گزارش پروژه

مهرناز حسینی ۲۲۰۷۰۱۰۵۱

هلیا رنج‌بر بافقی ۲۲۰۷۰۱۰۶۲

- ## مقایسه الگوریتم‌های تخصیص حافظه (بخش‌بندی و صفحه‌بندی)

در این پروژه، به بررسی عملکرد دو الگوریتم مختلف تخصیص حافظه به فرآیندها پرداخته شده است. این دو الگوریتم عبارتند از: الگوریتم بخش‌بندی (Segmentation) و الگوریتم صفحه‌بندی .(Paging) هدف از این پروژه شبیه‌سازی تخصیص حافظه به فرآیندها و مقایسه کارایی این دو الگوریتم در تخصیص حافظه به فرآیندهای مختلف است.

توضیحات پروژه

در ابتدا، فرآیندهایی با شناسه، مدت زمان حضور در حافظه و مقدار حافظه مورد نیاز ایجاد می‌شوند. سپس در مرحله تخصیص حافظه، سیستم به بررسی وضعیت حافظه و درخواست‌های فرآیندها پرداخته و حافظه مورد نیاز را به آنها تخصیص می‌دهد. این پروژه از دو الگوریتم بخش‌بندی و صفحه‌بندی برای تخصیص حافظه استفاده کرده است.

.1الگوریتم بخش‌بندی:

در این الگوریتم، حافظه به بخش‌های پیوسته تقسیم می‌شود. هر فرآیند به یک بخش پیوسته از حافظه نیاز دارد. الگوریتم ابتدا به دنبال پیدا کردن فضای خالی پیوسته‌ای می‌گردد که برای فرآیند کافی باشد. اگر فضای کافی پیدا شود، حافظه به فرآیند تخصیص داده می‌شود و در غیر این صورت فرآیند به صف انتظار منتقل می‌شود.

.2الگوریتم صفحه‌بندی:

در الگوریتم صفحه‌بندی، حافظه به صفحات تقسیم می‌شود و هر فرآیند به تعداد صفحات مورد نیاز خود حافظه می‌گیرد. در این الگوریتم، برای هر فرآیند تعداد صفحات مورد نیاز آن محاسبه شده و در صورت موجود بودن صفحات آزاد، حافظه به فرآیند تخصیص می‌یابد.

متغیرهای مورد استفاده

در شبیه‌سازی تخصیص حافظه، متغیرهای مختلفی برای شبیه‌سازی تخصیص حافظه و پردازش‌های آن استفاده شده‌اند که شامل موارد زیر می‌باشند:

1. **total_memory**: حافظه کل سیستم که برای تخصیص به فرآیندها در نظر گرفته شده است.
2. **allocated_memory**: مقدار حافظه‌ای که به هر فرآیند تخصیص داده شده است.

3. **request_memory**: حافظه درخواستی از هر فرآیند که به‌طور تصادفی از بازه مشخصی انتخاب می‌شود.
4. **remaining_memory**: حافظه باقی‌مانده بعد از تخصیص به فرآیندها.
5. **process_duration**: مدت زمان حضور فرآیند در سیستم که به‌طور تصادفی تعیین می‌شود.

پیاده‌سازی

1. مولفه ایجاد فرآیندها:

برای شبیه‌سازی درخواست‌های حافظه، 10,000 فرآیند به‌طور تصادفی با توجه به میزان حافظه مورد نیاز و زمان حضور در سیستم ایجاد می‌شوند.

2. مولفه تخصیص حافظه:

پس از تولید فرآیندها، الگوریتم‌ها تخصیص حافظه را انجام می‌دهند:

- **بخش‌بندی** :حافظه به‌طور پیوسته به فرآیندها تخصیص داده می‌شود.
- **صفحات** :حافظه به صفحات کوچک تقسیم و فرآیندها با توجه به نیاز صفحات خود تخصیص می‌یابند.

3. خروجی‌ها و گزارش‌ها:

برای هر فرآیند و تخصیص آن، اطلاعاتی نظیر:

- فرآیند تخصیص‌یافته: فرآیندهایی که حافظه دریافت کرده‌اند.
- فرآیندهای در انتظار: فرآیندهایی که در صف انتظار باقی مانده‌اند.
- فرآیندهای پایان‌یافته: فرآیندهایی که حافظه خود را آزاد کرده‌اند.

نتایج و تحلیل

در طول شبیه‌سازی، وضعیت حافظه و فرآیندها در هر لحظه زمان ثبت شده و به صورت فایل JSON ذخیره شده است. این داده‌ها شامل اطلاعاتی مانند فرآیندهای تخصیص یافته، فرآیندهای در صف انتظار و فرآیندهای خاتمه یافته می‌باشد. نتایج به تفکیک الگوریتم‌های بخش‌بندی و صفحه‌بندی جمع‌آوری شده و تحلیل شده است.

الگوریتم بخش‌بندی:

این الگوریتم به دلیل نیاز به فضای پیوسته برای تخصیص حافظه، ممکن است با مشکلاتی مانند تجزیه حافظه و کمبود فضای مناسب برای فرآیندها مواجه شود.

الگوریتم صفحه‌بندی:

این الگوریتم از آنجا که حافظه را به صفحات کوچکتر تقسیم می‌کند، قادر است به راحتی حافظه را بین فرآیندهای مختلف تخصیص دهد و مشکلات تجزیه حافظه را به حداقل می‌رساند.

نتیجه‌گیری

این پروژه نشان داد که الگوریتم صفحه‌بندی نسبت به الگوریتم بخش‌بندی مزایای بیشتری در تخصیص حافظه دارد. با توجه به اینکه صفحات می‌توانند در هر نقطه از حافظه قرار گیرند، این الگوریتم انعطاف‌پذیری بیشتری ارائه می‌دهد و از مشکلات تجزیه حافظه جلوگیری می‌کند. در مقابل، الگوریتم بخش‌بندی برای تخصیص حافظه به فرآیندها نیازمند فضای پیوسته است که در برخی شرایط می‌تواند منجر به ناکارآمدی شود.

- **مقایسه ی الگوریتم های مدیریت دیسک سخت (Disk Management)**

1. الگوریتم SSTF

الگوریتم Shortest Seek Time First (SSTF) یکی از روش‌های زمان‌بندی دیسک است. SSTF درخواست‌هایی که نزدیک‌ترین فاصله به موقعیت فعلی هد دیسک دارند را در اولویت قرار می‌دهد. به عبارت دیگر، هد دیسک ابتدا به سراغ درخواستی می‌رود که کمترین زمان جستجو (seek time) را نیاز دارد.

پیاده سازی:

1. دریافت ورودی‌ها:

- requests: لیست درخواست‌ها برای خواندن/نوشتن داده‌ها.
- initial_position: موقعیت اولیه هد دیسک.

2. متغیرهای مورد نیاز:

- seek_sequence: برای ذخیره ترتیب پردازش درخواست‌ها.

- current_position: موقعیت فعلی هد دیسک.

- seek_count: شمارش کل حرکت‌های هد.

- pending_requests: کپی از لیست درخواست‌ها که هنوز پردازش نشده‌اند.

- response_times: برای ذخیره زمان پاسخ به هر درخواست.

3. حلقه تا زمان پردازش تمامی درخواست‌ها:

- محاسبه فاصله‌ها:

- فاصله هر درخواست از موقعیت فعلی هد دیسک محاسبه می‌شود و در distances ذخیره می‌شود.

- انتخاب نزدیک‌ترین درخواست:

- نزدیک‌ترین درخواست با استفاده از مقدار فاصله‌ها انتخاب می‌شود (next_request).

- به‌روزرسانی ترتیب پردازش و شمارش حرکت‌ها:

- next_request به seek_sequence اضافه می‌شود.

- شمارش حرکت‌ها (seek_count) به‌روز می‌شود و current_position برابر با next_request قرار می‌گیرد.

- حذف درخواست پردازش شده:

- next_request از pending_requests حذف می‌شود.

- محاسبه و ذخیره زمان پاسخ:

- زمان پاسخ برای هر درخواست محاسبه و به response_times اضافه می‌شود.

4. محاسبه زمان پاسخ متوسط:

- زمان پاسخ متوسط با محاسبه میانگین response_times به دست می‌آید.

5. خروجی:

- تابع seek_sequence (ترتیب پردازش درخواست‌ها)، seek_count (شمارش کل حرکت‌ها) و average_response_time (زمان پاسخ متوسط) را برمی‌گرداند.

2. الگوریتم SCAN

در الگوریتم SCAN، بازوی دیسک از یک انتها شروع به حرکت می‌کند و به سمت انتهای دیگر حرکت می‌کند، و در مسیر، به محض رسیدن به هر سیلندر، درخواست‌ها را پردازش می‌کند تا به انتهای دیگر دیسک برسد. در آن انتها، جهت حرکت هد تغییر کرده و پردازش درخواست‌ها ادامه

می‌یابد. هد به طور پیوسته دیسک را از ابتدا تا انتها جستجو می‌کند. الگوریتم SCAN گاهی به عنوان الگوریتم آسانسوری نامیده می‌شود، چرا که بازوی دیسک همانند یک آسانسور در ساختمان عمل می‌کند؛ ابتدا تمامی درخواست‌های صعود را پردازش کرده و سپس برای پردازش درخواست‌های نزول، به مسیر بازمی‌گردد.

پیاده‌سازی

1. دریافت ورودی‌ها:

- requests: لیست درخواست‌ها برای خواندن/نوشتن داده‌ها.

- head: موقعیت اولیه هد دیسک.

- direction: جهت حرکت اولیه هد دیسک (چپ یا راست).

- disk_size: اندازه دیسک.

2. مرتب‌سازی درخواست‌ها:

- درخواست‌ها بر اساس موقعیت سیلندر مرتب می‌شوند ((request.sort()

3. **متغیرهای مورد نیاز:**

- seek_sequence: برای ذخیره ترتیب پردازش درخواست‌ها.

- total_seek_count: شمارش کل پیمایش‌های هد.

- response_times: برای ذخیره زمان پاسخ به هر درخواست. در انتها با میانگین گرفتن از این آرایه به متوسط زمان پاسخ‌دهی می‌رسیم.

4. جداسازی درخواست‌ها:

- درخواست‌ها به دو دسته تقسیم می‌شوند: left (کمتر از موقعیت اولیه هد) و right (بزرگتر یا مساوی با موقعیت هد).

5. **پردازش درخواست‌ها بر اساس جهت:**

- اگر جهت حرکت "چپ" باشد:

- درخواست‌های `left` به ترتیب معکوس پردازش می‌شوند.

- زمانی که هد به ابتدای دیسک رسید (اگر درخواست‌های right وجود دارد)، هد به ابتدا (موقعیت صفر) منتقل می‌شود.

- سپس درخواست‌های right پردازش می‌شوند.

- اگر جهت حرکت "راست" باشد:

- درخواست‌های right پردازش می‌شوند.

- زمانی که هد به انتهای آخرین درخواست راست رسید (اگر درخواست‌های left وجود دارد)، هد به انتهای دیسک منتقل می‌شود.

- سپس درخواست‌های left به ترتیب معکوس پردازش می‌شوند.

6. محاسبه زمان پاسخ و کل حرکت‌ها:

- زمان پاسخ به هر درخواست محاسبه و به response_times اضافه می‌شود.

- شمارش کل حرکت‌های هد به‌روزرسانی می‌شود (total_seek_count).

7. محاسبه متوسط زمان پاسخ:

- زمان متوسط پاسخ به هر درخواست با محاسبه میانگین response_times به دست می‌آید.

8. خروجی:

- تابع seek_sequence (ترتیب پردازش درخواست‌ها)، total_seek_count (شمارش کل حرکت‌ها) و avg_response_time (متوسط زمان پاسخ‌دهی) را برمی‌گرداند.

3. الگوریتم C-SCAN:

زمان‌بندی Circular SCAN (C-SCAN) نسخه‌ای از SCAN است که برای ارائه زمان انتظار یکنواخت‌تر طراحی شده است. مشابه SCAN، در این الگوریتم هد دیسک از یک انتها به انتهای دیگر حرکت می‌کند و در طول مسیر درخواست‌ها را پردازش می‌کند. اما زمانی که هد به انتهای دیسک می‌رسد، بدون پردازش هیچ درخواست دیگری، به سرعت به ابتدای دیسک باز می‌گردد.

پیاده‌سازی

1. دریافت ورودی‌ها، مرتب‌سازی درخواست‌ها((request.sort)، متغیرهای مورد نیاز و جداسازی درخواست‌ها به دو دسته right و left مانند الگوریتم SCAN است.

2. پردازش درخواست‌ها بر اساس جهت:

- اگر جهت حرکت "راست" باشد:

- ابتدا درخواست‌های right پردازش می‌شوند و هد به انتهای دیسک منتقل می‌شود.

- وقتی هد به انتهای دیسک رسید، به ابتدای دیسک منتقل می‌شود.

- سپس درخواست‌های left پردازش می‌شوند.

- اگر جهت حرکت "چپ" باشد:

- ابتدا درخواست‌های left به ترتیب معکوس پردازش می‌شوند (لیست به صورت صعودی مرتب شده بود، پس از آخر به اول باید پیمایش شود) و هد به انتهای دیسک منتقل می‌شود.

- وقتی هد به ابتدای دیسک رسید، به انتهای دیسک منتقل می‌شود.

- سپس درخواست‌های right به ترتیب معکوس پردازش می‌شوند.

3. محاسبه متوسط زمان پاسخ به هر درخواست، کل حرکت‌های پیمایش شده توسط هد و خروجی تابع مشابه الگوریتم SCAN است.

4. الگوریتم LOOK

الگوریتم LOOK یکی از روش‌های زمان بندی دسترسی به داده‌های روی دیسک است که مشابه الگوریتم SCAN عمل می‌کند. در این الگوریتم، هد دیسک به سمت اولین درخواست حرکت می‌کند و درخواست‌ها را به ترتیب بر اساس موقعیت سیلندر پردازش می‌کند، اما برخلاف الگوریتم SCAN، هد دیسک تنها تا آخرین درخواست در یک جهت حرکت می‌کند و سپس جهت خود را تغییر می‌دهد. به عبارت دیگر، هد دیسک فقط تا جایی که درخواست‌های معلق وجود دارد، حرکت می‌کند.

پیاده سازی:

1. دریافت ورودی‌ها (فقط در این الگوریتم به اندازه دیسک نیازی نداریم.)، مرتب‌سازی درخواست‌ها(()request.sort)، متغیرهای مورد نیاز و جداسازی درخواست‌ها به دو دسته right و left مانند الگوریتم SCAN است.

2. پردازش درخواست‌ها بر اساس جهت:

- اگر جهت حرکت "چپ" باشد:

- ابتدا درخواست‌های left به ترتیب معکوس پردازش می‌شوند.

- پس از اتمام درخواست‌های left، هد به سمت درخواست‌های right حرکت کرده و آنها را پردازش می‌کند. دیگر

نیازی به انتقال هد به ابتدای دیسک نیست.

- اگر جهت حرکت "راست" باشد:

- ابتدا درخواست‌های right پردازش می‌شوند.

- پس از اتمام درخواست‌های right، هد به سمت درخواست‌های left حرکت کرده و آنها را به ترتیب معکوس پردازش می‌کند.(چون لیست به ترتیب صعودی مرتب شده است.)

3. محاسبه متوسط زمان پاسخ به هر درخواست، کل حرکت‌های پیمایش شده توسط هد و خروجی تابع مشابه الگوریتم SCAN است.

5. الگوریتم C-LOOK

الگوریتم C-LOOK یا Circular LOOK یک نسخه بهینه‌شده از الگوریتم LOOK است که برای بهبود کارایی و کاهش زمان جستجو طراحی شده است. در این الگوریتم ،هد دیسک مانند الگوریتم LOOK به سمت اولین درخواست حرکت می‌کند و درخواست‌ها را بر اساس موقعیت

سیلندر پردازش می‌کند. اما برخلاف LOOK، در الگوریتم C-LOOK، هد دیسک پس از رسیدن به آخرین درخواست در یک جهت، مستقیماً به اولین درخواست در جهت مخالف بازمی‌گردد بدون اینکه درخواست‌های دیگر در مسیر بازگشت پردازش کند.

پیاده‌سازی

1. دریافت ورودی‌ها، مرتب‌سازی درخواست‌ها((request.sort()، متغیرهای مورد نیاز و جداسازی درخواست‌ها به دو دسته right و left مانند الگوریتم LOOK است.

5. پردازش درخواست‌ها بر اساس جهت:

- اگر جهت حرکت "راست" باشد:

- ابتدا درخواست‌های right پردازش می‌شوند.

- سپس هد دیسک مستقیماً به اولین درخواست left (حرکت دایره‌ای) می‌رود.

- سپس درخواست‌های left پردازش می‌شوند.

- اگر جهت حرکت "چپ" باشد:

- ابتدا درخواست‌های left به ترتیب معکوس پردازش می‌شوند.

- سپس هد دیسک مستقیماً به آخرین درخواست right (حرکت دایره‌ای) می‌رود.

- سپس درخواست‌های right به ترتیب معکوس پردازش می‌شوند.

3. محاسبه متوسط زمان پاسخ به هر درخواست، کل حرکت‌های پیمایش شده توسط هد و خروجی تابع مشابه الگوریتم SCAN است.

مقایسه الگوریتم ها:

الگوریتم	مزایا	معایب
SSTF	کاهش زمان جابجایی هد.	احتمال بروز Starvation برای درخواست‌های دورتر.
SCAN	پیاده سازی آسان و تعادل بهتر بین درخواست‌ها.	سر دیسک حتی اگر در آن جهت درخواستی نباشد همچنان به انتهای آن حرکت می‌کند.
C-SCAN	زمان انتظار به طور یکنواخت بین درخواست‌ها توزیع می‌شود.	هد در بازگشت به ابتدا یا انتهای دیسک کاری انجام نمی‌دهد.
LOOK	کاهش زمان جابجایی هد.	بازو باید به دقت درخواست‌های آخر را پیدا کند.

C-LOOK	عدم رخ دادن Starvation. زمانی که بازوی دیسک برای یافتن محل مورد نظر استفاده می‌شود، کمتر است.	بازو باید به دقت درخواست‌های آخر را پیدا کند.

مثال:

- هد دیسک در ابتدا روی **سر سیلندر 100** قرار دارد.
- لیست درخواست‌ها : **[50, 60, 110, 170, 190]**

1. SSTF:

ترتیب پردازش:
هد در هر مرحله نزدیک‌ترین درخواست را انتخاب می‌کند:
100→110→60→50→170→190

2. SCAN:

فرض :جهت هد در ابتدا به سمت راست است.
ترتیب پردازش:
100→110→170→190→60→50

فرض :جهت هد در ابتدا به سمت چپ است.
ترتیب پردازش:
100→60→50→110→170→190

3. C-SCAN:

فرض :هد در ابتدا به سمت راست می‌کند و پس از رسیدن به انتهای دیسک، به ابتدای دیسک بازمی‌گردد و دوباره حرکت را شروع می‌کند.
ترتیب پردازش:
100→110→170→190→60→50

فرض :هد در ابتدا به سمت چپ می‌کند و پس از رسیدن به ابتدای دیسک، به انتهای دیسک بازمی‌گردد و دوباره حرکت را شروع می‌کند.
ترتیب پردازش:
100→60→50→110→170→190

4. LOOK:

فرض : جهت هد در ابتدا به سمت راست است.

ترتیب پردازش:

100→110→170→190→60→50

فرض : جهت هد در ابتدا به سمت چپ است.

ترتیب پردازش:

100→60→50→110→170→190

5. C-LOOK:

فرض : جهت هد در ابتدا به سمت راست است.

ترتیب پردازش:

100→110→170→190→50→60

فرض : جهت هد در ابتدا به سمت چپ است.

ترتیب پردازش:

100→60→50→190→170→110

الگوریتم	مجموع جابه‌جایی هد	متوسط زمان پاسخ هر درخواست
SSTF	210	108
(Direction: Right) SCAN	248	131.2
(Direction: Left) SCAN	290	154
(Direction: Right) C-SCAN	358	175.2
(Direction: Left) C-SCAN	388	222.8
(Direction: Right) LOOK	230	124
(Direction: Left) LOOK	190	112
(Direction: Right) C-LOOK	240	128
(Direction: Left) C-LOOK	270	152

نتیجه‌گیری

- SSTF: این الگوریتم از نظر کاهش جابجایی هد مناسب دارد، اما احتمال گرسنگی (starvation) برای درخواست‌های دورتر وجود دارد.

- LOOK: این الگوریتم عملکردی متعادل‌تر و مشابه SCAN دارد اما کارایی بهتری نسبت به SCAN نشان می‌دهد.

- C-SCAN: برای درخواست‌های توزیع‌شده بهینه‌تر است، اما هزینه بیشتری دارد.

- C-LOOK: مشابه C-SCAN عمل می‌کند اما با کاهش حرکت‌های غیرضروری، کارایی بهتری نسبت به C-SCAN و SCAN دارد و برای بهینه‌سازی دسترسی به داده‌ها مناسب است.