم تا منبع آزاد شود و اگر از اول آزاد بود منبع allocate میشود، البته اگر نوبتش باشد بعد از این مرحله use میکند و بعد از آن release میکند و منبع را آزاد میکند.

بن بست به شکل گرافیکی: در سیستم عامل سه بخش داریم فرآیند دایره شکل و منبع مربع شکل و خود سیستم عامل، گاهی اوقات هم داخل این مربع یا مستطیل با دایره توپر نشان میدهیم و به تعداد منابع نشان میدهیم مثلا 3 تا پرینتر داریم، واژه hold هم داریم که معادل allocate هست که مثلا فلان فرایند منبع را نگه داشت است.

وقتی منتظر هستیم باید فرآیند بخوابد، و با خوابیدن بن بست رخ میدهد چون یکی منتظر یک منبع هست که اون یکی دست یک فرآیندی هست که الان خوابیده هست، یک سیکل 3 فرایند هم میتوانیم داشته باشیم، اینکه فکر کنیم به صرف اینکه یک فرآیند اولویت بالاتری دارد بزور بیایم یک منبع را از فرآیند دیگری بگیریم که اول اون حل شود بعد هر 2 تا را بدهیم به فرآیند قبلی این تفکر اشتباه هست، دقت هم کن یک فرآیند میتواند بی نهایت منبع داشته باشد، و خیلی مسخره هست که به فرآیند بگیم فقط یک منبع بگیر این تفکر کلا اشتباه هست، چرا سیستم عامل بزور منبع را نمیگیرد؟ وقتی میتواند چرا نمیکند؟ چون وقتی یک فرآیند روی اون منبع یک سری فعالیت انجام داده است نمیتوان وسط کار بزور ازش بگیریم و بدیم به یکی دیگه چون تداخل رخ میدهد، اصلا اگر وسط کار ول کنیم مشکل کجاست؟ به اون فعالیت هایی که روی منبع رخ داده است فکر کن باید برگرده از اول دوباره انجام بدهد و نمیتواند وقتی تو خط صد هزار هست برگردد به خط 2 هزار و از اونجا دوباره شروع کند اصلا نمیتواند چون نمیداند از کجا اومده و کجا بوده است پس دقت کن اصل مطلب همون کاغذ و نقاشی هست وقتی یک وسط کاغذ نقاشی کرده که نمیتونی ازش بگیری یک چون تداخل میشه با فرآیند بعدی و نقاشی بعدی و دو اینکه باید فرآیند اولی از اول شروع کند به کار کردن و اصلا این کار را نمیتواند انجام دهد، نه اون کاغذ بدرد دومی میخورد که نصفش نقاشی شده و نه بدرد اولی میخورد چون وسط کار بزور ازش گرفتیم و هیچی به هیچی، پس بزور گرفتن نتیجه نمیدهد و تفکر اشتباهی هست، یعنی نقاشی قبلی را کشتی، ما منابع را در سیستم عامل بزور نمیتوانیم وسط استفاده پس بگیریم چرا؟ به خاطر همان مسائلی که گفتیم، جمله ما منابع را بزور پس نمیگیریم هم غلط هست چون بستگی به منبع دارد از اول ما CPU را بزور پس میگیریم ولی پلاتر را بزور پس نمیگیریم چون اولی از نوع غیر انحصاری بود در ذاتش یعنی بزور پس گرفتن مثل round robin، و این خطرناک نیست و PCB کمک میکرد که یک رجیستر بود که یک پردازنده مجازی ایجاد میکرد و چندین فرآیند را به صورت مجازی اجرا میکرد و مشکلی نداشتیم و اطلاعات هر کدام را ذخیره میکرد و time sharing میکرد، چون اون نقاشی خراب میشود پس پلاتر را نمیتوانیم پس بگیریم و اون منبع میسوزد و از بین میرود و این منابع non primitive هست یعنی بزور نمیتوان پس گرفت، پس دقت کن طبق فرض non primitive بوده که بن بست رخ داده است و هر سیکلی که دایره میکشند بن بست نیست سیکل تنها شرط بن بست نیست بن بست 4 شرط دارد اولیش این است که منبع non primitive باشد CPU نیست ولی حافظه و پلاتر و پرینتر و بقیه موارد هستند. پیشنهاد دوم این است که زمان بندی کنیم و مهلت بدیم به فرآیند ها چون اونم وسطش پس میگیریم باز مشکل داریم، کلا هر روشی به نام بزور پس گرفتن منتفی است، به اشتراک گذاشتن منابع هم نمیشود چون همزمان نمیشود دو نفر نقاشی بکشند روی یک نقاشی، یکی دیگر از شروط این است که دو فرآیند باید انحصار متقابل را رعایت کنند و همزمان نمیشود از منبع به صورت مشترک همزمان استفاده کرد، CPU باعث بن بست نمیشود چون وسط کار نباید منبع را از یک فرآیند بزور بگیریم.

در صورتی که دستشویی هم no preemption دارد هم انحصار متقابل و باعث بن بست میشود.

حافظه هم انحصار متقابل نقض میشود و هم preemption داریم، در واقع شروط 1 و 3 نقض میشود و بن بست داریم از این شروط یکی را هم نداشته باشد کافی است و دچار بن بست نمیشود، خواندن از فایل انحصار متقابل را نقض نمیکند، ولی همزمان نمیتوانی بنویسی چون شرایط رقابتی میشود، یکی بخواند یکی بنویسد هم نمیشود نوشتن با نوشتن هم نمیشود مثل نوشتن در یک رکورد پایگاه داده، ولی اگر هر دو بخوانند موردی ندارد ولی دو مورد بعدی را نمیتوانیم و انحصار متقابل باید رعایت شود، مثلا اینطوری هم میشود چون انحصار متقابل نداریم یکی میخواد از فایل بخواند یکی هم منتظر پلاتر هست و میخواهد از فایل بخواند این بن بست نیست چون یکی پلاتر را میگیرد اون یکی هم میخوابد و در خواندن از فایل مشکلی نیست و بعد اون یکی که خوابیده بود بیدار میشود و استفاده میکند، همزمان نمیشود در منبع منطقی نوشت، حافظه باعث بن بست نمیشود بلکه اون منابع مثل منبع منطقی هستند که باعث میشود بن بست ایجاد شود، نمیگیم RAM باعث شده بلکه میگیم منبع منطقی باعث شده است که مثلا در یک خانه از حافظه همزمان بخوانیم و بنویسیم، خواندن از فایل read only انحصار متقابل نیست و باعث بن بست نمیشود، CPU تک هسته ای انحصار متقابل دارد اما non preemptive نیست و بن بست ندارد، ولی چاپگر و درایور تیپ و پلاتر و اسکنر هر دو شرط را دارند و بن بست دارند، چند پردازنده ای هم باعث بن بست نمیشود و انحصار متقابل هم ندارد، سی دی وسط نوشتن را نمیتونی ازش CPU را بگیری پس استفاده از منبع مشترک هم چون داستان انحصار متقابل هست غلط هست، فقط read only موردی ندارد، وقتی بن بست رخ بدهد یعنی منابع انحصار متقابل دارند و non preemptive هستند، اگر بن بست رخ بدهد باید فرآیند را بکشیم راه دیگری نیست ولی نیاز نیست هر دو فرآیند را بکشیم یکی را بکشیم کافی هست البته کشتن راه حل نیست بلکه ترمیم هست، بین فرآیند قدیمی و جدید اولویت با قدیمی هست یعنی اول جوان تر ها را بکشید چون بیشتر منابع استفاده کرده است به صرفه تر هست چون اون دو دقیقه ای مثلا دو دقیقه هست ولی یکی ممکنه ده روز طول کشیده باشه نمیشه برگشت ده روز عقب، پس به این کشتن ترمیم یا recovery میگویند، بن بست اگر و فقط اگر این 4 شرط رخ بدهد و اگر این 4 شرط برقرار باشد بن بست رخ داده است یعنی شرط لازم و کافی هستند، hold and wait را پذیرفتیم که یک فرآیند ممکن است وسط اجراش که یک منبع را دارد منتظر بماند برای یک منبع دیگر، بر حسب نیاز به منابع درخواست میدهند اگر سیکل به وجود بیاد که بدبخت هستیم 3 تا شرط اول با 4 ام فرق دارد 3 تا اول تو ذات منبع هست ذات پرینتر non preemptive هست ذات اسکنر انحصار متقابل هست یعنی در یک لحظه از قبل مشخص نیست بگیم در ثانیه فلان در یک لحظه بن بست میشود، شرط 4 ام در یک لحظه رخ میدهد.

ممکن است یک شرایطی هر 3 تا شرط اول باشد ولی هنوز بن بست نیست چون انتظار چرخشی رخ نداده است و سیکل تشکیل نشده است بعد که بشود بن بست میشود، سیستم عامل ها میتوانند خودشان فرآیند را بکشند و بعد دوباره اجرا کنند ولی معمولا این کار را نمیکنند دقت کن به فلان فرآیند نمیتوانیم بگوییم فلان منبع را نخواه وقتی میخوابونیم یعنی منتظر نگهش میداریم، دقت کن اینکه دو تا پرینتر داشته باشیم بازم ممکن است بن بست شود، افزایش تعداد منابع باعث جلوگیری صد در صد نیست ولی احتمال را کم میکند.

الگوریتم تشخیص بن بست داریم از الگوریتم کافمن البته این حل نمیکند فقط میفهمد و بعد از detection باید ترمیم کنیم، یکی را میکشیم تا بقیه را آزاد کنیم و از بن بست در بیایم، دقت هم کن فرآیند ها اول یک منبعی را میگیرند بعد درخواست یک منبع دیگر را میدهند، در اون مثال سیلبر شاتز ممکن است p4، r3 را ازاد نکند ما مطمئن نیستیم بن بست میشود یا نه و ما مطمئن نیستیم ازاد میکند یا خیر ممکن است اون وسط اون فرآیند بخواهد r1 را بخواهد و بن بست شود، امیدواریم که اون منبع را آزاد کند، اون مثال بخاطر این بن بست هست چون سیکل ندارد، سیکل گرد نیست گروهی از فرآیند ها هستند که همه منتظر هم هستند.

جلسه 34 به طور کامل دیده شد.

راهکار های بن بست:

پس فهمیدیم وقتی بن بست رخ داد کاری نمیتونیم بکنیم ولی آیا میتوانیم ازش پیشگیری کنیم که بن بست رخ ندهد؟ 4 استراتژی داریم: پیشگیری، اجتناب ( هر دو وجه مشترک این هست که بن بست رخ ندهد)، کشف و ترمیم، شتر مرغ. البته دقت کن 2 راه حل اول جواب نمیدهد و فقط ترمیم و شتر مرغ جواب میدهند، شتر مرغ یعنی بیخیال شو عین کبک سرتو بکن توی برف، یعنی ignore کنید، اولی دومی میخوان یه کاری کنن بن بست رخ ندهد سومی میگه تشخیص بده بعد بکش، با الگوریتم کافمن تشخیص میدهیم بعد میکشیم که میگوییم detection and recovery، یعنی میپذیریم رخ داده بعد میکشیم، لینوکس از ویندوز از روش های اول و دوم نمیرن از روش سوم هم نمیرن از روش شتر مرغ میرن چون کشف و ترمیم پر هزینه هست و نمیصرفه، سومی رو میشه انجام داد ولی نمیصرفه، به نوعی trade off میکنیم، سود ترمیم نسبت به هزینش خیلی کم هست چرا کمه؟ بخاطر الگوریتم هست که یک گراف خیلی بزرگ با صد ها نخ هست، هزاران منبع داریم و ماتریس های بزرگ داریم و وقت گیر هست، دوما کی الگوریتم را باید اجرا کنیم؟ چند بار باید اجرا کنیم؟ کی الگوریتم detection باید اجرا شود؟ 3 شرط اولیه همیشه برقرار هست از اون 4 تا چون مشخصه دیگه پرینتر از دیروز انحصار متقابل نداشته که از اول داشته و برای همین میپذیریم همیشه این شرایط رو، حالا کی باید اجرا کنیم؟ هر وقت یک فرآیند درخواست یک منبع جدید میکند که مشغول هست یا آزاد نیست و دست یکی دیگست و wait جدید هست البته درست از لحاظ کنکور این هست هر وقت درخواست جدید اومد، و چون تعداد درخواست ها زیاد هست پر هزینه میشه چون تعداد گراف ها و زمانش خیلی بیشتر هست و سود چیه؟ فقط میکشیم یعنی بعد اینکه فهمیدیم و کلی زمان برد فقط میکشیم و هیچکاری نمیکنیم، بن بست نادر هست بعد هر 6 ماه و همیشه کند باشیم و آخر سر هم بکشیم پس بیخیال باشیم و سریعتر باشیم بهتر هست، حالا کی بن بست را میفهمد؟ وقتی برنامه هنگ میکند و جواب نمیدهد، کی کشف کرد؟ کاربر و بعد خود کاربر کشت و ویندوز این کارا نمیکند، و سیستم عامل وقت خودش را تلف نمیکند و به عهده کاربر هست، الگوریتم کافمن برای detection هست و الگوریتم banker جز اجتناب هست، فرآیند ها از دید سیستم عامل black box هستند یعنی از درون آن ها با خبر نیست، سیستم عامل های شتر مرغی نه در آینده نه در حال و نه در گذشته کاری انجام نمیدهند، الگوریتم های اجتنابی که عاجز هستند باید در گذشته اجرا شود و الگوریتم detection باید همیشه اجرا شود قبل و حال و بعد، کافمن کی اجرا میکردیم؟ هعی اجرا میکنیم یکیش بلاخره بن بست میشه اینطوری نبود تو یک لحظه اجرا کردیم بعد همون موقع بن بست شد، الگوریتم کافمن را دوره ای بگیریم اگر کم به کم باشد که فرقی نمیکند اگر زیاد هم باشد که بعدش خیلی منتظر میشویم و الکی هست.

پیشگیری: کاری کنیم که حداقل یکی از 4 شرط بن بست رخ ندهد چون هر 4 تا باید رخ بدهد تا بن بست شود پس اگر جلوی یکی را بگیریم جلوی بن بست را گرفتیم، که کلا شکست میخوریم البته، دلیل اینکه ما جلوگیری نمیکنیم از بن بست هزینه زیاد و یا نادر بودن نیست بلکه غیرممکن هست، حتی در سیستم های همه منظوره مثل لینوکس و ویندوز، اگر embedded باشد چی؟ ( منظور برای سیستم عامل های عاری از بن بست هست)، در **پیشگیری روش ها غیر ممکن** هستند، پیشگیری یعنی به یکی از اون 4 شرط حمله کنیم مثلا یکی برای انحصار متقابل هست، برای روش انحصار متقابل باید گفت که همه منابع فیزیکی و منطقی نمیتوان همزمان نوشت یا spool کرد مثلا در پرینتر میشود ولی در رکورد یک پایگاه داده نمیشود همزمان دو نفر بنویسند یا مثال دیگر PCB هست، محدودیت تعداد رکورد های PCB مثلا بیشتر از 100 تا نمیشود و بیشتر از 100 تا نمیخواهیم فرآیند درست کنیم و پس اگه بخواهیم فرزند بسازیم نمیشود چون باید منتظر بماند یکی از بین برود تا بیاید داخل PCB، و خوب وقتی هیچکس نرود بیرون بن بست رخ میدهد، انحصار متقابل یعنی چند نفر همزمان نمیتوانیم از یک منبع استفاده کنیم، اگر همه منابع قابل spool بودن باز ممکن هست بن بست رخ دهد به علت محدود بودن فضا disk یا spool.

در مثال spool چون همه منتظر همدیگه هستن همه خوابیدن و فضا پر شده است و مثلا یکی میخواد صفحه 201 رو پرینت بگیره ولی نمیشه چون فضا پر هست بعد میخوابد درسته این اتفاق نادر هست که این همه فرآیند بخواهند این مقدار از پرینتر استفاده بکنند ولی اگه اتفاق افتاد چیکار کنیم؟

برای حمله به نگهداری و انتظار میگیم همون اول بگو هر چی میخوای این غیر ممکن هست ولی اگر بشه چی میشه؟ اگر همه با هم همه منابع رو بخوان؟ مثلا چند تا مشترک بخوان چی؟ دو حالت داریم مثلا 5 تا منبع داریم، یا همه یا هیچ یا همش رو میدیم یا هیچکس رو نمیدیم میگیم بخواب، خوب اگه همش رو بدی من هرگز wait نمیکنم، چون من چیزی ندارم که بقیه بخوان منتظرش بشن یا هیچی ندارم که کلا wait کردم، وقتی wait کردیم چون چیزی ندارم که بقیه منتظرش بشن و کسی هم منتظر ما نمیشه، مشکل کجاست؟ هزینش زیاده مثلا تو الان پرینتر رو گرفتی تا پس فردا نمیخواهیش پس داریم هدر میدیم و اصلا غیر ممکن هست چون ما باید پیشبینی کنیم که دو روز دیگه پرینتر میخواهیم که نمیشود باید اجرا شود تا به آن خط کد برسد، بن بست شبیه مرگ در زندگی انسان ها هست و قابل پیشبینی نیست، پس چرا نمیتوانیم جلوی مرگ را بگیریم چون نمیتوانیم پیشبینی کنیم پس فرآیند ها هم نمیتوانند از بن بست پیشبینی کنند و خیلی از روش ها نیاز به پیشبینی آینده دارد، یک روش دیگر این هست که مثلا 2 تا منبع گرفتیم الان سومیش رو میخواهیم میگیم نمیشه اول یک و دو رو بده بعد من بیا هر 3 تا رو بهت میدیم، بعد هر 3 تا رو درخواست میدهیم و این روش غیرممکن هست چون ما از منبع یک و دو وسط کار هستیم نمیتوانیم پس بدهیم چون منابع non preemptive هستند، در انحصاری نمیتوان به زور به منابع حمله کنی که این هم غیر ممکن هست، در انتظار چرخشی هم نمیشود در هر لحظه از زمان یک فرایند فقط یک منبع را داشته باشد غیر ممکن هست و باعث اتلاف منابع هم میشود چون یک فرآیند همزمان نیاز دارد بعد وقتی همه آزاد هست چه نیازی هست بگیم در هر لحظه فقط از یکی استفاده کن که خوب این غلط هست منابع هدر میرود، یک روش دیگر برای حمله به انتظار چرخشی این است که منابع را شماره گذاری کنیم و به درخواست آن ها را به ترتیب مثلا صعودی، غیرممکن هست ولی البته سیکل تشکیل نمیدهد ولی زنجیره تشکیل میدهد زنجیره hold and wait ولی این از روش مهندسی قابل پیاده سازی نیست چرا؟ چون ممکنه یکی اول r3 میخواد بعد r1 شماره ها را هم که نمیتوانیم عوض کنی پس به مشکل میخوریم پیشنهاد اینکه قبل از گرفتن r5 بیا r3 رو درخواست بده اگه بعدا میخوای بعد الکی میگیره تا جایی که نیاز داره هدر میده جلوی بن بست رو میگیره ولی بازم نمیتوانی چون پیشبینی آینده نداریم که تو آینده اون r3 را میخواهیم یا نه که روش 6 با 7 یکی هست فقط میگه منابع فعلی را شماره گذاری کن به صورت صعودی.

این رو هم مشکل بالا را دارد چون صعودی نیست مثلا الان r7 رو داریم ولی نیاز به r5 داریم یعنی نسبت به بالایی بهتر هست چون میگه نسبت به شرایط فعلی شماره گذاری کن.

همه این ها نیاز به پیشگیری دارد علت شکست اکثرشون پیشبینی آینده بود پس پیشگیری غیر ممکن هست.

تا آخر 35 دیده شد.

آخرین تلاش اجتناب یا avoidance هست که دو نوع کنترل داریم کنترل با پذیرش درخواست و کنترل با پذیرش فرآیندها، در پیشگیری هر درخواست را بررسی نمیکنیم مثلا spooling داریم کنترل به اون شکل نداریم ولی اینجا میگیم با ورود یک فرآیند جدید به بن بست نزدیک تر میشویم یا خیر، الگوریتم آقای دکسترا که به نام بنکر هست یا banker هست که به بن بست نزدیک میشود و در حین اینکه نزدیک میشود سعی میکند اجتناب کند یعنی تا حد ممکن نزدیک میشود ولی نا امن نمیشود یعنی نمیگذارد شرایطی پیش بیاد که دیگه کار از کار بگذرد، البته الگوریتم کار نمیکند و باید ببینیم در آینده چه میشود چون به پیشبینی نیاز دارد، یک تشبیه به بانک هست در آنجا خداوندگار مدیر بانک هست و اینجا سیستم عامل اون یکی سری منابع دارد ما هم یک سری منابع داریم اون پول دارد ما پرینتر و ... ، اونجا یک منبع بیشتر نداریم ولی اینجا n تا منبع داریم، اول یک منبع رو در نظر میگیریم بعد برای n تا منبع حل میکنیم، از همون حداکثر نیاز مشخص هست که این الگوریتم کار نمیکند و قابل مشخص شدن نیست که فرآیند ها در آینده بگن ما 3 تا منبع نیاز داریم یا 2 تا، دقت کن توی اون منابع مثال باید need یا available راه محاسبه کنیم خودمان چون آن ها را نمیدهند:



برای محاسبه possessed هم allocation ها را با هم جمع میکنیم، وضعیت امن و یا نا امن یک راه این هست که ببینیم مقدار need کمتر مساوی مقدار allocated هست یعنی اگر یک سطر هم بود میتوانیم بگوییم این امن هست البته این کافی نیست و به تنهایی نمیتوان گفت.

ادامه مسیر این میشود اونکه برقرار بوده تا آخر بریم هر چه منبع خواست بدیم اجرا شه بعد که max را دادیم استفاده کرد و آزاد شد حالا برویم اون هایی که برقرار شدند را حساب کنیم که در نهایت باید درخت بکشیم، بعد که یکی را رفتیم خواستیم بدونیم چه قدر داریم که کار بعدی ها را راه بندازیم اون فرمول پایین را که قدیمی را بعلاوه allocated کرده میزنیم تا ببینیم امن هست یا نیست. و تعداد مسیر امن ها را بدست میاوریم، وجود حتی یک مسیر امن هم کافی هست. همیشه باید available را اپدیت کنی. دقت هم کن ما تو وضعیت ب هستیم فرض میکنیم رفتیم تو ج تا ببینیم امن هست یا نا امن هست. اگر هیچکدوم شرط اول بالا را که بررسی کردیم نداشت که نا امن هست اگه یکی داشت اون آغاز امن بودن هست باید تا آخر بریم ببینیم امن هست یا نه به تعداد همه هم باید چک کنیم. حالا نا امن هست یعنی چه؟ یعنی ممکن هست بن بست بشود نه که حتما میشود، در نا امن وقوع بن بست حتمی نیست. بلکه احتمال دارد.

چرا میگیم احتمال دارد قطعی نیست؟ به کجا امیدواریم؟ فرآیند شاید قبل از درخواست منبع، یک منبع آزاد کند، یعنی اینکه کی گفتی صعودی اینقدر میگیرد تا به max برسد یعنی هعی افزایشی میگیرند ولی نه اینطور نیست، یعنی هعی ممکنه بگیره آزاد کنه بگیره آزاد کنه تا یکروزی به max برسد اینطوری نیست به صورت افزایشی هعی بگیرند تا بترکند. تنها مراقبت کن وارد فضای نا امن نشیم.

در الگوریتم banker، ما در بدترین حالت میریم یک فرآیند را تا ته اجرا میکنیم تا ببینیم چی میشود یعنی در بدترین حالت یک به یک اجرا میشوند ولی وقتی که اوضاع خوب باشد که اصلا اینکار را انجام نمیدهیم، پس دقت کن الگوریتم banker، با اشتراک زمانی و همروندی مشکلی ندارد بلکه تو بدترین حالت ممکن فرآیند ها یک به یک اجرا میشوند.

حالا الگوریتم banker با منبع چندگانه: فقط فرقش اینجاست که اونجا allocated یک ستون بود اینجا چندین منابع به شکل ماتریس هستند. اگر خوش شانس باشیم در کنکور available را بهمون میدن اگر بد شانس باشیم موجودی اولیه را به ما میدهند، باید منهای possessed کنیم تا available بدست بیاد، بعد جمع possessed چطوری بدست میاد؟ جمع allocated ها از یک ستون بعد حاصل را زیر قسمت مرتبط با اون مینویسیم چون دقت کن چند ستون هست و ماتریس هست و باید متناسب با هر exist مقدار possessed را بدست بیاوریم، جداول رو برای چک کردن حالت امن چک کن need کوچکتر مساوی available هست یا نه، به ازای هر ردیف چک کن متناظر با available قرار بده هر جا که کمتر مساوی بود اوکی بود دقت کن به تمام ارقام اون ردیف باید برقرار باشد.

مسیری که با فرآیند های آزاد شده برویم تعداد راهی که باز میشود فاکتوریل هست که مثلا در اون مثال چندگانه میشود 3 فاکتوریل، چرا چون میگیم p1 چند تا رو واجد شرایط میکنه؟ 3 تا پس میشود 3 فاکتوریل، وقتی یکی رفتی و تعداد بیشتری داشت دیگه لازم نیست اون یکی رو بری چون مشخص هست کمتر آزاد میکند ولی اگر کل مسیر های امن رو خواست طبیعی هست که باید هر دو طرف بری و مسیر های آزاد شده را جمع بکنی. پس دیگه لازم نیست کل درخت رو بکشی سرعت بالاتر میرود با اون فاکتوریل.

دقت کن توی اون حالت الف به ب اگر یکی بخواد درخواست منبع کنه میگیم بخواب بهت نمیدیم، مگر اینکه c بدیم که راه فرار ماست چون بقیه رو میخوابونیم بدترین حالت اینه c رو تا اخر بریم.

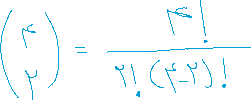
الگوریتم کافمن که خیلی شبیه الگوریتم بنکر هست البته این الگوریتم برای detection هست و فقط کافی هست جا need بالا request بزاریم، این الگوریتم نیازی به آینده ندارد و فقط وضعیت موجود هست، در کافمن شکست بخوریم بن بست هست و در بنکر شکست بخوریم نا امن هست.

Need عین request هست همون رفتار را باهاش داشته باش و همچنین available بدرد ما میخورد این هم دقت کن.

در اون تست 5 هیچکدام در بن بست نیستند، کافمن عین الگوریتم بنکر هست و به آینده کاری ندارد و از روی گراف تشخیص میدهد سیکل داریم یا نه، توی allocated و request ها دنبال بن بست میگردد، میگه که اینقدر منبع در گذشته بوده اینقدر هم available داریم میگه الان بگو بن بست داریم یا نه، همین کافیه تا بفهمیم و ربطی به آینده ندارد.

چگونه سیستم فاقد بن بست هست؟

فرمول چند وضعیت بن بست داریم: که در واقع میشود



انتخاب2 از 4 که جایگشت هست. که 4 تا فرآیند هست و 2 تا منبع

که هر جفتشون میگن با منتظر دومی هستیم، جزوه شماره 19 هست یا 17 چت ها از سوال بالا به پایین، اگر p1 دو تا منبع داشت دیگر بن بست نیست که max to run complete اجرا میشود و آزاد میکند. ببین 4 تا فرآیند هستن دو تا رو انتخاب کن اینطوری فکر کن از 4 تا آدم 2 تا رو انتخاب کن. مثلا تو بعدی به چند طریق میتوانیم یکی را انتخاب کنیم که دو تا منبعی باشه، خوب میشه یک از چهار دیگه و چطوری میتوانیم از 3 تا دو تا را انتخاب کنیم که میشود 2 از 3.

فرمول سیستم فاقد بن بست: اگر هر فرایندی 3 تا میخواست و مثلا p تا دادیم یعنی به هر کدومشون max -1 میدادیم که میشود n \* max -1=e این میشود تک وضعیت بن بست یعنی همه اونقدر گرفتن و منتظر یکی دیگه هستن که میشود تک وضعیتی شرط فاقد بن بست چیه؟ اون n \* max-1 به جای مساوی با exist، exist یکی بیشتر باشد که میشود سیستم فاقد بن بست هست، این شرط را حفظ کن:



البته این فرمول در شرایطی هست که فرآیند ها حداکثر نیاز را

به صورت مساوی بگیرند اگر یکی حداکثر هفت تا میخواست یکی 4 تا به این شکل میگیم که همه یکی کمتر بگیرند از اون max خودشون:



شروع ویدیو 38:

فاقد بن بست باید منبع خیلی زیادی برایش کنار بگذاریم، از کجا بفهمیم باید از این فرمول ها استفاده کنیم ؟ وقتی گفت سیستم فاقد بن بست باشد، فرمول های بالا برای چند منبع نیست.