اصول سیستمهای عامل

دکتر نوری خواه

پارسا آهنی – ۹۷۱۳۰۳۷

نحوه کامپایل و اجرا برنامه:

برای کامپایل کردن برنامه های سرور و کلاینت حافظه اشتراکی، میتوان از دستور های زیر استفاده کرد:

→ gcc -o server-shm ./server_shm.c -lpthread -lrt

→ gcc -o client-shm ./client_shm.c -lpthread -lrt

برای اجرا برنامه سرور کافیست server-shm اجرا شود. برای اجرای برنامه کلاینت، الزامی است پیام مورد نظر به عنوان آرگومان به برنامه داده شود:

./client-shm text

عملکرد برنامه

پس از اجرای برنامه سرور، پیامی مبنی بر اجرای موفقیت آمیز سرور در خروجی استاندارد چاپ میشود که به معنی اختصاص یافتن فضای حافظه اشتراکی و سمافور های مورد نیاز برنامه است. پس از آن می توانید برنامه کلاینت را اجرا کنید. هر بار که یک فرایند کلاینت به طور موفقیت آمیز به سرور وصل شود و درخواست خود را ثبت کند، تعداد کلاینت هایی که تا کنون به سرور وصل شده اند چاپ میشود.

برنامه کلاینت پس از دریافت جواب از سمت سرور، آن را چاپ میکند و سپس زمان سپری شده از هنگام ثبت درخواست ارسال پیام تا دریافت جواب را چاپ میکند. این زمان از درخواست دریافت سمافور client_lock_a شروع شده و با خواندن پیام سرور از روی حافظه اشتراکی b تمام میشود.

در صورتی که حافظه اشتراکی فضای کافی برای پیام کلاینت را نداشته باشد، یک پیام خطا توسط فرایند کلاینت چاپ می شود و کلاینت terminate میشود. همچنین در صورتی که تعداد کلاینت های متصل به سرور بیش از تعداد معینی باشد، با اجرای فرایند کلاینت جدید، پیامی مبنی بر پر بودن بافر چاپ میشود و آن فرایند خاتمه مییابد.

برنامه سرور سیگنال interrupt را catch میکند و با دریافت آن، حافظه اشتراکی و سمافور ها را آزاد و پاکسازی میکند. برای بستن سرور میتوان از این سیگنال یا ترکیب c + ctrl استفاده کرد.

ارزیابی عملکرد

برای مقایسه عملکرد حافظه اشتراکی و سوکت، پیام هایی یکسان با اندازه های 100, 1024, 2048, 4096, 8192 بایت برای هر کدام از سرور ها میفرستیم و زمان پاسخ دهی را مقایسه میکنیم. در ضمن پس از هر درخواست سرور متوقف نمیشود و فرایند سرور در تمام درخواست ها ثابت است.

هر پیام را ۵ بار امتحان میکنیم تا میانگین زمان پاسخ دهی بدست آید.

واحد تمامی زمانهای ثبت شده میکرو ثانیه است.

۱۰۰ بایت

Trial	Socket	Shared-Memory
1	266	179
2	304	123
3	265	151
4	305	351
5	296	225
Average	287.2	205.8

- ۱ کیلو بایت

Trial	Socket	Shared-Memory
1	297	221
2	332	166
3	313	164
4	302	123
5	343	125
Average	317.4	159.8

- ۲ کیلو بایت

Trial	Socket	Shared-Memory
1	268	240
2	403	214
3	340	135
4	218	129

5	310	201
Average	307.8	183.8

- ۴ کیلو بایت

Trial	Socket	Shared-Memory
1	372	223
2	232	191
3	277	134
4	278	272
5	262	189
Average	284.2	201.8

- ۸ کیلو بایت

Trial	Socket	Shared-Memory
1	299	164
2	300	209
3	273	281
4	278	232
5	266	196
Average	283.2	216.4

در تمامی تست ها حافظه اشتراکی میانگین زمانی بهتری را ثبت میکند که میتواند به دلیل دسترسی مستقیم دو فرایند کلاینت و سرور به حافظه باشد. همچنین اجرای پروتکل هایی مانند TCP دارای سربار زمانی است که حافظه اشتراکی نیازی به آن ندارد. در عوض پیاده سازی حافظه اشتراکی پیچیده تر و نیازمند استفاده از ابزار های همگام سازی است.

جزئیات پیاده سازی

دایرکتوری پروژه دارای ۴ فایل server_shm.c, client_shm.c mydefs.h, و shm_info.h که دو فایل هدر برای کامپایل و اجرای هر دو برنامه سرور و کلاینت مورد نیاز هستند. در فایل shm_info.h ساختار هدر حافظه اشتراکی و برخی توابع مورد نیاز کلاینت و سرور در ارتباط با حافظه اشتراکی قرار گرفته است.

- **فایل shm_info.h**

struct_shm_info کل هدر حافظه اشتراکی را شامل میشود. فیلد next_msg_offset آفست آخرین پیامی که روی حافظه نوشته شده است را نشان میدهد. فیلد messsage_info_arr آرایه از ساختار struct_msg_info است. هر خانه این ارایه که اندازه آن MSG BUFFER_SIZE است، شامل اطلاعاتی مانند: آزاد بودن یا نبودن این خانه از آرایه است(اگر آزاد باشد کلاینت میتواند اطلاعات متن را در این خانه قرار دهد)

۲- آفست این پیام بر روی حافظه اشتراکی ۳- طول پیام است.

تابع write_init_info برای نوشتن هدر ها بر روی حافظه اشتراکی استفاده میشود. به طوریکه پوینتر به ابتدای حافظه را دریافت کرده و shm_info که در بالا توضیح داده شد را روی آن می نویسد. این تابع فقط هنگام آغاز به کار سرور اجرا میشود و مقادیر اولیه هدر را ست میکند.

تابع get_shm_info پوینتر به یک shm_info را دریافت میکند و آدرس فیلد های آن را دقیقا برابر آدرس فیلد های متناظر در هدر حافظه اشتراکی قرار میدهد، بطوریکه با استفاده از آن بتوان مستقیما اطلاعات هدر حافظه اشتراکی را تغییر داد.

تابع read_msg_by_length ، با دریافت پوینتر آغازی و اندازه پیام، یک رشته را از حافظه خوانده و بر میگرداند.

- **فایل mydefs.h**

این فایل شامل تمام کتابخانه ها و defenition های مشترک بین سرور و کلاینت است که هر دو برای اجرا به آن ها نیاز دارند. سایز حافظه های اشتراکی، نام سمافور ها و نام حافظه های اشتراکی در اینجا قرار دارد.

- **server و client**

در برنامه سرور، پیش از ایجاد سمافور ها و حافظه ها، تابع clean_up صدا زده میشود. این تابع تمام سمافور ها و حافظه های اشتراکی را بسته و پاک میکند. هدف از فراخوانی آن در ابتدای برنامه آن است که در صورتی که پیش از شروع این فرایند، سرور دیگری این منابع را ایجاد کرده باشد و در اثر خطا یا سیگنال kill بسته شده باشد، این منابع حذف و سمافور ها ریست شوند.تابع initialize_shared_memory تمام سمافور های مورد نیاز کلاینت و سرور (named) و همچنین هر دو memory shared را به سرور تخصیص میدهد. همچنین هدر های هر دو حافظه را ایجاد کرده و پوینتر به آن ها را دریافت میکند. دو آرایه از سمافور ها نیز مورد نیاز است که هر کدام در یک حلقه ایجاد میشوند. تعداد اعضای این دو آرایه برابر با گنجایش آرایه message_info در هدر حافظه است. وظیفه سمافور های آرایه sem_msg_buffer_arr
ارسال سیگنال از سمت کلاینت به سرور است مبنی بر اینکه پیام خود را روی حافظه a نوشته است.
سمافور های sem_signal_response_arr نیز از سرور به کلاینت اطلاع میدهند که جواب سرور آماده است.
در نهایت سرور با توقف بر روی سمافور sem_signal_new_client منتظر یک کلاینت جدید می ماند.

در برنامه کلاینت برای اینکه بتواند هدر حافظه را تغییر دهد و به سرور درخواستش را اطلاع دهد، باید برای سمافور client_lock_a صبر کند (این سمافور حداکثر به یک فرایند اجازه ورود میدهد.) پس از آن بر روی هدر حافظه a باید یک موقعیت free پیدا کند تا اطلاعات پیام خود را روی آن بنویسد. در صورتی که چنین موقعیتی پیدا نشود، فرایند خاتمه میابد. اگر این موقعیت را پیدا کرد، اندیس آن را در متغیر last_index هدر قرار داده و با سمافور new_clinet به سرور اطلاع میدهد. سرور این اندیس را از روی هدر برداشته و سمافور client_lock_a را افزایش میدهد تا دیگر کلاینت ها درخواست خود را ثبت کنند. سرور با ایجاد یک thread و دادن این اندیس به آن، منتظر کلاینت های جدید می شود و thread ایجاد شده وظیفه جواب دادن به این کلاینت را دارد.. Thread جدید با صبر کردن برای سمافوری از آرائیه sem_sig_req_ready_arr که اندیس آن برابر اندیس دریافت شده از کلاینت میباشد، منتظر کلاینت

می ماند تا متن را روی حافظه a بنویسد و اطلاعات آن را روی هدر قرار دهد. کلاینت باید اطلاعاتی شامل offset و طول پیام را در همان خانه ای از هدر قرار دهد که اندیسش را برای سرور فرستاده است. پس از انجام این کار و اطالع دادن به server، کلاینت برای سمافورِ sem_signal_response_arr[index] صبر میکند تا جوابش آماده شود. در سرور هم نخ ایجاد شده برای این کالینت پیام را از حافظه a خوانده و جواب را رو b مینویسد. از آنجا که در این برنامه طول پیام کالینت و جواب سرور یکسان است، برای آفست پیام روی حافظه b از همان آفست حافظه a استفاده می شود و عمل دو حافظه همگام با هم پر میشوند. به همین دلیل نیازی به آپدیت کردن هدر حافظه b نیست. پس از ثبت جواب سرور روی حافظه b، به کلاینت اطلاع داده میشود و نخ مربوط به آن کلاینت در سرور خاتمه میابد. کلاینت نیز پس از خواندن جواب از b، متغیر free از خانه ای از هدر را که اشغال کرده بود، true میکند تا کلاینت های دیگر بتوانند از آن خانه
استفاده بکن