موسسه بابان

انتشارات بابان و انتشارات راهیان ارشد

درس و کنکور ارشد

سیستم عامل

(حل تشریحی مهندسی کامپیوتر دولتی ۱۳۹۷)

ویژه‌ی داوطلبان کنکور کارشناسی ارشد مهندسی کامپیوتر و IT

براساس کتب مرجع

آبراهام سیلبرشاتز، ویلیام استالینگز و اندرو اس تنن‌بام

ارسطو خلیلی‌فر

تست‌های فصل چهارم

۷۵- کدام عبارت درست‌تر است؟

(مهندسی کامپیوتر – دولتی ۹۷)

۱) Virtualization به شبیه‌سازی یک سیستم عامل وابسته به سخت‌افزار، بر روی یک سیستم عامل وابسته به سخت‌افزار دیگر اطلاق می‌شود.

۲) NUMA یک حافظه توزیع شده است که در آن هر پردازنده یا هسته، به بخش‌های مختلف اختصاصی دسترسی دارد.

۳) System Daemon یک برنامه سیستمی مقیم در حافظه است که در صورت لزوم به صورت ناهمگام اجرا می‌شود.

۴) Emulation به فرآیند شبیه‌سازی یک سیستم عامل داخل سیستم عامل دیگر اطلاق می‌شود.

پاسخ‌های فصل چهارم

۷۴- گزینه (۳) صحیح است.

صورت سوال به این شکل است:

عبارت درست‌تر است؟

۱) Virtualization به شبیه‌سازی یک سیستم عامل وابسته به سخت‌افزار، بر روی یک سیستم عامل وابسته به سخت‌افزار دیگر اطلاق می‌شود.

گزینه اول پاسخ سوال نیست. زیرا مجازی‌سازی را می‌توان تکنولوژی استفاده از نرم‌افزار به جای سخت‌افزار دانست.

از مجازی‌سازی برای ایجاد کردن چندین سرور مجازی بر روی یک سخت‌افزار استفاده می شود. بعنوان مثال شما می‌خواهید از نرم‌افزارهایی استفاده کنید که فقط بر Windows XP اجرا می شود ولی سیستم عامل نصب شده بر روی سیستم شما Windows 7 است و قصد پاک کردن آن را هم ندارید در چنین شرایطی کاربر باید از مجازی‌سازی استفاده کند و هر دوی سیستم عامل‌ها را به روی یک سیستم فیزیکی نصب و اجرا کنند. و یا اینکه شما مدیر شبکه یک سازمان هستید و قصد گسترش ابعاد شبکه سازمان را دارید به طور حتم الویت شما فراهم کردن سرور های بیشر است که بسیار مشکل و هزینه بر خواهد بود اما با بهرگیری از مجازی‌سازی همه‌ی محدودیت‌ها و موانع از سر راه شما برداشته می‌شود و شما می‌توانید به راحتی و با کمترین هزینه شبکه سازمان گسترش دهید. همانطور که قبلا گفتیم از مجازی‌سازی برای ایجاد چندین سیستم مجازی استفاده می‌شود تا بتوانیم رفع نیاز کنیم. اگر قصد بررسی مجازی‌سازی را در لایه‌های معماری داشته باشیم به ترتیب در اولین لایه دستگاه SAN Storage یا ذخیره‌سازی اطلاعات قرار خواهد گرفت و در لایه بعدی سخت‌افزارها و سیستم‌ها تامین منابع قرار خواهند گرفت و در آخر سرور میزبان قرار می‌گیرد. از جمله برنامه‌ها و نرم‌افزارهای پرکاربرد مجازی‌سازی می‌توان از Microsoft ،Vmware ،Hyper-V ،Sun xVM و Citrix نام برد. در واقع این برنامه‌ها همان سرور میزبان (Host Server) در لایه آخر معماری هستند که سیستم مورد نیاز ما را در خود نگهداری می کنند. با استفاده از این روش می توان از منابع سخت افزاری موجود به نحو احسنت استفاده کرد و از بروز بسیاری از مشکلات و هزینه ها پیشگیری کرد.

تکنولوژی مجازی سازی شامل سه شاخه اصلی می‌باشد:

۱- Server virtualization

این تکنولوژی که با نام‌های دیگری مثل Hardware virtualization و OS Virtualization نیز شناخته می‌شود، دو ویژگی زیر را برای ما مهیا می‌نماید:

الف) به جای راه اندازی تعداد زیادی سرور سخت‌افزاری در شبکه خود که مستلزم هزینه زیاد، هزینه و زحمات نگهداری بالا و مشکلات بسیار زیادی می‌باشد، یک یا تعداد بسیار کمتری سرور فیزیکی مناسب (با مشخصات سخت افزاری بالا) راه‌اندازی می‌نماییم و بر روی این سرورهای فیزیکی، سیستم‌عامل‌های مورد نیاز را به صورت ماشین‌های مجازی (virtual machine) راه‌اندازی می‌نماییم.

توجه: این ماشین‌های مجازی در حقیقت نسخه شبیه‌سازی شده نرم‌افزاری سیستم‌های سخت افزاری هستند.

ب) با استفاده از این تکنولوژی، بر روی هر یک از سرورهای فیزیکی تعداد زیادی سیستم‌عامل همزمان و مستقل از یکدیگر سرویس می‌دهند و اگر اختلال یا مشکلی در هر یک از این سیستم عامل‌ها روی دهد در دیگری تاثیری نمی‌گذارد.

به دلیل استفاده همزمان چندین سیستم عامل از سخت‌افزار یک سرور، این تکنولوژی «استفاده بهینه از سخت افزار» را برای ما مهیا می‌نماید.

۱- Application virtualization

مواقع زیادی این نیاز پیش می‌آید که که برای برخی از کاربران سازمان، امکان استفاده از نرم‌افزاهای سنگینی مثل photoshop، Autocad و غیره فراهم نماییم، اما سیستم فعلی آنها مشخصات سخت افزاری ضعیفی دارد و امکان نصب و استفاده از این نرم‌افزارهای سنگین در سیستم آنها وجود ندارد.

با راه اندازی تکنولوژی Application virtualization که یکی از سرویس های معروف مایکروسافتی آن سرویسی به نام RemoteApp میباشد، این نرم افزارها بر روی یک یا تعداد کمی سرور با مشخصات سخت افزاری بالا به روشی خاص نصب میکنیم و سپس نوع خاصی فایل shortcut از آن نرم افزار ایجاد کرده و آنها را در اختیار آن کاربران قرار میدهیم.

هنگامی که کاربران آن فایل shortcut را بر روی سیستم خود اجرا مینمایند، همگی به نرم افزاری که بر روی سرور نصب شده است متصل می‌شوند و در حقیقت آن نرم افزار بر روی سرور اجرا می‌شود و کاربر فقط صفحه کار با نرم افزار را بر روی سیستم خود مشاهده می‌کند (دقیقاً شبیه زمانی که نرم‌افزار بر روی سیستم خود

کاربر نصب شده بود). این تکنولوژی این قابلیت را ارایه می‌کند که نرم‌افزار همزمان چندین بار بر روی سرور اجرا شود.

درهنگام ذخیره فایل خروجی، کاربر می‌تواند آن فایل را بر روی سیستم محلی خود و یا در درایوهای سرور دخیره نماید.

۳- Desktop virtualization

در حالت سنتی، برای هر یک از کاربران سازمان یک کیس سخت‌افزاری مجزا تهیه می‌شود و سیستم عامل و نرم افزارهای مورد نیاز آن کاربر به صورت جداگانه بر روی این سیستم نصب می‌شوند.

ضمناً در صورت نیاز به ارتقا سخت افزار سیستم‌های کاربران در آینده، باید تک تک سیستم‌ها به صورت مجزا ارتقا داده شوند و قطعات سخت‌افزاری مورد نیاز درون کیس آنها بسته شود که این پروسه در تعداد بالا، فرایندی زمان بر و طاقت فرسا می‌باشد.

در تکنولوژی Desktop virtualization، برای کارمندان سازمان نیز virtual machineهایی در روی سرورهای سازمان راه اندازی می‌نماییم و مدیریت سخت افزارها و منابع آنها به صورت متمرکز و با سرعت و سهولت بسیار بیشتر از پشت سرور انجام می‌شود.

۲) NUMA یک حافظه توزیع شده است که در آن هر پردازنده یا هسته، به بخش‌های مختلف اختصاصی دسترسی دارد.

گزینه دوم پاسخ سوال نیست. زیرا Non-uniform Memory Access (به‌صورت مخفف NUMA) یک طراحی حافظه است که در این طرح هر processor (پردازنده) حافظه‌ی محلی مخصوص به خود را دارد و سرعت دسترسی پردازنده به حافظه محلی خود بالاتر از حافظه‌های غیرمحلی است.

NUMA (نوما) نقطه مقابل معماری SMP است که در آن تمام پردازنده‌ها از یک حافظه مشترک استفاده می‌کنند، مناسب پروسس‌هایی وابسته به یک کاربر یا task است، در نتیجه اجرای پروسس در یک نود و با یک RAM اختصاصی سرعت اجرا را بیشتر می‌کند. برای اجرای بهینه برنامه‌ها باید هر پروسس تا حد ممکن از RAM محلی پردازنده‌ای که در آن مقیم است استفاده کند.

۳) System Daemon یک برنامه سیستمی مقیم در حافظه است که در صورت لزوم به صورت ناهمگام اجرا می‌شود. گزینه سوم پاسخ سوال است، زیرا daemon یک برنامه کامپیوتری است که بعنوان یک فرآیند در background سیستم عامل اجرا می‌شود بدین معنی که در کنترل مستقیم کاربر نیست و کار خاصی را در زمان مشخص و یا در پاسخ به یک رویداد خاص بصورت تکراری و ناهمگام انجام می دهد. در لینوکس و البته دیگر سیستم عامل‌های چند وظیفه‌ای، مدام با اصطلاحی به نام Daemon مواجه می‌شوید، Daemon در مدیریت سرور لینوکس نقش دارد.

یک daemon فرآیندی است که مدت زمان زیادی در پس زمینه سیستم عامل در حال اجرا است تا به درخواست‌های سرویس‌ها پاسخ دهد. این اصطلاح بیشتر در لینوکس استفاده می‌شود. اما در سیستم عامل‌های دیگر مثل ویندوز و مکینتاش هم وجود دارد. در لینوکس به طور قراردادی در انتهای نام هر Daemon یک کاراکتر d هم وجود دارد. به عنوان مثالinetd ، nfsd، sshd، named و lpd از جمله دیمون‌ها هستند.

پس Daemon یک برنامه است که به عنوان یک فرآیند پشت صحنه اجرا می‌شود و در ارتباط مستقیم با کاربر نیست، یعنی شما آن برنامه را به صورت مستقیم نمی‌بینید. در محیط لینوکس فرآیند والد یک Daemon اغلب و نه همیشه، یک فرآیند init است. به همین دلیل است که عبارت init را همیشه در کنار Daemon خواهید شنید.

سیستم همیشه Daemon ها را در زمان بالا آمدن اجرا می‌کند تا هر کدام گوش به زنگ باشند تا کاری را انجام دهند. مثل پاسخ به درخواست‌های شبکه، فعالیت‌های سخت‌افزاری و برخی از فعالیت‌ها که مربوط به نرم‌افزارهای خاصی می‌شوند. Daemonها حتی می‌توانند پیکربندی‌های سخت افزاری(udevd) ، اجرای وظایف زمان‌بندی شده (cron) و دسته‌ای از وظایف دیگر را انجام دهند که تمامی آنها در پس زمینه سیستم شما انجام خواهند شد.

بدین ترتیب فهمیدیم که Daemonها برنامه‌های پشت پرده‌ای هستند که توسط خود سیستم در زمان اجرای کامپیوتر راه‌اندازی می‌شوند و هر کدام کاری را برای ماشین انجام می‌دهند.

۴) Emulation به فرآیند شبیه‌سازی یک سیستم عامل داخل سیستم عامل دیگر اطلاق

می‌شود.

گزینه چهارم پاسخ سوال نیست. زیرا اگر از نرم‌افزارهای Simulator یا شبیه‌ساز استفاده کرده باشید و یا با ابزارهای Emulator یا تقلید کننده آشنایی داشته باشید. شاید برای شما هم جالب باشد تفاوت بین این دو واژه را بیشتر بدانید. در وهله اول شما باید بدانید که این دو مفهوم با همدیگر تفاوت‌های اساسی دارند و نباید واژه‌های Emulator و Simulator به جای هم استفاده شود. دقت کنید زمانیکه صحبت از Simulator یا شبیه‌ساز می‌شود ما در خصوص یک سیستم صحبت می‌کنیم که هم می‌تواند نرم‌افزار و هم می‌تواند سخت‌افزاری باشد و این سیستم رفتارهایی بسیار نزدیک و شبیه به سیستم واقعی را دارد و در زمان استفاده از این Simulator شما تصور می کنید که در حال استفاده از سیستم اصلی هستید. اما نحوه پیاده‌سازی Simulator ها کاملا متفاوت است، در واقع شبیه‌سازها یا Simulator ها دقیقا از قوانین و رفتارهایی که سیستم واقعی دارد پیروی نمی‌کنند و برای خودشان قوانینی دارند که ممکن است به هیچ عنوان در سیستم واقعی شبیه‌سازی نشده اتفاق نیوفتد. در واقع زمانیکه صحبت از Simulator یا شبیه‌ساز می‌شود که شما در خصوص یک سیستم یا بهتر بگوییم در خصوص ایده و روش کارکرد یک سیستم صحبت می کنید و در خصوص جزئیات کامل کار کردن سیستم صحبتی ندارید.

نرم افزارهایی وجود دارند که شبیه‌ساز پرواز با هواپیما هستند، سخت‌افزارهایی هم وجود دارند که همین کار شبیه‌سازی پرواز را انجام می‌دهند، نرم افزارهایی وجود دارند که برای ما شبکه را شبیه‌سازی می کنند. اینگونه نرم‌افزارها یا سخت‌افزارها به شما این احساس را می‌دهند که در حال کار کردن با یک سیستم واقعی هستید و برای مثال یک هواپیما را از زمین بلند می کنید و بر روی زمین می‌نشانید یا سویچ‌ها و روترهای شبکه را پیکربندی می‌کنید و بین آنها ارتباط برقرار می کنید. اما این محیط‌های شبیه‌سازی شده کاملا از محیط واقعی جدا هستند و هیچ ارتباطی با محیط واقعی ندارند، شما در یک نرم‌افزار شبیه‌ساز پرواز می‌توانید یک هواپیما را بصورت کاملا سر و ته پرواز بدهید اما آیا واقعا در محیط واقعی هم می‌توانید اینکار را انجام دهید؟ شما در نرم افزارهایی مثل Packet Tracer یا NetSim روترها و سویچ های شبکه را شبیه‌سازی می‌کنید و بر روی آنها دستورات خود را وارد می‌کنید و در یک محیط شبیه‌سازی شده تست ارتباطی هم می گیرید، اما آیا می‌توانید درجه گرمایی که CPU روترها و سویچ های شما بعد از هر پیکربندی دارند را احساس کنید؟ اینگونه محیط‌ها بیشتر مصارف آموزشی و آشنایی با شکل کلی کار با محیط واقعی را دارند، هر چند واقعی هم طراحی شوند باز هم محیط شبیه‌سازی شده هستند و بروز مشکل در چنین محیط‌هایی هیچ تاثیری در محیط واقعی نخواهد داشت. بسیاری از دستورات و کارهایی که در محیط‌های شبیه‌سازی شده انجام می‌دهید ممکن است در محیط واقعی به درستی کار نکند.

زمانیکه صحبت از Emulator یا مقلد می‌شود در خصوص یک سیستم کاملا مشابه با آنچه در محیط واقعی است صحبت می‌کنیم. این سیستم تقلیدکننده دقیقا همان قوانینی را دارد که در

سیستم واقعی وجود دارد و می‌توان از آن Copy و Paste سیستم اصلی یاد کرد. نرم‌افزارهای Emulator حتی سورس کد مشابه و در اصطلاح Clone سیستم اصلی هستند، شما هر کاری که در محیط واقعی با نرم‌افزار می‌توانید انجام بدهید در محیط Emulator هم می‌توانید عینا انجام دهید. حتی ورودی و خروجی نرم‌افزار و روش پردازش آن نیز کاملا شبیه به محیط اصلی نرم افزار است، تنها تفاوت در محیط کاری است که بر روی آن نرم‌افزار اجرا می‌شود که طبیعتا محیطی به غیر از محیط اصلی اجرای نرم افزار اصلی خواهد بود. قوانین سیستم اصلی و سیستم emulator کاملا شبیه به هم هستند و غیرقابل تغییر هستند. برای مثال شما زمانیکه می‌خواهید نرم افزاری برای سیستم عامل اندروید بنویسید می‌توانید از Emulator ای به نام BlueStacks استفاده کنید. BlueStacks یک نرم‌افزار شبیه‌ساز نیست زیرا عینا برای شما سیستم‌عامل اندروید را روی ویندوز نمایش می‌دهد و تمامی قوانین اندروید را بر روی آن قرار می‌دهد. شما نرم افزارهای اندرویدی خود را برای تست می توانید در این محیط نصب کنید و اجرا کنید Emulator های بازی هم به همین شکل عمل می کنند، ممکن است شما بازی‌های PlayStation یا Xbox را بتوانید بصورت کامل در سیستم شخصی خودتان با استفاده از یک Emulator اجرا کنید. در واقع در اینجا شما یک کپی اصلی از بازی اصلی گرفته‌اید و این Emulator یا مقلد است که امکان اجرای آن بر روی سیستم اصلی را می دهد. حتی Bug ها و مشکلات و Crash هایی که ممکن است در سیستم اصلی به وجود بیاید عینا در سیستم Emulate شدن نیز به وجود می آید زیرا سورس یا هسته نرم افزار به هیچ عنوان تغییر نکرده است .

بصورت کلی زمانیکه صحبت از Emulator می شود در خصوص یک کپی از سیستم اصلی صحبت می شود که عین دستگاه و سیستم واقعی کار می کند. اما زمانیکه صحبت از Simulator یا شبیه‌ساز می شود در واقع در خصوص یک سیستم مدل‌سازی یا Modeling از نرم‌افزار یا سیستم اصلی صحبت می‌کنیم. توجه کنید که همیشه قرار نیست یک سیستم Simulator چیزی شبیه به یک سیستم Emulator شود. اگر بخواهیم مثال شبکه ای بزنیم که شما بیشتر برایتان این موضوع جا بیوفتند می‌توانیم بگوییم نرم‌افزار مثل Packet Tracer یک نرم‌افزار شبیه ساز یا Simulator است اما نرم افزاری مثل GNS3 یک نرم‌افزار Emulator است زیرا در GNS3 شما سیستم عامل اصلی روترها و سویچ ها را بر روی نرم‌افزار Emulator نصب می‌کنید و آن را اجرا می کنید. از نظر سرعت، سرعت Emulator ها به مراتب کندتر از Simulator ها است.

تست‌های فصل چهارم

۷۵- سیستمی با ترجمه آدرس دو-سطحی و اندازه هر صفحه ۴ کیلوبابت در نظر بگیرید. اگر اندازه هر مدخل جدول صفحه برابر ۲ بایت (شامل اطلاعات ترجمه و دیگر اطلاعات کنترلی لازم) باشد، چه تعداد فضای بیتی به ترتیب (از راست به چپ) برای جا به جایی (Offset)، اندیس به جدول صفحه اول و اندیس به جدول صفحه دوم برای آدرس مجازی (Virtual Address) ۳۲-بیتی لازم است؟

(مهندسی کامپیوتر – دولتی ۹۷)

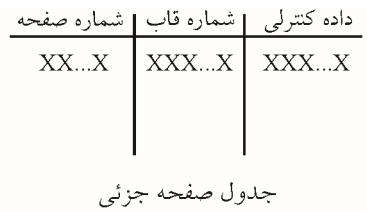
۱) ۱۲، ۱۰، ۱۰ ۲) ۱۲، ۱۱، ۹ ۳) ۱۰، ۱۰، ۱۲ ۴) ۹، ۱۱، ۱۲

پاسخ‌های فصل چهارم

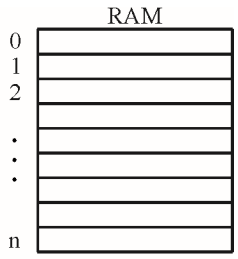
۷۵- گزینه () صحیح است.

در اینجا برای جدول صفحه جزئی محدودیتی به اندازه یک قاب (صفحه) داریم.

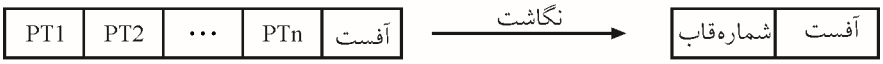
بنابراین اندازه جدول صفحه جزئی برابر اندازه قاب (صفحه) می‌باشد. بنابراین برای محاسبه تعداد سطرهای جدول صفحه جزئی، کافی است، اندازه قاب که برابر اندازه جدول صفحه جزئی است بر اندازه عرض جدول صفحه جزئی تقسیم گردد. به شکل زیر توجه کنید:



جدول صفحه جزئی



حافظه فیزیکی



آدرس منطقی(مجازی) آدرس فیزیکی

توجه: عرض جدول صفحه همواره برابر حاصل جمع تعداد بیت‌های کنترلی و تعداد بیت‌های شماره قاب است، دقت کنید که تعداد بیت‌های شماره صفحه جزو عرض جدول صفحه نمی‌باشد، بلکه شماره صفحه، اندیس هر سطر جدول صفحه می‌باشد.

بنابراین داریم:

تعداد بیت‌های کنترلی + تعداد بیت‌های شماره قاب= عرض جدول صفحه جزئی

$2B$ = عرض جدول صفحه جزئی

توجه: مطابق فرض سؤال، هر مدخل جدول صفحه (عرض جدول صفحه جزئی) 2 بایت در نظر گرفته شده است.

$$\text{تعداد سطرهای جدول صفحه جزئی} = \frac{\text{اندازه قاب}}{\text{عرض جدول صفحه}} = \frac{2^2 \times 2^{10}\,B}{2^1\,B} = 2^{11} = 2048$$

$2^{32}\,B$ =اندازه فرآیند (فضای آدرس مجازی)

$4KB = 4096B = 2^2 \times 2^{10}\,B = 2^{12}\,B$ =اندازه صفحه = اندازه قاب

$$f: \text{تعداد صفحات فرآیند} = \frac{\text{اندازه فرآیند}}{\text{اندازه صفحه}} = \frac{2^{32}}{2^{12}} = 2^{20}$$

r : تعداد سطرهای جدول صفحه جزئی $= 2048 = 2^{11}$

توجه: دقت کنید و لطفا یاد بگیریم که اندازه فرآیند(فضای آدرس مجازی) مطابق اطلاعات صورت سوال برابر $2^{32}\,B$ است.

توجه: در اینجا برای جدول صفحه جزئی محدودیتی به اندازه یک قاب (صفحه) داریم. اما چرا این محدودیت وجود دارد؟ زیرا کل جدول صفحه در حالت تک سطحی داخل یک قاب جا نمی‌شود در واقع اندازه جدول صفحه تک سطحی از اندازه یک قاب بیشتر است. بنابراین مطابق آنچه گفتیم رابطه‌ی زیر برقرار است:

$2^{32}\,B$ =اندازه فرآیند (فضای آدرس مجازی)

$4KB = 4096B = 2^2 \times 2^{10}\,B = 2^{12}\,B$ =اندازه صفحه = اندازه قاب

$$\text{تعداد درایه‌های جدول صفحه تک سطحی=تعداد صفحات فرآیند} = \frac{\text{اندازه فرآیند}}{\text{اندازه صفحه}} = \frac{2^{32}}{2^{12}} = 2^{20}$$

توجه: عرض جدول صفحه همواره برابر حاصل جمع تعداد بیت‌های کنترلی و تعداد بیت‌های شماره قاب است، دقت کنید که تعداد بیت‌های شماره صفحه جزو عرض جدول صفحه نمی‌باشد، بلکه شماره صفحه، اندیس هر سطر جدول صفحه می‌باشد.

بنابراین داریم:

تعداد بیت‌های کنترلی + تعداد بیت‌های شماره قاب= عرض جدول صفحه جزئی

$2B$ = عرض جدول صفحه جزئی

اندازه جدول صفحه تک سطحی و بدون استفاده از جداول صفحه چندسطحی
$2^{21}B = 2^{20} \times 2B$ =عرض جدول صفحه ×تعداد درایه‌های جدول صفحه تک سطحی=
اندازه قاب <اندازه جدول صفحه تک سطحی

$2^{21}B > 2^{12}B$

بنابراین واضح است که اندازه جدول صفحه تک سطحی از اندازه قاب **بیشتر و بزرگتر** است، پس با **محدودیت قاب** مواجه هستیم و باید به سمت طراحی جداول صفحه چندسطحی حرکت کنیم.

توجه: اغلب سیستم‌های کامپیوتری، فضای آدرس منطقی(مجازی) بزرگی را پشتیبانی می‌کنند، مانند کامپیوترهایی که آدرس‌های 32 یا 64 بیتی را پشتیبانی می‌کنند، در چنین محیط‌هایی جدول صفحه معمولی بسیار بزرگ خواهد شد. برای مثال یک سیستم را با 32 بیت فضای آدرس منطقی(مجازی) در نظر بگیرید. اگر اندازه صفحه در این سیستم برابر با 4KB (2^{12}) باشد، در اینصورت جدول صفحه شامل بیش از یک میلیون درایه خواهد بود. $\left(2^{32} / 2^{12} = 2^{20}\right)$. فرض کنید هر درایه جدول صفحه معمولی، شامل 4 بایت باشد، در اینصورت هر فرآیند به $\left(4 \times 2^{20}\right)$ 4MB فضای آدرس فیزیکی فقط برای نگهداری جدول صفحه معمولی نیاز دارد. همچنین مشخص است که با توجه به محدودیت اندازه قاب (4KB) نمی‌توان این جدول صفحه معمولی را بصورت پیوسته درون یک قاب جا داد. 4MB اندازه جدول صفحه معمولی در فضای 4KB مربوط به یک قاب جا نمی‌شود. یک راه حل تقسیم جدول صفحه معمولی به جداول صفحه جزئی و ایجاد جدول صفحه چندسطحی است، در این حالت جدول صفحه معمولی نیز همانند فرآیند صفحه‌بندی می‌شود.

حال اطلاعات کافی برای محاسبه تعداد بیت PT1 یعنی اندیس به جدول صفحه اول و PT2 یعنی اندیس به جدول صفحه دوم را در اختیار داریم:

روش تجزیه

تعداد صفحات فرآیند باید در اندازه (2^{11})r تجزیه گردد.

$$\text{تعداد صفحات فرآیند} = 2^{20} = 2^{9\nearrow PT1} \times 2^{11\nearrow PT2} = \text{تعداد صفحات فرآیند}$$

روش لگاریتم

$$\text{تعداد سطوح جدول چند سطحی} = d = \left\lceil \log_r^f \right\rceil = \left\lceil 10g_{2^{11}}^{2^{20}} \right\rceil = 2$$

روش تقسیم متوالی

$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{2^{20}}{2^{11}} = 2^9$$

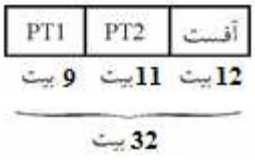
$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول صفحه جزئی در سطح دوم}}{r} = \frac{2^9}{2^{11}} < 1$$

توجه: سطح اول، یک جدول به حساب می‌آید، که 2^9 سطر دارد.

توجه: تعداد تقسیم متوالی برابر 2 است، بنابراین تعداد سطوح جدول صفحه چند سطحی برابر 2 است.

$$\text{تعداد بیت آفست} = \log_2^{\text{اندازه صفحه}} = \log_2^{2^{12}} = 12 \text{ بیت}$$

بنابراین شکل آدرس منطقی (مجازی) به صورت زیر خواهد بود.



مطابق آنچه گفتیم دو دسترسی به جداول صفحه جزئی سطح اول و دوم برای ترجمه آدرس و یک دسترسی به داده اصلی (مقصد) لازم است، که مجموع آن شامل سه دسترسی به حافظه می‌گردد. بنابراین نیاز به سه دسترسی به حافظه است.

یک بار دیگر صورت سوال را به دقت بررسی کنید، سیستمی با ترجمه آدرس دو-سطحی و اندازه هر صفحه ۴ کیلوبابت در نظر بگیرید. اگر اندازه هر مدخل جدول صفحه برابر ۲ بایت (شامل اطلاعات ترجمه و دیگر اطلاعات کنترلی لازم) باشد، چه تعداد فضای بیتی به ترتیب (از راست به چپ) برای جا به جایی (Offset)، اندیس به جدول صفحه اول و اندیس به جدول صفحه دوم برای آدرس مجازی (Virtual Address) ۳۲-بیتی لازم است؟

(مهندسی کامپیوتر – دولتی ۹۷)

۱) ۱۲، ۱۰، ۱۰ ۲) ۱۲، ۱۱، ۹ ۳) ۱۰، ۱۰، ۱۲ ۴) ۹، ۱۱، ۱۲

با توجه به شرایط صورت سوال و مفروضات مساله گزینه اول نمی‌تواند طراحی شود. اگر می‌توانید طراحی کنید؟

اگر مصلحت و صلاح دانشجویانی که یکسال مطالعه کرده‌اند را در نظر بگیریم و **اندیس به جدول صفحه اول و اندیس به جدول صفحه دوم مطرح شده در صورت سوال را به ترتیب PT2 و PT1 در نظر بگیریم، مطابق آنچه بارها در پیام‌ها مطرح کردیم آنگاه گزینه دوم می‌تواند پاسخ منطقی برای سوال باشد، در غیر اینصورت و درحالت سختگیرانه این سوال پاسخ درستی ندارد و باید حذف شود.**

اما ...

دوستانی می‌گویند که گزینه اول می‌تواند درست باشد...

روش تقسیم متوالی

$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{2^{20}}{2^{11}} = 2^9$$

$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول صفحه جزئی در سطح دوم}}{r} = \frac{2^9}{2^{11}} < 1$$

توجه: مطابق فرض سؤال، هر مدخل جدول صفحه (عرض جدول صفحه جزئی) 2 بایت در نظر گرفته شده است.

$$\text{تعداد سطرهای جدول صفحه جزئی} = \frac{\text{اندازه قاب}}{\text{عرض جدول صفحه}} = \frac{2^2 \times 2^{10}B}{2^1 B} = 2^{11} = 2048$$

توجه: در این شرایط و رسیدن به گزینه اول، شما برای رسیدن به 2^{10}، باید مقدار عرض جدول صفحه در سطح دوم که این عرض فقط و فقط شامل حاصل جمع تعداد بیت‌های کنترلی و تعداد بیت‌های شماره قاب در سطح دوم است را برابر $2^2 B$ در نظر بگیرید که این فرض کاملا نادرست است و به تبع رابطه زیر را بر قرار کنید:

$$\text{تعداد سطرهای جدول صفحه جزئی} = \frac{\text{اندازه قاب}}{\text{عرض جدول صفحه}} = \frac{2^2 \times 2^{10}B}{2^2 B} = 2^{10} = 1024$$

که امکان پذیر نیست. اگر می‌توانید طراحی کنید...

توجه: عرض جدول صفحه همواره برابر حاصل جمع تعداد بیت‌های کنترلی و تعداد بیت‌های شماره قاب است، دقت کنید که تعداد بیت‌های شماره صفحه جزو عرض جدول صفحه نمی‌باشد، بلکه شماره صفحه، اندیس هر سطر جدول صفحه می‌باشد.

بنابراین داریم:

تعداد بیت‌های کنترلی + تعداد بیت‌های شماره قاب= عرض جدول صفحه

$2B$ = عرض جدول صفحه

تست‌های فصل پنجم

۷۶- در یک سیستم متشکل از ۴ قاب که در ابتدا خالی هستند، رشته دستیابی به قاب‌ها را به ترتیب از چپ به راست 1, 2, 3, 4, 5, 6, 2, 3, 2, 1, 6, 7 در نظر بگیرید. اگر سیستم صفحه‌بندی تماما مبتنی بر درخواست (pure demand paging) باشد. در صورت استفاده از الگوریتم‌های FIFO و LRU به ترتیب (از راست به چپ) تعداد نقص صفحه (page fault)، کدام است؟

(مهندسی کامپیوتر – دولتی ۹۷)

۱) ۱۰، ۱۰ ۲) ۱۰، ۹ ۳) ۶، ۶ ۴) ۶، ۵

۷۶- گزینه (۱) صحیح است.

الگوریتم LRU (Least Recently Used)

ایده اصلی این الگوریتم این است که اگر صفحه‌ای در چند دستور اخیر مراجعات زیادی داشته است، به احتمال قوی در دستورات بعدی هم ارجاعات زیادی خواهد داشت، همچنین اگر یک صفحه، اخیرا هیچ مراجعه‌ای نداشته، احتمالاً در آینده نزدیک هم ارجاعی نخواهد داشت. در واقع این الگوریتم بیان می‌کند هنگام وقوع خطای نقص صفحه، صفحه‌ای را حذف کنید که طولانی‌ترین زمان عدم استفاده را دارد.

می‌توان گفت LRU تقریبی از الگوریتم بهینه می‌باشد که در آن به جای توجه به آینده، به گذشته توجه می‌شود.

توجه: الگوریتم LRU ناهنجاری بی‌لیدی ندارد، بنابراین با افزایش تعداد قاب، همواره، نقص صفحه کاهش می‌یابد.

اگر 4 قاب در نظر گرفته شود:

نگاه به گذشته ←

رشته مراجعات	1	2	3	4	5	6	2	3	2	1	6	7								
قاب 1	1	1	1	1	5	5	5	5	5	1	1	1								
قاب 2		2	2	2	2	6	6	6	6	6	6	6								
قاب 3			3	3	3	3	2	2	2	2	2	2								
قاب 4				4	4	4	4	3	3	3	3	7								
نقص صفحه	*	*	*	*	*	*	*	*		*		*								

در این حالت 10 نقص صفحه به وقوع پیوست.

الگوریتم FIFO (First In First Out)

این الگوریتم ساده‌ترین الگوریتم از نظر پیاده‌سازی است. در این روش سیستم عامل لیستی از صفحات را به ترتیب ورود به حافظه نگه می‌دارد. وقتی یک خطای نقص صفحه رخ می‌دهد، سیستم عامل قدیمی‌ترین صفحه را برای بیرون رفتن انتخاب می‌کند. ایده این روش این است که قدیمی‌ترین صفحه شانس مورد استفاده قرار گرفتن را به اندازه کافی در اختیار داشته و اکنون باید این شانس به صفحه دیگری داده شود.

نکته: نقص الگوریتم FIFO این است که حتی اگر صفحه‌ای بارها و به طور مکرر استفاده شود، سرانجام به قدیمی‌ترین صفحه تبدیل و حذف می‌شود، در صورتی که احتمالاً بلافاصله باید

دوباره به حافظه آورده شود.

رشته مراجعات	1	2	3	4	5	6	2	3	2	1	6	7								
قاب 1	1	1	1	1	2	3	4	5	5	6	6	2								
قاب 2		2	2	2	3	4	5	6	6	2	2	3								
قاب 3			3	3	4	5	6	2	2	3	3	1								
قاب 4				4	5	6	2	3	3	1	1	7								
نقص صفحه	*	*	*	*	*	*	*	*		*		*								

در این حالت 10 نقص صفحه به وقوع پیوست.