سیستم‌های عامل - دکتر ابراهیمی‌مقدم

امیرحسین منصوری - ۹۹۲۴۳۰۶۹

تمرین سری چهارم

سوال ۱

بله؛ چون ممکن است دو فیلسوفی که در چپ و راست یک فیلسوف هستند به طور متناوب چنگال بردارند، و اجازه ندهند فیلسوف وسطی دو چنگال بردارد. همچنین سازوکاری برای اولویت دادن به فیلسوف‌ها نداریم تا به طور مثال بتوان اولویت را به فیلسوفی که مدت زیادی منتظر چنگال است بدهیم.

سوال ۲

فرض شده که در ~~نانوایی~~ رستوران لزلی، حداکثر N مشتری داریم.

```
1  bool choosing[N]; // all initialized to false
2  int number[N]; // all initialized to 0
3
4  finishedGettingServed(i) {
5      return number[i] == 0;
6  }
7
8  hasHigherPriority(i, j) {
9      return number[i] < number[j] || (number[i] == number[j] && i < j);
10 }
11
12 serveCustomer(i) {
13     // Entry Section
14
15     // Indicate that a number is being chosen for customer i
16     choosing[i] = true;
17
18     // Choose a unique number for i, save in number[i]
19     number[i] = max_element(number) + 1;
20
21     // Indicate that number is chosen
22     choosing[i] = false;
23
24     // Wait for all customers with higher priority to finish
25     for (int j = 0; j < N; j++) {
26
27         // Wait for customer j to finish getting number
28         while (choosing[j]);
29
30         // Wait for customer j to finish getting served, but
31         // ONLY IF it has higher priority
```

```
32          while (!finishedGettingServed(j) && hasHigherPriority(j, i));
33      }
34
35      // END OF Entry Section
36
37      // Critical Section
38      serveCustomer(i)
39
40      // Remainder Section
41
42      // Set customer i as finished
43      number[i] = 0;
44
45
46  }
```

تابع `serveCustomer(i)` در ترد iام اجرا می‌شود. در ابتدا در تابع `choosing`، مشخص می‌کنیم که در حال انتخاب شماره برای مشتری iام هستیم (چون این موضوع در Entry section تردهای دیگر بررسی می‌شود). سپس یک شماره منحصر به فرد برای مشتری iام انتخاب می‌شود و در `number[i]` ذخیره می‌شود. این شماره برابر با بالاترین شماره‌ای که از قبل به مشتریان داده شده به علاوهٔ یک خواهد بود. حال که عمل صدور شماره تمام شده، `choosing[i]` برابر `false` می‌شود.

قبل از ورود به Critical section، باید صبر کنیم تا کار تمام تردهای دیگر که اولویت بیشتر دارند و قصد انجام شدن دارند (برایشان شماره صادر شده) به اتمام برسد. حلقه `for` خط ۲۵، این موضوع را بررسی می‌کند. نشانه اتمام کار یک ترد و خارج شدنش از Critical section، صفر شدن شماره‌اش در آرایه `number` است. همچنین لازم به ذکر است که اگر ترد جدیدی وارد شود و آرایه `number` را تغییر دهد، نیازی به چک کردن مقدار جدید نوشته شده در این آرایه نداریم؛ چون این ترد جدید قطعا اولویت کمتری نسبت به ترد iام خواهد داشت و نباید برایش صبر کنیم. پس چه این حلقه، ترد جدید را بررسی کند و چه بررسی نکند، برای ترد جدید صبر نمی‌کنیم و الگوریتم همچنان صحیح کار می‌کند.

در نهایت، وارد Critical section می‌شویم و پس از خارج شدن از آن، به نشانه اتمام کار، `number[i]` را صفر می‌کنیم.

حال شرط‌های سه‌گانه را بررسی می‌کنیم:

- **شرط Mutual exclusion:** چون اولویت هیچ مشتری‌ای با مشتری دیگر یکی نیست (با فرض یکتا بودن شماره هر مشتری)، و هر مشتری صبر می‌کند تا مشتری‌های با اولویت بیشتر به پایان برسند، قطعا هیچ دو مشتری‌ای با هم وارد Critical section نمی‌شوند. پس این شرط برقرار است.
- **شرط Progress:** اگر کسی در Critical section نباشد، پس هیچ مشتری‌ای در آرایه `number` برای خودش شماره ندارد. پس اگر مشتری جدیدی وارد شود، هیچ‌کدام از شرط‌های خط ۲۸ و ۳۲ برای بقیه مشتریان (و حتی خودش) برقرار نخواهد بود؛ و بلافاصله وارد Critical section می‌شود. پس این شرط نیز برقرار است.
- **شرط Bounded-waiting:** تعداد مشتریانی که هر مشتری جدید در Entry section برایش صبر می‌کند محدود، و حداکثر برابر $N-1$ است. ورود مشتریان جدیدتر نیز به این تعداد چیزی اضافه نمی‌کند؛ چون مشتریان جدیدتر همه اولویت کمتری خواهند داشت. پس هر مشتری جدید، مدت زمان محدودی را صبر خواهد کرد و بالاخره وارد Critical section خواهد شد. پس این شرط نیز برقرار است.

سوال ۳

```
1  monitor binSemaphore {
2      bool locked = false;
3      condition lockCondition;
4
5      signal() {
```

```
6           lock = false;
7           lockCondition.signal();
8       }
9
10      wait() {
11          while (locked) lockCondition.wait();
12          locked = true;
13      }
14  }
```

چون در `signal()`، تابع `lockCondition.signal()` آخرین خط است، تفاوتی نمی‌کند که Condition variable ما با روش Signal-and-wait کار کند یا Signal-and-continue.

سوال ۴

این راه، چندان راه خوبی نیست و فقط شرط Mutual exclusion را برآورده می‌کند:

- هر پروسه‌ای که وارد Critical section شده، `flag` خود را ست می‌کند؛ و در Entry section خود، ست شدن `flag` پروسهٔ دیگر را بررسی می‌کند و در این صورت، وارد Critical section نمی‌شود. بنابراین هیچ‌وقت دو پروسه با هم وارد Critical section نمی‌شوند. پس شرط Mutual exclusion برقرار است.
- می‌توان طوری بین این دو پروسه Context-switch انجام داد که هیچ‌کدام وارد Critical section نشوند! فرض می‌کنیم پروسهٔ A تا آخر خط ۳ اجرا می‌شود؛ سپس Context-switch رخ داده و پروسهٔ B نیز تا آخر خط ۳ اجرا می‌شود. در این حالت، `flag` هر دو پروسه ست شده است. سپس دوباره Contxt-switch رخ می‌دهد و خط ۴ پروسهٔ A نیز اجرا شده و به خط اول این پروسه بر می‌گردیم. در اینجا نیز یک Context-switch دیگر رخ می‌دهد و اتفاق مشابه برای برای پروسهٔ B رخ می‌دهد و به خط اول در این پروسه نیز برمی‌گردیم.

  در اینجا هر دو پروسه به خط اول بازگشته‌اند. اگر به همین روال Context-switch داشته باشیم، هیچ‌گاه از Entry section عبور نمی‌کنیم. پس شرط Progress برقرار نیست.
- فرض می‌کنیم پروسهٔ A تا آخر خط ۳ اجرا می‌شود و سپس Context-switch رخ می‌دهد. چون `flag(A)` ست شده، پروسهٔ B در Entry sectionاش می‌ماند و جلوتر نمی‌رود. باز فرض می‌کنیم در حالتی که `flag(B) = 0`، یک Context-switch دیگر رخ داده و به پروسهٔ A برمی‌گردیم. فرض می‌کنیم پروسهٔ A کد Critical sectionاش را کامل انجام می‌دهد، و به خط ۳ برمی‌گردد و پس از اجرای این خط، دوباره Context-switch داریم. اگر به همین روال Context-switch داشته باشیم، پروسهٔ B هیچ‌گاه موفق به ورود به Critical sectionاش نمی‌شود و فقط پروسهٔ A موفق خواهد بود. پس شرط Bounded-waiting هم برقرار نیست.

سوال ۵

روند اجرای دستورات به شکل زیر است. در این جدول، R نشانه وضعیت Running، و W نشانه وضعیت Waiting است.

Process	Semaphore Value	P_1	P_2	P_3
P_1	$S = 0$	R	R	R
P_2	$S = -1$	R	W	R
P_3	$S = 0$	R	R	R
P_2	$S = -1$	R	W	R
P_1	$S = 0$	R	R	R
P_3	$S = 1$	R	R	R
P_2	$S = 0$	R	R	R
P_2	$S = -1$	R	W	R
P_3	$S = 0$	R	R	R
P_1	$S = 1$	W	R	R

در انتها، تنها پروسه P_1 در وضعیت Waiting خواهد بود. بقیه پروسه‌ها در وضعیت Running (یا دقیق‌تر بگوییم، Ready) خواهند بود.

سوال ۶

سمافور S_2 فقط در P_2 استفاده شده و در همان خط اول نیز `wait` روی آن صدا زده شده است. پس P_2 همان اول به حالت waiting رفته و تا آخر در همین حالت می‌ماند؛ بنابراین می‌توانیم P_2 را نادیده بگیریم.

پروسه P_0 نیز همان ابتدا به حالت waiting می‌رود؛ اما پس از یک بار اجرای حلقه‌ی P_1، دوباره می‌تواند اجرا شود.

در ابتدای حلقه‌ی P_1 نیز یک `wait(S1)` و در انتهای حلقه هم یک `release(S1)` داریم. چون هیچ `wait` دیگری در حلقه نداریم و در هیچ پروسهٔ دیگری نیز روی S_1 `wait` نداریم، عملا P_1 هیچ‌گاه متوقف نمی‌شود و همیشه اجرا خواهد شد. لازم به ذکر است که اجرای خط دوم P_0 نیز تاثیری بر این موضوع نخواهد داشت.

بنابراین عبارت مشهور `"ALI ALAVI"` به تعداد نامعلومی و تا زمان خاتمه P_1 (مثلاً توسط سیستم‌عامل) چاپ خواهد شد.

سوال ۷

این حالت از Condition variable، به جای استفاده از صف FIFO برای نگهداری پروسه‌های منتظر، از یک صف اولویت استفاده می‌کند. تابع `wait` نیز یک مقدار c می‌گیرد، و پروسه فعلی را با این مقدار اولویت وارد صف می‌کند. هنگامی که روی این Condition variable، تابع `signal` را صدا بزنیم، پروسه با اولویت بیشتر از صف بیرون می‌آید و شروع به کار می‌کند. به عبارت دیگر، زمان ورود به صف دیگر ملاک نیست.