نمونه سوالات فصل نهم

۱ـ به جای بارگزاری تمام برنامه در حافظه می‌توان بخش‌هایی از برنامه‌ها را که می‌خواهند اجرا شوند در حافظه بارگزاری کنیم. این روش چه مزایایی دارد؟

پاسخ : صفحه ۳۹۸

- برنامه دیگر توسط مقدار حافظه سخت‌افزاری سیستم محدود نمی‌شود. کاربران می‌توانند از فضای آدرس مجازی بزرگی استفاده کنند.
- به این خاطر که هر برنامه به فضای کم‌تری از حافظه احتیاج دارد می‌توانیم برنامه‌های بیش‌تر را به طور هم‌زمان در حافظه قرار دهیم که باعث افزایش کارایی CPU و throughput می‌شود اما روی زمان پاسخ و turnaround time تاثیری ندارد.
- مقدار I/O کم‌تری برای بارگزاری یا جابجایی[1]برنامه‌ها در حافظه نیاز است، در نتیجه هر برنامه کاربر سریع‌تر اجرا می‌شود.

۲ـ سه مزیت استفاده از حافظه مجازی (صفحه‌بندی) را توضیح دهید.

پاسخ:

- ایزوله کردن حافظه هر پردازه از دسترسی غیرمجاز دیگر پردازه‌ها (در صورت عدم استفاده از حافظه مشترک)
- ساختار ساده برای فضای آدرس مجازی پردازه‌ها(یک فضای آدرس پیوسته و مشابه برای تمام پردازه‌ها)
- ایجاد امکان استفاده از Swapping

۳ـ Demand Paging، Swapping و Copy-on-Write چگونه در مصرف حافظه فیزیکی صرفه‌جویی می‌کنند؟

پاسخ: صفحات ۴۰۱، ۴۰۲ و ۴۰۸

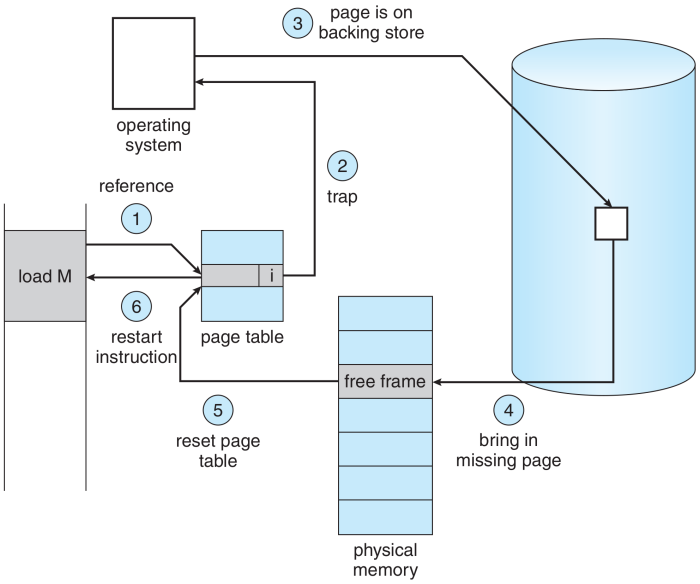
در Demand Paging تا زمان که به یک بخش از فضای آدرس مجازی پردازه دسترسی به وجود نیاید، صفحه مربوط به آن وارد حافظه نخواهد شد. به صفحه‌ای که به حافظه آورده شده است، Memory Resident می‌گویند. یک Lazy Swapper، هنگام انتقال حافظه پردازه به حافظه فیزیکی به منظور اجرا، تنها بخشهای مورد نیاز را از دیسک دریافت می‌کند. همچنین در صورت که حافظه کافی در دسترس نیست، طبق الگوریتم از پیش تعیین شده، محتوای فریم از حافظه فیزیکی به دیسک منتقل می‌شود. Copy-on-Write، سازوکاری است که در فراخوان سیستمی fork استفاده می‌شود. پس از ایجاد پردازه فرزند، صفحه‌های حافظه والد با فرزند به اشتراک گذاشته می‌شود. یعنی آدرس مجازی متفاوت و آدرس فیزیکی یکسان خواهد بود. تنها نیاز به ایجاد فضای جدید برای جدول صفحه فرزند است. در صورت که والد یا فرزند تغییری در محتوای صفحه‌های مشترک ایجاد کنند صفحه جداگانه‌ای برای پردازه تغییردهنده، تخصیص خواهد یافت.

[1] بین دیسک و حافظه نه جابجایی فریم‌ها در حافظه

۴ـ مراحل رسیدگی به نقص صفحه را شرح دهید.

پاسخ: ص ۴۰۳

شش مرحله ذکر شده در کتاب. خلاصه آن در شکل ۱ نشان داده شده است.



شکل ۱: مراحل مدیریت نقص صفحه

۵ـ رابطه مربوط به زمان مؤثر دسترسی به حافظه در Demand Paging را نشان دهید. روشهای بهبود کارایی Demand Paging را توضیح دهید.

پاسخ: ص ۴۰۷

اگر p احتمال نقص صفحه باشد، مطابق رابطه ۱ داریم:

$$EffectiveAccessTime = (1 - p) \times MemoryAccessTime + p \times PageFaultServiceTime$$

با توجه به این که فضای Swap نسبت به فایل سیستم، سریع‌تر است می‌توان با کپی کردن کل برنامه در این فضا هنگام آغاز اجرای برنامه و Demand Paging از آن کارایی را بالا برد.

می‌توان صفحه‌ها را ابتدا از فایل سیستم به صورت Demand Paging دریافت نموده و سپس در حین اجرا و در صورت نیاز در فضای Swap کپی نمود. به این ترتیب تنها صفحه‌های مورد نیاز از فایل سیستم دریافت شده و Pagingهای بعدی از فضای Swap انجام خواهد شد.

همچنین می‌توان هنگام Page Replacement، تنها فریم‌های مربوط به بخش‌هایی از حافظه مجازی برنامه مانند پشته و هیپ که فایل متناظری ندارند (Anonymous Memory) در فضای Swap کپی شوند. بخش‌های دیگر مانند کد برنامه با توجه به وجود یک کپی از آنها روی دیسک ، Overwrite شده و نیازی به نوشتن آنها در فضای Swap نخواهد بود.

۶ـ دلیل استفاده از modify بیت یا dirty بیت چیست؟

پاسخ : صفحه ۴۱۱

شرایطی را در نظر بگیرید که در آن حافظه پر شده است و پردازه‌ای درخواستی داده که برای برآورده کردن آن باید یک صفحه از فضای swap در حافظه بارگزاری شود. در این شرایط باید یکی از صفحات حافظه را به فضای swap منتقل کرده تا فضای لازم برای بارگزاری صفحه جدید را فراهم کنیم. به این نکته توجه داشته باشید که برای این کار به دوبار جابجایی صفحات بین حافظه و فضای swap احتیاج داریم (یک‌بار نوشتن در آن و یک‌بار خواندن) که زمان effective access و page-fault service را تحت تاثیر قرار می‌دهد.

می‌توانیم این سربار را با معرفی modify بیت (یا dirty بیت) بهبود دهیم. به این صورت که برای هر صفحه در سخت‌افزار حافظه یک بیت در نظر گرفته می‌شود و هنگامی که دستور نوشتن روی صفحه‌ای را دریافت کردیم، سخت‌افزار modify بیت آن صفحه را یک می‌کند تا نشان دهد، این صفحه تغییریافته است.

هنگامی که شرایط گفته شده پیش آمد و خواستیم صفحه مورد نظر را به فضای swap منتقل کنیم، به modify بیت آن نگاه می‌کنیم، اگر این بیت صفر بود نیازی نداریم آن را به فضای swap منتقل کنیم، زیرا این صفحه در فضای swap موجود است. این روش زمان مورد نیاز برای مدیریت page fault را به میزان چشمگیری کاهش می‌دهد، زیرا اگر modify بیت صفر باشد زمان I/O نصف می‌شود.

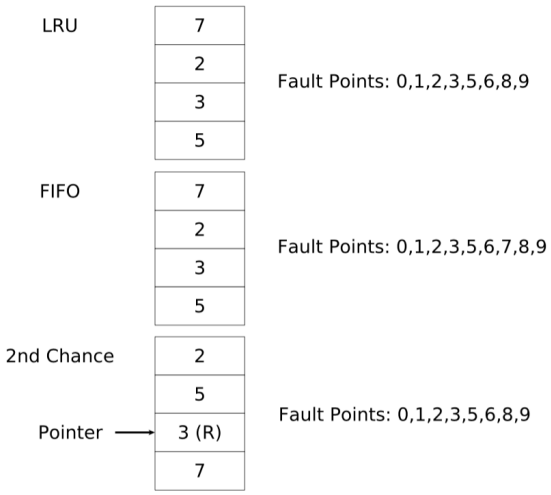
۷ـ مجموعه ارجاعات زیر را در نظر بگیرید:

1,3,8,7,3,4,5,3,2,7

چه تعداد نقص صفحه با فرض وجود چهار فریم و استفاده از الگوریتم‌های LRU، FIFO و Second Chance رخ خواهد داد؟

پاسخ: ص ۴۱۳، ۴۱۶ و ۴۱۸

صف نهایی فریم‌های تخصیص شده و زمان‌های نقص صفحه در شکل ۲ نشان داده شده است. در دو الگوریتم نخست، عنصر پایین صف و در الگوریتم آخر، اولین عنصر بدون ارجاع با شروع از اشاره‌گر، اولویت خروج دارند.



شکل دوم : نتایج الگوریتم جایگزینی صفحه

۸ـ سه نمونه Page Buffering را توضیح دهید.

پاسخ: ص ۴۲۰

می‌توان مکمل‌هایی را به الگوریتم جایگزینی صفحه‌ها افزود:

- یک استخر از فریم‌های آزاد به طور جداگانه نگهداری خواهد شد. هنگام نیاز به دریافت صفحه از دیسک و جایگزینی آن با یک فریمی موجود در حافظه فیزیکی، این جایگزین صورت نمی‌گیرد. بلکه صفحه از دیسک به یک فریم آزاد انتقال می‌یابد. نوشتن فریمی که قرار است به دیسک برود سر فرصت صورت گرفته و در نهایت فریم آزاد شده، به استخر آزاد انتقال می‌یابد. به این ترتیب پردازه، منتظر انتقال محتوای فریم به دیسک نمی‌ماند.
- می‌توان حالت یک را بهبود داد. به این ترتیب که پیش از نیاز به جایگزینی ، فریم‌های Dirty را با دیسک ، همگام نمود. این همگام‌سازی را می‌توان در مواقع که دیسک ، بی‌کار است انجام داد.
- یک بهبود دیگر روی حالت یک، این است که صفحه حافظه مجازی مربوط به فریمهایی که پس از همگام‌سازی به استخر آزاد فرستاده می‌شود به خاطر سپرده شود. به این ترتیب در صورت نیاز مجدد به آن بخش از حافظه مجازی، محتوای آن در فریم آزاد مربوطه قرار خواهد داشت. یعنی در صورت نقص صفحه به استخر آزاد رجوع می‌شود. اگر فریم مربوطه، وجود نداشت از دیسک خوانده خواهد شد.

۹ـ تقسیم بندی الگوریتم‌های جایگزینی صفحات (page replacement) به صورت سراسری یا محلی را مقایسه کنید.

پاسخ: صفحه ۴۲۴

می‌توانیم الگوریتم‌های جایگزینی صفحات را به دو خانواده کلی سراسری و محلی تقسیم کنیم. جایگزینی سراسری به پردازه این اجازه را می‌دهد تا فریم مناسب برای جایگزینی را از بین تمام فریم‌های موجود سیستم انتخاب کند، حتی اگر آن‌ها متعلق به پردازه‌ای دیگر باشد. در جایگزینی محلی پردازه باید فریم مناسب برای جایگزینی را از فریم‌های خودش انتخاب کند.

به عنوان مثال فرض کنید در یک سیستم با جایگزینی سراسری به پردازه‌ی پراولویت اجازه داده شده، تا فریم‌های پردازه‌های کم اولویت را تصاحب کند. در این سیستم یک پردازه پراولویت هنگام جایگزینی می‌تواند یکی از فریم‌های خودش یا یک فریم از پردازه‌های کم اولویت را انتخاب کند. در روش‌های سراسری جایگزینی صفحات به پردازه‌ها این امکان داده شده است تا تعداد فریم‌های در دسترس خود را افزایش دهند، اما چنین امکانی در سیستم‌های محلی وجود ندارد و پردازه نمی‌تواند فریم دیگری غیر از فریم‌های خود برای جایگزینی را انتخاب کند در نتیجه تعداد فریم‌های پردازه ثابت می‌ماند.

۱۰ـ تخصیص فریم به صورت Proportional چه مشکلی برای پردازه‌های پراولویت ایجاد می‌کند؟ راه حل چیست؟

پاسخ: ص ۴۲۴

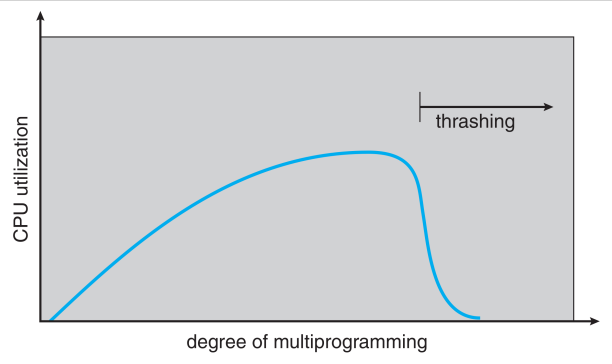
این روش تخصیص، تمایزی میان پردازه‌های پراولویت و کم اولویت قائل نمی‌شود. برای حل این مشکل می‌توان نسبت فریم‌های تخصیص به هر پردازه را به صورت تابعی از اولویت آن یا ترکیب اولویت و اندازه حافظه مجازی آن تعریف نمود.

۱۱ـ Thrashing چیست؟ علت وقوع آن را توضیح دهید.

پاسخ: ص ۴۲۵

پردازه‌ای که میزان Paging آن نسبت به میزان اجرایش بیش‌تر باشد، دچار Thrashing شده است. علت این است که تعداد صفحه‌های در حال استفاده فعال آن از تعداد صفحه‌های در دسترس، بیش‌تر است.

فرض می‌شود تخصیص فریم‌ها سراسری بوده و زمان‌بند بر اساس بهره‌وری پردازنده، درجه چندبرنامگی را تعیین می‌کند. اگر برنامه‌ای پس از مدتی اجرا نیاز به فریم‌های بیش‌تری پیدا کرده و فریم خالی موجود نباشد، نقص صفحه رخ داده و محتوای فریم از پردازه دیگر (که خود به فریمهایش احتیاج دارد) به دیسک منتقل می‌شود. به این ترتیب هر دو پردازه به دیسک، درخواست داده‌اند (اولی برای پر کردن فریم و دومی برای خالی کردن). پردازه دوم نیز چون به فریمش احتیاج دارد برای پر کردن فریم، درخواست به دیسک داده و محتوای فریم از پردازه سوم، به دیسک منتقل خواهد شد. به همین ترتیب بار دیسک افزوده شده و بار پردازنده کاهش می‌یابد. زیرا پردازه‌ها نیاز به ورودی/خروجی داشته و این به کاهش بهره‌وری پردازنده منجر خواهد شد. زمان‌بند با مشاهده کاهش بهره‌وری پردازنده، با زمان‌بندی پردازه‌ای جدید، بار دیسک را افزایش داده و بهره‌وری همچنان کاهش می‌یابد. در شکل ۳، پس از افزایش درجه چندبرنامگی از حد آستانه‌ای، Thrashing یا کوبیدگی رخ داده است.



شکل سوم :کوبیدگی

۱۲ـ چگونه می‌توان از Thrashing اجتناب نمود؟ آیا تخصیص محلی فریم‌ها، مشکل کندی سیستم را حل می‌کند؟

پاسخ: ص ۴۲۷ و ۴۲۹

روش اول، استفاده از Working Set Model است. پارامتر Δ، پنجره Working Set را تعیین نموده و تخمینی از محلیت برنامه است. یعنی مجموعه Δ ارجاع اخیر به صفحه‌ها توسط پردازه i، Working Set آن پردازه یا WSS_i نام دارد. سیستم عامل به اندازه WSS هر پردازه به آن حافظه اختصاص می‌دهد. اگر مجموع WSSها از اندازه حافظه فیزیکی ، تجاوز نمود، یک پردازه، معلق می‌شود. یعنی محتوای فریم‌های آن در دیسک نوشته شده و خود فریم‌ها به دیگر پردازه‌ها اختصاص خواهد یافت. دشواری این روش، در تعیین WSS در حین اجرا است که روش‌هایی برای تخمین آن وجود دارد.

یک روش ساده‌تر، استفاده از فرکانس نقص صفحه است. هرچه نرخ نقص صفحه افزایش یافت نیاز به فریم بیش‌تری است. اگر نرخ نقص صفحه، بیش از حد کم باشد، یعنی فریم‌هایی اضافه هستند. بر این اساس، می‌توان یک حد بالا و پایین برای نرخ نقص صفحه تعیین نمود. اگر نرخ نقص صفحه، از حد پایین، کم‌تر شد می‌توان یک فریم از پردازه دریافت نمود و اگر از حد بالا، بیش‌تر شد یک فریم در اختیار پردازه قرار خواهد گرفت. همچنین اگر نرخ نقص صفحه از حد بالا، بیش‌تر شد و فریم آزادی وجود نداشت یک پردازه معلق شده و فریم‌هایش آزاد می‌گردد. سوال این است که آیا می‌توان این فریم‌ها را میان پردازه‌های با نرخ نقص صفحه بالا توزیع نمود؟

خیر، با وجود آن که فریم دیگر پردازه‌ها دزدیده نمی‌شود، در صورت که پردازه‌های متعددی در حال Thrashing باشند، صف درخواستهای دیسک طولانی می‌شود. لذا پردازه‌هایی که نقص صفحه چندان متحمل نمی‌شوند نیز تأثیر منفی می‌پذیرند. زیرا نقص صفحه آن‌ها با وجود حذف Thrashing، کندتر خواهد شد.

۱۳ـ Locality Model اجرای پردازه‌ها چیست؟

پاسخ: ص ۴۲۷

طبق این مدل، هر پردازه در حین اجرا از Locality ها متعددی عبور می‌کند. هر Locality، مجموعه‌ای از صفحه‌ها است که با هم استفاده می‌شوند. این مجموعه‌ها ممکن است اشتراک داشته باشند. مثلاً فراخوانی یک تابع می‌تواند یک Locality جدید ایجاد کند که شامل صفحه‌های حافظه مربوط به کد تابع، متغیرهای محلی آن و بخشی از متغیرهای سراسری است و خروج از تابع، منجر به خروج از Locality خواهد شد.

۱۴ـ شباهت‌ها و تفاوت‌های Memory Mapped File و Memory Mapped I/O را برشمرید.

پاسخ: ص ۴۳۰ و ۴۳۵

در هر دو روش، یک نگاشت در بخشی از فضای آدرس مجازی پردازه ایجاد می‌شود. این نگاشت، امکان دسترسی به فایل از طریق حافظه مجازی (در Memory Mapped File، دسترس حافظه‌ای بدون نیاز به فراخوان عملیات فایل سیستم مانند فراخوانی‌های سیستمی read و write) را فراهم می‌آورد که منجر به بهبود کارایی دسترسی خواهد شد.

کاربرد Memory Mapped File، در دسترسی به فایل‌ها است. این دسترسی می‌تواند خواندن از فایل‌های معمولی، خواندن کد برنامه و دسترسی مشترک (از طریق ایجاد حافظه مشترک) به کتابخانه‌های پویا باشد. همچنین ممکن است اجزایی از حافظه مجازی که مستقیماً با فایلی بر روی دیسک در ارتباط نیستند (مانند پشته و هیپ) از این سازوکار استفاده نمایند. در Memory Mapped I/O، بخشی از فضای آدرس مجازی پردازه به دسترسی به اجزای سخت‌افزاری دستگاه‌های ورودی/خروجی نگاشت داده می‌شود. یعنی به جای استفاده از دستورالعمل‌های ورودی/خروجی پردازنده برای دسترسی به ثبات‌های یک دستگاه، از دسترسی به حافظه استفاده می‌شود. مثلا ممکن است آرایه‌ای از حافظه، متناظر با محتوای چاپ شده روی نمایش‌گر باشد.

۱۵ـ سیستم مدیریت حافظه Buddy چیست؟ یک مشکل و یک مزیت آن را بیان کنید.

پاسخ: ص ۴۳۶

یک روش تخصیص حافظه در سطح هسته سیستم عامل است (لزومی ندارد حافظه سطح کاربر، از این روش تخصیص نیابد.) تخصیص از یک Segment با اندازه ثابت و توان از دو صورت می‌گیرد. مثلا برای تخصیص ۱۱ کیلوبایت باید ۱۶ کیلوبایت

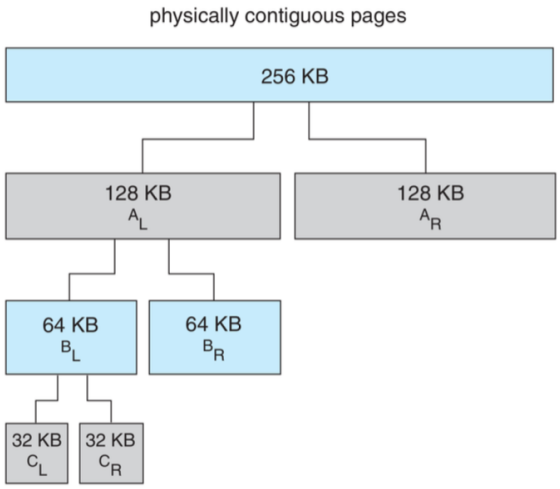
تخصیص شود. به طور کلی هر تخصیصی که توانی از دو نباشد به کوچک‌ترین توان دو که بزرگ‌تر از آن است گرد می‌شود. در صورت آزاد شدن دو قطعه حافظه هم‌اندازه مجاور که از نصف شدن یک قطعه ایجاد شده بودند (مانند B_R و B_L در شکل ۴)، می‌توان این دو قطعه را به هم چسباند (Coalescing). به این ترتیب تا حدی از External Fragmentation کاسته می‌شود. اما با توجه به این که تنها اندکی بزرگ‌تر بودن از توان دو منجر به تخصیص با اندازه دو برابر می‌شود، در بدترین حالت نزدی به ۵۰ درصد Internal Fragmentation به وجود خواهد آمد.

۱۶ـ سیستم مدیریت حافظه Slab چیست و چه رابطه‌ای با Buddy دارد؟ مزایای آن چیست؟

پاسخ: ص ۴۳۷

مشابه Buddy یک روش تخصیص حافظه در سطح هسته سیستم عامل است (در لینوکس بر روی Buddy پیاده‌سازی شده). مختص ساختار داده‌های هسته بوده و به ازای هر یک از ساختار داده‌های پراستفاده، یک Cache دارد. هر Cache شامل چندین Slab و هر Slab شامل یک یا چند صفحه پشت سر هم حافظه فیزیکی است.

هدف آن کاهش Internal Fragmentation است. به این ترتیب که چندین شئ از نوع خاص مانند Semaphore یا PCB را در یک صفحه جای می‌دهد. در هر Cache، استخرهایی از اشیاء از پیش تخصیص شده وجود داشته و به این ترتیب کارایی تخصیص حافظه، بالا است.



شکل چهار مثالی از Buddy

۱۷ـ مصالحه‌های ایجاد شده هنگام انتخاب اندازه صفحه را بیان نمایید.

پاسخ: ص ۴۴۰

اندازه صفحه کوچک‌تر: بهره‌وری بهتر حافظه (کاهش Internal Fragmentation)، کاهش میزان ورودی/خروجی (به دنبال افزایش Resolution)

اندازه صفحه بزرگ‌تر: اندازه جدول صفحه کوچک‌تر، کاهش تأثیرگذاری کندی دسترسی به دیسک ناشی از نقص صفحه (کاهش تعداد نقص صفحه‌ها) به طور کل ، تمایل به سمت افزایش اندازه صفحه‌ها است.

۱۸ـ با ذکر یک مثال نشان دهید که آشنایی کاربر با روش demand paging سیستم می‌تواند کارایی آن را بهبود دهد.

جواب : ص ۴۴۳

فرض کنید روی سیستمی با صفحات ۱۲۸ کلمه‌ای، کد زیر اجرا می‌شود که مقادیر موجود در آرایه data را صفر می‌کند.

```
int i,j;
int [128][128] data;

for ( j=0 ; j < 128 ; j++)
   for ( i=0 ; i < 128 ; i++)
      data [i][j] = 0;
```

آرایه data به این صورت پشت سر هم در حافظه ذخیره شده است:

data[0][0], data[0][1], data[0][2], …. , data[0][127] in page 0
data[1][0], data[1][1], data[1][2], …. , data[1][127] in page 1
….
data[127][0], data[127][1], data[127][2], …. , data[127][127] in page 127 in page 127

با اجرای این کد ابتدا مقدار data[0][0] از صفحه صفر مقداردهی می‌شود، سپس data[0][1] از صفحه اول، data[2][0] از صفحه دوم و ..... که اگر سیستم عامل کم‌تر از ۱۲۸ فریم در اختیار این پردازه قرارداده باشد باعث ۱۲۸×۱۲۸ نقص صفحه می‌شو . اما اگر کد بالا به صورت زیر تغییر داده شود، ابتدا تمام کلمه‌های یک صفحه مقداردهی می‌شوند سپس سراغ صفحه بعدی می‌رویم و در کل ۱۲۸ نقص صفحه خواهیم داشت.

```
int i,j;
int [128][128] data;

for ( i=0 ; i < 128 ; i++)
   for ( j=0 ; j < 128 ; j++)
      data [i][j] = 0;
```