آمادگی برای آزمون میانی سیستم عامل

۱) قطعه‌کدهای زیر را در نظر بگیرید که در ریسمان‌های (Threads) متفاوتی اجرا می‌شوند. با مثالی توضیح دهید مشکل Race condition چرا پیش می‌آید و منجر به چه نتیجه‌ی غیر منتظره‌ای می‌شود. مشکل را با استفاده از قفل Mutex (یا Semaphore یا Monitor) حل نمایید. آیا می‌توان در این مسئله از متغیرهای Atomic برای حل مشکل دسترسی‌های همزمان استفاده نمود؟

۲) ریسمان‌های P1، P2 و P3 از منبع مشترکی استفاده می‌نمایند. با توجه به این مسئله که منبع در هر لحظه می‌تواند فقط توسط یک ریسمان استفاده شود، وقتی که یکی از این ریسمان‌ها از این منبع استفاده می‌کند، بقیه باید برای استفاده از آن منتظر باشند. فرض کنید یکی از ریسمان‌ها در حال استفاده از منبع باشد؛ وقتی کار این ریسمان با منبع تمام می‌شود از بین ریسمان‌های منتظر، باید ریسمانی از منبع استفاده نماید که اندیس کوچکتری دارد (برای مثال اگر P1 و P3 منتظر باشند، P1 باید به منبع دسترسی داشته باشد). راه حلی برای این مسئله با استفاده از Monitor ارائه دهید. راه حلی برای این مسئله با استفاده از Semaphore ارائه دهید. آیا در این مسئله قحطی (Starvation) رخ می‌دهد؟ در این صورت، راه حل دیگری شرح دهید (با مشخص کردن ریسمان منتظری که باید منبع به آن داده شود) که در آن قحطی رخ ندهد. راه حل خود را با Monitor پیاده‌سازی نمایید.

در این مسئله (و مسئله‌های مشابه) می‌توانید برای هر ریسمان (یا رده‌ی اولویت) یک متغیر Condition در مانیتور در نظر بگیرید (مثل آرایه‌ی C[3]) و ریسمان‌ها منتظر را در یک آرایه (مثل waiting[3]) علامت بزنید. در مانیتور، یک تابع برای تقاضای استفاده از منبع (مثل acquire()) و یک تابع برای رها کردن آن (مثل release()) تعریف کنید. در تابع acquire() در صورتی که منبع آزاد باشد، از آن استفاده کنید. در غیر این صورت، درایه‌ی مناسب آرایه‌ی waiting را تغییر دهید و تابع wait() را برای متغیر Condition مربوط به آن ریسمان در آرایه‌ی C صدا بزنید. در زمان اتمام کار یک ریسمان، تابع signal() را برای متغیر Condition-ای صدا بزنید که مربوط به ریسمان منتظر با کوچک‌ترین اندیس باشد.

۳) در شبه کد زیر از فراخوانی سیستمی ()fork برای ایجاد پردازه‌ی جدید استفاده می‌شود و فراخوانی سیستمی ()getpid شماره‌ی پردازه را بر می‌گرداند. در این شبه کد چند پردازه ایجاد می‌شوند؟ با فرض اینکه شماره‌ی پردازه‌های (Process Identifiers) ایجاد شده در طول زمان افزایش می‌یابند (بدیهی است که شماره‌ی پردازه‌ها تکراری نیستند) یک خروجی ممکن از اجرای این کد را نمایش دهید. درخت پردازه‌ها را (به هر پردازه یک رأس تخصیص دهید و فرزندهای یک پردازه، فرزندهای آن پردازه در درخت نیز هستند) با نشان دادن شماره‌ی پردازه‌ها نمایش دهید.

در این سؤال و سؤال‌های مشابه دقت نمایید که هر یک از پردازه‌های فرزند نیز خود می‌توانند ()fork را فراخوانی کنند. همچنین دقت کنید که پردازه‌ی ایجاد شده با فراخوانی سیستمی ()fork دقیقا از بعد از فراخوانی این تابع، اجرای دستورات را شروع می‌نماید.

۴) در جدول زیر زمان اضافه شدن تعدادی پردازه به صف آماده‌باش (Ready queue) سیستم عامل و زمان استفاده‌ی آنها از CPU (CPU Burst) نشان داده شده‌اند. نمودار Gantt (نمایش وضعیت اجرای پردازه‌ها در طول زمان) را برای زمانبندهای FCFS (First-Come, First-Served)، SJF (Shortest Job First) و RR (Round-Robin) با برش‌های زمانی (Timeslice) ۱۰ میلی‌ثانیه برای یک پردازنده نشان دهید (فرض کنید زمانبند SJF، Preemptive باشد). مقدار میانگین زمان Waiting و زمان Turnaround را برای هر سه الگوریتم محاسبه نمایید. با توجه به اولویت پردازه‌ها در این جدول، به صورت مشابه این عملیات را برای الگوریتم زمانبندی Priority نیز انجام دهید.

۵) در الگوریتم زمانبندی MFQ (Multilevel Feedback Queue) فرض کنید صف اول با الگوریتم RR و برش زمانی ۱۰ میلی ثانیه، صف دوم با الگوریتم RR، برش زمانی ۱۵ میلی ثانیه و صف سوم با الگوریتم FCFS زمانبندی می‌گردند. همچنین فرض کنید پردازه‌های جدید در صف اول وارد می‌شوند و تنها وقتی پردازه‌ها در صف‌های دوم و سوم اجرا می‌شوند که صف‌های قبلی خالی باشند. نمودار Gantt را برای زمانبندی پردازه‌های زیر نشان دهید. پردازه‌ای از صف اول به صف دوم (یا از صف دوم به صف سوم) انتقال می‌یابد که در برش زمانی خود کارش به پایان نرسد.