سید علیرضا میرشفیعی

81010532

OS_CA3

1)

الف)

5)

پس از اینکه ترد ها تولید میشوند، سیستم عامل برای اینکه تصمیم بگیرد بین ترد های تولید شده به کدام یک اول CPU را اختصاص دهد از یک الگوریتم زمان بندی یا همان scheduling استفاده میکند، این الگوریتم کاملا وابسته به وضعیت لحظه ای سیستم عامل است، که درنتیجه باعث میشود پیشبینیه اینکه سیستم عامل کدام ترد را انتخاب میکند غیر ممکن شود.

همچنین جدای از اسکجولر، از آنجایی که بصورت رندوم قبل از چاپ متن، هر ترد بصورت تصادفی یک sleep با زمان تصادفی انجام میدهد میتواند باعث شود که ترتیب اولیه نیز بهم بخورد.

همچنین درون خروجی نیز دیده میشود که گاهی در هنگام پرینت روی کنسول، متن دو ترد درون یکدیگر چاپ میشود، این نکته نشان میدهد که Preemption اتفاق افتاده است، به این معنی که در حین انجام یک دستور در یک ترد، سیستم عامل CPU را از آن ترد گرفته و به ترد دیگری اختصاص داده است.

ب)

4)

خیر نسخه چند تردی همیشه بهتر نیست و ممکن است که حتی از نسخه تک تردی کند تر شود.

سربار های موجود در نسخه چند تردی شمال سربار ساخت و تخریب ترد ها، مدیریت سینک کردن ترد ها برای دسترسی مشترک به حافظه، سربار context switch و همچنین در صورت عدم مدیریت کش و خراب کردن آن سربار لود کردن دوباره و دوباره حافظه درون کش میباشد.

بخشی از این سربار ها مانند مدیریت کش قابل مدیریت هستن اما در موارد دیگر مانند موارد مدیریتی و ساخت و تخریب ترد ها، این سربار ها اجتناب ناپذیرند. حال درصورتی که بخش محاسباتیه ما که در اینجا سایز آرایه میباشد از حد مشخصی کمتر باشد، سربار موجود در بخش چندتردی خودش را بیشتر نشان میدهد و باعث میشود که بخش تک تردی زمان بهتری در اجرا داشته باشد.

(5

اندازه آرایه (N)	تعداد ترد (T)	زمان تک تردی (ms)	زمان چندتردی (ms)
10^3	4	0.014864	0.906443
10^4	4	0.157615	1.0225
10^5	4	1.42461	1.49204
10^6	4	14.5088	2.20274
10^7	**4**	**83.6636**	**10.2928**

همانطور مشاهده میشود، مرزی که اورهد بخش چندتردی خودش را نشان میدهد در اندازه آرایه 10^5 قرار دارد.ب

همانطور که در بخش قبل نیز گفته شد، بخشی از سربار موجود در نسخه چندتردی همواره وجود دارد، در نتیجه اگر حجم محاسبات کوچک باشد سربار بخش چندتردی زمان بیشتری نسبت نسخه سریال خواهد داشت.

(6

مفهوم Granularity (دانه بندی) به معنای نسبت زمان محاسبه به زمان یا سربار ارتباطات در یک فرآیند موازیست، در اینجا منظور از ارتباطات، ارتباط میان بخش های موازیست و همچنین با بخش اصلی که مدیریت بخش های موازی را برعهده دارد میباشد.

حال اگر هر ترد حجم بزرگی از کار محاسباتی را انجام دهد، نیاز به ارتباط با سایر ترد ها درآن کمتر میشود یا حداقل در بازه زمانی های بیشتری این نیاز برای آن ترد پیدا میشود.

حال اگر حجم کاری این ترد ها کوچک باشد، باعث میشود که دوره زمانی کاری هر ترد کمتر شود و نیاز به همگان سازی های مکرر با سایر ترد ها داشته باشد، که این موضوع خود سربار مدیریتی بالایی خواهد داشت.

7و8)

در 2 سیستم عامل متفاوت، نحوه پیاده سازی مدیریت ترد ها متفاوت است، که در نتیجه آن باعث میشود هر سیستم عامل سربار متفاوتی در ایجاد، مدیریت و از بین بردن یک ترد داشته باشد.

همچنین در سیستم عامل های مختلف الگوریتم اسکجولینگ CPU نیز متفاوت است، در نتیجه ممکن است کارایی context switch در OS های مختلف متفاوت باشد. همچنین مدیریت کش نیز ممکن است در یک سیستم عامل بهینه تر از دیگری باشد.

ج)

7)

چرا دستور افزایش مقدار global_counter استفاده شده یک دستور atomic نیست، به عنوان مثال اگر 2 ترد بخواهند این عملیات را که شامل 3 مرحله load, increment, store هست را برای متغیر global انجام دهند، ممکن که قبل از اینکه یکی از ترد ها مقدار جدید را store کند، ترد دیگر همان مقدار قبل را load کند که همان race condition رخ خواهد داد. چرا که هیچ مکانیزمی برای جلوگیری از این موضوع اتفاق نیفتاده است.

10)

نقطه بحرانی همان بخش افزایش مقدار متغیر global است چرا که قصد دارد بر روی یک متغیر مشترک در حافظه تغییر ایجاد دهد. حال اگر در اطراف این دستور قفل ست کنیم در هر لحظه تنها یک ترد میتواند به این ناحیه وارد شده و تغییرات بر روی متغیر global انجام دهد. به همین جهت تضمین میشود که ترد بعدی که وارد این ناحیه میشود از مقدار نهایی global استفاده خواهد کرد، و ترد دیگری نیز بصورت موازی در این ناحیه نخواهد بود.

2)

در الگوریتم RR انتخاب q کاملا بستگی به نوع ورودی دارد، به عبارتی اگر q نسبت به burst time های داده شده بطور میانگین برای تمام پراسس ها، بیش از حد بزرگ باشد (مثلا بزرگتر از ماکزیمم burst time موجود)، الگوریتم مشابه FCFS عمل خواهد کرد.

در طرف دیگر اگر مقدار q بیش از حد کوچک باشد، با اینکه کار های کوچک زمان waiting time کمتری را تجربه خواهند کرد اما نکته مهم تعداد context switch های بالا خواهد بود که خود سربار قابل توجهی دارد. با اینکه در این پروژه از آن صرف نظر شده اما در یک فضای واقعی وجود خواهد داشت.

سناریو 1)

الگوریتم (Algorithm)	**کوانتوم (Quantum)**	**میانگین زمان انتظار (Avg Waiting Time)**	**میانگین زمان پاسخ‌دهی (Avg Turnaround Time)**	**وضعیت عملکرد**
SJF (Non-Preemptive)	**-**	**3.25**	**7.25**	**بهترین عملکرد** (کمترین تأخیر)
Round Robin	1	6.25	10.25	متعادل (عدالت خوب برای کارهای ریز)
Round Robin	2	7.00	11.00	متوسط
Round Robin	4	7.25	11.25	نزدیک به FCFS
FCFS	**-**	8.25	12.25	بدترین عملکرد

سناریو 2)

الگوریتم (Algorithm)	کوانتوم (Quantum)	میانگین زمان انتظار (Avg Waiting Time)	میانگین زمان پاسخ‌دهی (Avg Turnaround Time)	وضعیت عملکرد
SJF (Non-Preemptive)	**-**	**4.00**	**8.00**	**بهترین عملکرد**
Round Robin	4	4.50	8.50	خوب (چرا که نسبیت به burst time پراسس ها بهتر است)
FCFS	**-**	4.75	8.75	متوسط
Round Robin	2	5.00	9.00	ضعیف‌تر از FCFS
Round Robin	1	5.50	9.50	بدترین (سربار سوئیچ زیاد بدون فایده خاص)