به نام خدا

سیستم‌های عامل- گروه ۱ (نیم‌سال دوم ۹۹-۴۰۰)

تمرین شماره ۱: مفاهیم پایه

آخرین تاریخ بارگذاری پاسخ در courses:

ساعت ۲۳:۵۹ روز ۲۹ اسفند ۱۳۹۹

سوال ۱: سامان یک هکر با آینده‌ای درخشان است، او یک بدافزار به نام OSAnswerFinder نوشته که وقتی اجرا می‌شود با گرفتن صورت سوالات درس سیستم‌عامل جواب آن ها را با جست‌جو در وب پیدا کرده و نمایش می‌دهد، حداقل کاربر ناآگاه که بسیار برای پیدا کردن جواب های تمرین هیجان زده است، اینطور فکر می‌کند! سامان این برنامه را طوری طراحی کرده که وقتی اجرا می‌شود با خواندن اطلاعات دیگر برنامه‌ها مانند مرورگر تلاش بر فهمیدن هویت کاربر کرده و آن را برای سامان ارسال می‌کند تا سامان بتواند او را برای تقلب جریمه کند. نکته‌ای که توجه تیم تدریسیاری را در مورد این برنامه جلب کرد این بود که این برنامه کاری خطرناک و غیرعادی انجام می‌دهد (با اینکه پیدا کردن جواب تمرینات نیز بسیار کار خطرناکی است و بنظر ما سیستم‌عامل باید جلوی آن را بگیرد، در اینجا منظور فهمیدن هویت کاربر است) و قاعدتا سیستم‌عامل نباید اجازه چنین کاری را به آن بدهد. شقایق پس از مطالعه در مورد این ساز و کارها به ما توضیح داد که سیستم‌عامل از چیزی به نام user mode و kernel mode برای اعمال چنین محدودیت‌هایی کمک می‌گیرد.

الف) چه تفاوت‌هایی بین این دو mode وجود دارد؟

ب) سه دستور نام ببرید که سیستم‌عامل باید روی اجرای آن‌ها کنترل اعمال کند. در کل چه نوع دستوراتی نیازمند کنترل شدن هستند؟

پ) این کنترل نرم‌افزاری انجام می‌شود یا سخت‌افزاری؟ توضیح دهید، آیا میتوان بیش از دو mode مختلف دسترسی تعریف کرد (مثلا برای مجازی‌سازی)؟ چطور؟

سوال ۲: آتش دعوای همیشگی کدام سیستم عامل از همه بهتره با شروع ترم بین تیم تدریسیاری اوج گرفته، ماهان که شدیدا از اپل متنفر است، پس از تحقیقات فراوان متوجه شد که MacOS از ساختار microkernel، لینوکس از ترکیبی از monolithic و modular و ویندوز از monolithic به عنوان ساختار اصلی خود استفاده می‌کنند. ماهان که از این همه تحقیق خسته شده تصمیم گرفته که مقایسه کردن این سه ساختار رو به شما بسپاره تا خودش بتونه استراحت کنه و بعدا با استفاده از جواب های شما دعوای بهترین سیستم عامل رو ببره.

الف) اول برای اینکه ماهان مطمئن بشه که جوابی که میگیره معتبره توضیح بدین که ساختار سیستم عامل و تفاوتش با معماری اون چیه؟

ب) سه ساختاری که بالا گفته شد رو از نظر معیارهای طراحی، قابلیت اطمینان، قابلیت اشکال زدایی و کارایی کلی با هم مقایسه کنید و دلیل خودتون هم برای هر مورد ذکر کنید.

سوال ۳: بعد از خواندن مبحث فراخوانی سیستمی (system call) و فهمیدن این که تقریبا هیچ برنامه‌ای بدون استفاده از آن ها نمی‌تواند عملیاتی انجام دهد (حتی hello world!). شما که تا الان کدهای زیادی به زبان های مختلفی نوشته‌اید احتمالا این سوال برایتان پیش آمده که «پس چرا من تا الان از فراخونی‌های سیستمی استفاده نکرده‌ام؟» جوابی که برای این سوال وجود دارن این است که بدون اینکه خودتان متوجه باشید استفاده کرده‌اید زبان‌های برنامه‌نویسی واسطهایی برای فراخوانی‌های سیستمی در اختیار ما قرار می‌دهند که امروزه توسعه دهندگان به طور عمده از این واسطهای برنامه‌نویسی (API) استفاده می‌کنند. برای مثال تابع printf از چندین سیستم کال برای نوشتن خروجی استفاده می‌کند. برای اینکه این موضوع رو بهتر درک کنید به سوالات زیر پاسخ بدید:

الف) مزایای استفاده از این واسطها را بیان کنید.

ب) سه زبان C, Java, Python را در نظر بگیرید، برای ترجمه API ها زبان C از شیوه Compile شدن برای هر سیستم خاص، زبان پایتون از اجرای مفسری و زبان جاوا از شیوه ماشین مجازی برای اجرا شدن روی سیستم های مختلف استفاده میکنند. این سه شیوه رو با یکدیگر مقایسه کنید و معایب و مزایای هرکدام رو نام ببرید.

پ) سیستم عامل لینوکس از چند راه برای عبور متغیرها (Parameter Passing) استفاده می کند. آن‌ها را ذکر کرده و توضیح دهید که استفاده ترکیبی از آن ها چه فایده‌هایی دارد؟

نحوه تحویل تمرین ۱

پاسخ به سوالات را در قالب یک فایل پی دی اف (اسکن یا تایپ شده) با نام «hw1_student_id» در صفحه درس اپلود کنید. پاسخ‌های شما بایستی دقیق و خوانا باشند.

جریمه دیرکرد

هر روز تاخیر در ارسال تمرین ۱۰٪ نمره منفی خواهد داشت. امکان اپلود تمرین تنها تا ۵ روز از تاریخ تعیین شده ممکن خواهد بود.

جریمه تقلب

۱. همه دانشجویان بایستی که خود تمرین را انجام دهند و هرگونه تقلب یا ارسال کار دیگران یا کارهای موجود در وب که تمرین را به شکل جزئی یا کلی انجام داده است، غیرقابل پذیرش و عواقب شدیدی خواهد داشت.

۲. بنده و گروه حل تمرین تمام تلاش خود را برای شناسایی تقلب‌های احتمالی خواهیم کرد تا در نهایت یک ارزیابی عادلانه از همه دانشجویان عزیز داشته باشیم. ما از Moss برای شناسایی فایل‌های مشابه استفاده خواهیم کرد.

۳. در صورت شناسایی تقلبی که ۵۰ درصد یا پایینتر از کار را شامل می‌شود، دانشجوی مورد نظر اخطار اول را دریافت کرده و نمره «۰.۵- * بارم تمرین ۱» به ایشان تعلق می‌گیرد و در صورت شناسایی تقلبی که بیشتر از ۵۰ درصد کار را پوشش می‌دهد به دانشجوی مورد نظر اخطار دوم تعلق گرفته و نمره «۱- * بارم تمرین ۱» به ایشان تعلق می‌گیرد. علاوه بر این نمره منفی، گرفتن دو اخطار، به معنی لحاظ شدن نمره منفی برابر با بارم همه تمرینات خواهد بود.

*در نهایت، هر گونه سوال در مورد تمرین و بخش‌های آنها را **تنها و تنها** از طریق سایت درس و ایجاد مباحثه با عناوین مرتبط مطرح بفرمایید.*

تندرست و موفق باشید

تیم درس سیستم‌های عامل

سوال اول

الف) چه تفاوت‌هایی بین این دو mode وجود دارد؟

در حالت Kernel، کد اجرایی (کدهای هسته سیستم عامل) دسترسی کامل و بدون محدودیت به سخت افزار و منابع سیستم دارد، در حالی که در حالت User کد اجرایی (کدهای کاربر) دسترسی مستقیم به حافظه اصلی یا سخت افزار را ندارد و دسترسی محدود است. به این ترتیب بد افزارها یا کاربرهای بدخیم نمی توانند کد هسته سیستم عامل را تغییر دهند و به سخت افزار دسترسی داشته باشند.

هنگام بوت و بارگذاری سیستم عامل، سیستم در حالت kernel قرار دارد و پس از آن به حالت User تغییر داده می‌شود و برنامه‌های کاربر اجرا می‌شود. در صورت فراخوانی‌های سیستمی یا ایجاد وقفه سیستم مجدد به حالت kernel می‌رود.

ب) سه دستور نام ببرید که سیستم عامل باید روی اجرای آن‌ها کنترل اعمال کند. در کل چه نوع دستوراتی نیازمند کنترل شدن هستند؟

تغییر و پاک کردن حافظه، تغییر آدرس‌های حافظه، دستورالعمل‌های I/O و دستور halt، خاموش کردن دریافت وقفه و ... از جمله دستوراتی هستند که باید تحت نظارت و کنترل سیستم عامل انجام شوند که به آن‌ها دستورات Privileged گفته می‌شود.

ج) این کنترل نرم افزاری انجام می‌شود یا سخت افزاری؟ توضیح دهید، آیا می‌توان بیش از دو mode مختلف دسترسی تعریف کرد (مثلا برای مجازی سازی)؟ چطور؟

این پیاده سازی به صورت سخت افزاری انجام می‌شود. در حالت عادی یک بیت برای تعیین حالت سیستم (Kernel mode, User mode) کافی است. حال در صورتی که CPU قابلیت پشتیبانی از چند حالت را داشته باشد با در نظر گرفتن بیت‌های بیشتر می‌توان حالت‌های بیشتری تعریف نمود. به طور مثال با دو بیت می‌توان 4 حالت ایجاد کرد و یک مقدار را برای حالت user و یک مقدار را برای حالت kernel در نظر بگیریم و دو مقدار دیگر به عنوان حالت‌های میانی در نظر گرفته شود.

سوال دوم

الف) اول برای اینکه ماهان مطمئن بشه که جوابی که میگیره معتبره توضیح بدین که ساختار سیستم عامل و تفاوتش با معماري اون چیه؟

چون سیستم عامل یک برنامه بزرگ و پیچیده است، باید به دقت مهندسی شود تا به خوبی کار کند و راحت تغییر داده شود. یک راه این است که به ماژول‌ها و کامپوننت‌های کوچکتر بشکنیم که هر ماژول یک بخش خوش تعریف از سیستم باشد. به طور کلی میتوان از ساختارهای ساده مثل ساختار MS-DOS، پیچیده تر مثل ساختار UNIX، ساختار لایه‌ای و یا microkernel استفاده کرد. به طور کلی، ساختار روش‌های ارتباط بخش‌های مختلف سیستم عامل به یکدیگر است.

معماری سیستم شامل ارتباط هسته‌ها و پردازنده‌ها و مواردی از این قبیل است که دو حالت تک پردازنده‌ای و چند پردازنده‌ای دارد. سیستم تک پردازنده ای، یک پردازنده و یک هسته دارد. سیستم‌های چند پردازنده‌ای میتواند دو یا چند پردازنده داشته باشد که هرکدام یک CPU تک هسته‌ای دارند. در این حالت، bus و گاهی اوقات clock، حافظه و peripheralها به اشتراک گذاشته میشوند. تعریف سیستم‌های چندپردازنده‌ای با گذر زمان، شامل سیستم‌های چند هسته‌ای می‌شود که در آن‌ها چند هسته روی یک chip قرار می‌گیرند که از سیستم‌ها ی با چند chip که هرکدام یک هسته دارند کارایی بیشتری دارند و ارتباط هسته‌ها نیز سریع‌تر است.

ب) سه ساختاري که بالا گفته شد رو از نظر معیارهاي طراحی، قابلیت اطمینان، قابلیت اشکال زدایی و کارایی کلی با هم مقایسه کنید و دلیل خودتون هم براي هر مورد ذکر کنید.

ساختار monolithic: چون فانکشن‌ها ی زیاد در یک لایه قرار میگیرند به آن یکپارچه گفته میشود. در این ساختار، kernel شامل همه چیز بین اینترفیس فراخوانی سیستمی و سخت افزار است. ساده ترین ساختار است که در آن تمام عملکرد kernel داخل یک فایل باینری استاتیک قرار میگیرد و در یک فضای حافظه اجرا میشود. با وجود سادگی ظاهری، پیاده سازی و گسترش این ساختار دشوار است. با این حال، ارتباطات درون kernel سریع است و overhead کمتری هنگام فراخوانی سیستمی پیش می‌آید و سرعت و کارایی بالایی دارد.

سطح حمله در این ساختار گسترده است که یکی از مشکلات آن است و در نتیجه امنیت پایین تری دارد و چون همه بخش‌ها در دست kernel است قابلیت اطمینان هم کمتر است چون ممکن است از کار افتادن یک قسمت روی بقیه هم اثرگذار باشد. چون همه بخش‌ها یکجاست، تغییر در یک قسمت تاثیر زیادی روی بقیه قسمت‌ها خواهد داشت و اشکال زدایی نسبت به microkernel دشوارتر است چون همه قسمت‌ها به هم وابسته و در یک unit هستند.

ساختار microkernel: در این حالت، کامپوننت‌های غیر ضروری از kernel حذف شده و به عنوان برنامه‌های user پیاده سازی می‌شوند و در آدرس‌های جدا قرار میگیرند. در نتیجه، kernel کوچک تر خواهیم داشت.

کاربرد اصلی این است که بین برنامه client و بقیه سرویس‌هایی که در فضای user در حال اجرا هستند، ارتباط برقرار شود. این ارتباط توسط message passing انجام می‌شود که در این حالت client و سرویس به صورت غیر مستقیم با یکدیگر ارتباط دارند و پیام‌ها را از طریق microkernel منتقل میکنند. از فواید این ساختار این است که گسترش سیستم عامل را آسان تر میکند. تمام سرویس‌های جدید به فضای user اضافه میشوند و به اصلاح و تغییر kernel نیازی نخواهد بود و در نتیجه تغییرات کمتر میشوند چون kernel هم کوچک تر است. انتقال سیستم عامل در این ساختار از یک طراحی سخت افزاری به بقیه راحت تر است. پیاده سازی و نگهداری آسان تر است. این ساختار امنیت و قابلیت اطمینان بیشتری دارد زیرا بیشتر سرویس‌ها به عنوان user اجرا می‌شوند و اگر سرویسی از کار بیفتد، روی بقیه اجزا تاثیری نخواهد داشت. امنیت به دلیل کوچک شدن سطح حمله بالاتر و به دلیل کوچک شدن بخش‌ها اشکال زدایی هم راحت‌تر است.

کارایی این ساختار به علت افزایش overhead نسبت به حالت قبل پایین می‌آید. مثلا وقتی دو سرویس user-level قرار است ارتباط برقرار کنند، پیام‌ها باید بین سرویس‌ها که هر کدام در فضای جدا قرار دارند کپی شود.

همچنین ممکن است سیستم عامل مجبور شود از یک process به دیگری سوییچ کند که پیام‌ها را مبادله کند.

Overheadای که در این مراحل ایجاد میشود بزرگترین مانع گسترش سیستم عامل‌های مبتنی بر این ساختار است.

ترکیب monolithic و modular: میتوان یک سیستم را به بخش‌های کوچک تر تقسیم کرد که هر کدام کاربرد خاصی داشته و در کنار هم kernel را تشکیل دهند. (loosely coupled) یک سیستم به روش‌های مختلفی می‌تواند ماژولار شود که ساختار لایه‌ای یکی از این روش هاست. در این حالت، سیستم عامل به لایه‌های مختلفی شکسته میشود که پایین‌ترین آن‌ها سخت افزار و بالاترین interface است. هر لایه شامل یک سری ساختمان داده و فانکشن است. فایده اصلی این روش سادگی پیاده سازی و اشکال زدایی است. مثلا لایه اول میتواند بدون تاثیری روی بقیه سیستم، اشکال زدایی شود. وقتی این لایه اشکال زدایی شود، سراغ لایه دوم می‌رود و به همین ترتیب تا آخر. در این صورت میتوان مطمئن شد که اگر یک خطا در لایه‌ای پیدا شود، این خطا مربوط به همان لایه است چون لایه‌های پایینی اشکال زدایی شده‌اند. بنابراین طراحی و پیاده سازی سیستم آسان میشود. یکی از سیستم‌های عاملی که از ترکیب این دو استفاده می‌کند، UNIX است که در آن، دو بخش مجزای kernel و system program است

که خود kernel به چند اینترفیس و device driver تقسیم می‌شود که به همین دلیل می‌توان آن را ترکیبی از monolithic و modular دانست. این حالت مزایا و معایب هر دو روش را شامل میشود.

به صورت کلی، بهترین شیوه استفاده از LKM هاست که در آن، kernel سرویس‌های هسته را انجام داده و سایر سرویس‌ها به صورت پویا پیاده‌سازی می‌شوند.

** نمره این قسمت به دانشجویانی که سوال را با توجه به ساختارهای monolithic microkernel modular به درستی پاسخ داده‌اند نیز به صورت کامل تعلق گرفته است.

سوال سوم

الف) مزاياي استفاده از اين واسطها را بيان كنيد.

این واسطها (API) مجموعه‌ای از فانکشن‌ها که برای برنامه‌نویس قابل استفاده‌اند، پارامترهایی که به هر فانکشن داده می‌شود و آن فانکشن برمیگرداند را مشخص میکند. از دلایلی که برنامه نویسان ترجیح میدهند از API استفاده کنند، میتوان به آنها اشاره کرد: این که برنامه روی هر سیستمی که همان API را ساپورت میکند اجرا شود. (portability)، بالا بردن سرعت، اینکه لازم نیست برنامه نویس از تمام جزئیات فراخوانی سیستمی مطلع باشد و می‌تواند کارها را به سیستم عامل بسپارد، یکپارچه شدن و بالا رفتن انعطاف پذیری و کارایی برنامه‌ها و افزایش اسکوپ.

ب) سه شيوه رو با يكديگر مقايسه كنيد و معايب و مزاياى هركدام را نام ببريد.

کامپایلر کد در زبان سطح بالا مثل جاوا را میگیرد و به زبان سطح پایین و قابل فهم برای ماشین تبدیل می‌کند. در این روش ابتدا کد کامپایل شده و سپس وارد مرحله اجرا می‌شود. در این روش کد یک جا کامپایل می‌شود و خط به خط نیست و در نتیجه اشکال زدایی سخت تر است. چون کامپایلر وابسته به سیستم عامل است، فقط روی سیستم عاملی که آن کامپایلر برایش طراحی شده میتواند اجرا شود. این روش چون یک جا کامپایل می‌کند نسبت به مفسر سریع تر است.

روش مفسری: این روش در ساختار شبیه روش کامپایل است، با این تفاوت که مفسر دستورالعمل‌ها را مستقیما از منبع اصلی اجرا کرده و آن‌ها را ترجمه نمی کند. مفسر برنامه را به صورت خط به خط اجرا میکند و در نتیجه نسبت به کامپایلر، اشکال زدایی راحت‌تری دارد. این روش سرعت کمتر و انعطاف پذیری بیشتری دارد. مفسر وابسته به سیستم عامل نیست که استفاده راحت تری خواهد داشت.

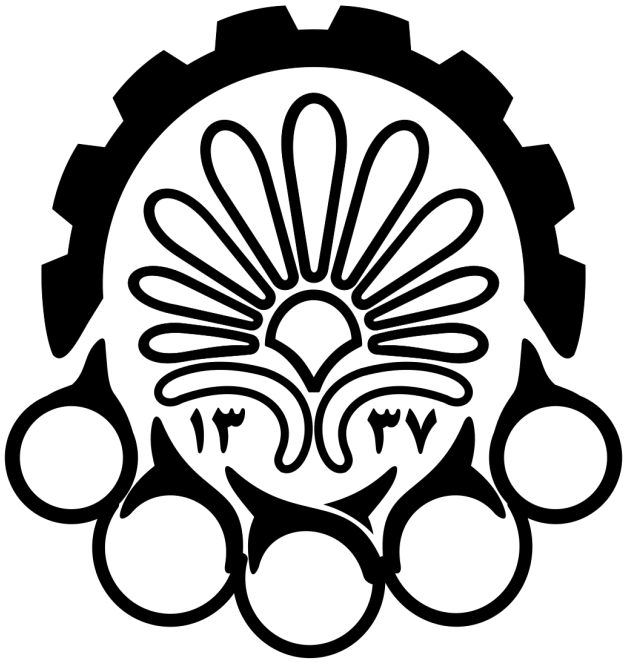
روش ماشین مجازی: در این روش، ابتدا کامپایلر کد را به یک فایل میانی تبدیل می‌کند که این فایل وابسته به سیستم عامل است و سپس مفسر این فایل را به یک زبان قابل فهم برای ماشین تبدیل می‌کند که در این حالت دیگر به سیستم عامل وابسته نیست. خود ماشین مجازی نیز به سیستم عامل وابسته نیست. کارایی و سرعت نسبت به دو حالت قبل بیشتر است.

ج) سيستم عامل لينوكس از چند راه براي عبور متغيرها (Parameter Passing) استفاده مى‌كند. آن‌ها را ذكر كرده و توضيح دهيد كه استفاده تركيبى از آن‌ها چه فايده‌هايى دارد؟

میتوان پارامترها را در رجیسترهای مختلف ذخیره کرد. چون تعداد رجیسترها معمولا محدود است، ممکن است از تعداد پارامترها کمتر شود. حالت بعدی این است که پارامترها را در استک

push و pop کنیم و حالت سوم این است که پارامترها را در بلاک‌ها یا خانه‌های حافظه ذخیره کنیم و آدرس آن در حافظه را در رجیسترها بریزیم که این روش تعداد زیادی پارامتر را هم پشتیبانی میکند. در این روش سرعت پایین تر است چون ارتباط مستقیم با حافظه زمان‌برتر از ارتباط با رجیسترها است. استفاده از هر سه روش بنا به تعداد پارامترها بهترین حالت است. اگر تعداد پارامترها زیاد باشد باید از روش سوم که کند تر است استفاده کنیم و در غیر این صورت می‌توانیم پارامترها را در رجیستر ذخیره کنیم.

به نام خدا



دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

سیستم‌های عامل- گروه ۱ (نیم‌سال دوم ۱۳۹۹-۱۴۰۰)

تمرین شماره ۲

مدیریت پردازه (**process management**)

آخرین تاریخ بارگذاری پاسخ در **courses**:

ساعت ۲۳:۵۹ روز ۱۰ اردیبهشت ۱۳۹۹

سوال ۱

برنامه‌ی زیر به زبان پایتون را در نظر بگیرید که یک عدد از کاربر گرفته و مشخص می‌کند که آیا آن عدد زوج است یا فرد.

```
def main():
    number = int(input())
    result = 'Even' if number % 2 == 0 else 'Odd'
    print(result)
main()
```

فرض کنید که تنها دستورات input و print بلافاصله انجام نمی‌شوند و نیازمند صبر کردن برای I/O هستند. حالات (process states) مختلف پردازه این برنامه را در طول چرخه حیات آن بررسی کنید.

سوال ۲

فرض کنید می‌خواهید یک سیستم‌عامل طراحی کنید که Process Control Block در آن تنها ۵ فیلد مختلف می‌تواند داشته باشد، این فیلدها را چه چیزی‌هایی انتخاب می‌کنید؟ فرضیات خود درباره سیستم عامل‌تان را بیان و دلایل خود را ذکر کنید. در چه مواردی ممکن است این سیستم عامل در context switch با مشکل مواجه شود؟

تذکر: هر فیلد PCB باید شامل یک عدد یا رشته یا آرایه‌ای از اعداد و رشته ها باشد، همچنین این اعداد/رشته ها نباید داده های multiplex شده باشند و باید داده خام و مستقیما قابل استفاده باشند، برای مثال نمیتوانید

program_counter * 10000 + pid

را به عنوان یک فیلد اختصاص دهید زیرا ۲ داده multiplex شدند تا یک داده را تشکیل دهند.

سوال ۳

۱. پردازه‌های independent و cooperating به چه پردازه‌های گفته می‌شود و چه ویژگی‌های دارند؟ توضیح دهید.
۲. چرا به ارتباط میان پردازه‌ها نیاز داریم؟ ۳ دلیل برای آن بیاورید و آن‌ها را توضیح دهید.
۳. چگونه می‌توان با ایجاد ارتباط همکاری میان پردازه‌ها باعث سریع‌تر انجام شدن یک برنامه شد؟

سوال ۴

خروجی کدهای زیر را با رسم شکل درختواره پردازه‌ها توضیح دهید (سعی کنید ابتدا بر روی کاغذ تحلیل خود را انجام دهید (انچه که در پاسخ تمرین ارسال خواهید کرد) و سپس برنامه را کامپایل و اجرا کنید):

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
     if (fork()) {
          if (!fork()) {
               fork();
               printf("1 ");
          }
          else {
               printf("2 ");
          }
     }
     else {
          printf("3 ");
     }
     printf("4 ");
     return 0;
}
```

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
     if (fork() && (!fork())) {
          if (fork() || fork()) {
               fork();
          }
     }
     printf("2 ");
     return 0;
}
```

سوال ۵

در مورد shared memory به سوالات زیر پاسخ دهید.

۱. این فضای ذخیره‌سازی مشترک در کجا قرار داد؟
۲. پردازه‌های دیگری که تمایل به استفاده از این فضا دارند چگونه می‌توانند به آن متصل شوند؟
۳. پس از اتصال چند پردازه به این فضای مشترک چگونه می‌توانند داده‌ها را با هم به اشتراک بگذارند و ارتباط برقرار کنند؟
۴. در پروسه‌ی ارتباط پردازه‌ها با روش shared memory سیستم‌عامل چه نقشی دارد و تا چه حد دسترسی دارد؟
۵. در حین ارتباط با روش shared memory چه مشکلاتی ممکن است به وجود بیاید و چه بخشی مسئول حل این مشکلات است؟
۶. مشکل producer-consumer در حین استفاده از shared memory چیست و چگونه میتوان آن را حل کرد؟

نحوه تحویل تمرین

پاسخ به سوالات را در قالب یک فایل پی دی اف (اسکن یا تایپ شده) با نام «HW2_student_id» در صفحه درس اپلود کنید. پاسخ‌های شما بایستی دقیق و خوانا باشند.

جریمه دیرکرد

هر روز تاخیر در ارسال تمرین ٪۱۰ نمره منفی خواهد داشت. امکان اپلود تمرین تنها تا ۵ روز از تاریخ تعیین شده ممکن خواهد بود.

جریمه تقلب

۱. همه دانشجویان بایستی که خود تمرین را انجام دهند و هرگونه تقلب یا ارسال کار دیگران یا کارهای موجود در وب که تمرین را به شکل جزئی یا کلی انجام داده است، غیرقابل پذیرش و عواقب شدیدی خواهد داشت.
۲. بنده و گروه حل تمرین تمام تلاش خود را برای شناسایی تقلب‌های احتمالی خواهیم کرد تا در نهایت یک ارزیابی عادلانه از همه دانشجویان عزیز داشته باشیم. ما از Moss برای شناسایی فایل‌های مشابه استفاده خواهیم کرد.
۳. در صورت شناسایی تقلبی که ۵۰ درصد یا پایینتر از کار را شامل می‌شود، دانشجوی مورد نظر اخطار اول را دریافت کرده و نمره «۰.۵- * بارم تمرین ۱» به ایشان تعلق می‌گیرد و در صورت شناسایی تقلبی که بیشتر از ۵۰ درصد کار را پوشش می‌دهد به دانشجوی مورد نظر اخطار دوم تعلق گرفته و نمره «۱- * بارم تمرین ۱» به ایشان تعلق می‌گیرد. علاوه بر این نمره منفی، گرفتن دو اخطار، به معنی لحاظ شدن نمره منفی برابر با بارم همه تمرینات خواهد بود.

در نهایت، هر گونه سوال در مورد تمرین و بخش‌های آنها را تنها و تنها از طریق سایت درس و ایجاد مباحثه با عناوین مرتبط مطرح بفرمایید.

تندرست و موفق باشید

تیم درس سیستم‌های عامل

سوال ۱

برنامه‌ی زیر به زبان پایتون را در نظر بگیرید که یک عدد از کاربر گرفته و مشخص می‌کند که آیا آن عدد زوج است یا فرد.

```
:()def main
    (()number = int(input
    'result = 'Even' if number % 2 == 0 else 'Odd
        (print(result
()main
```

فرض کنید که تنها دستورات input و print بلافاصله انجام نمی‌شوند و نیازمند صبر کردن برای I/O هستند. حالات (process states) مختلف پردازه این برنامه را در طول چرخه حیات آن بررسی کنید.

از آنجایی که پایتون یک زبان مفسری است تمامی کد های این زبان توسط یک instance از مفسر آن اجرا می‌شوند.

با دادن دستور اجرای این کد توسط سیستم عامل یک پردازه جدید برای اجرای مفسر ایجاد شده و در حالت new قرار میگیرد.

پس از مقداردهی های اولیه و لود شدن مفسر در حافظه و نسبت داده شدن کد به آن پردازه در حالت ready قرار میگیرد.

پس از تخصیص cpu به این پردازه، پردازه در حالت running قرار میگیرد

به محض رسید به input() پردازه درخواست I/O را ارسال کرده و در حالت running قرار میگیرد

پس از رسیدن I/O (وارد کردن یک عدد توسط کاربر) و انتقال آن به پردازه توسط سیستم عامل پردازه در حالت ready قرار میگیرد.

پس از تخصیص cpu به این پردازه، پردازه در حالت running قرار میگیرد

به محض رسیدن به دستور print پردازه درخواست I/O را ارسال کرده و در حالت waiting قرار میگیرد

بعد از انجام درخواست I/O توسط سیستم عامل (چاپ کردن result در std out) پردازه در حالت ready قرار میگیرد.

پس از تخصیص cpu به این پردازه، پردازه در حالت running قرار میگیرد و از آنجایی که کد دیگری برای اجرا ندارد پایان یافته و در حالت terminated قرار میگیرد.

سوال ۲

فرض کنید می‌خواهید یک سیستم‌عامل طراحی کنید که Process Control Block در آن تنها ۵ فیلد مختلف می‌تواند داشته باشد، این فیلدها را چه چیزی‌هایی انتخاب می‌کنید؟ فرضیات خود درباره سیستم عامل‌تان را بیان و دلایل خود را ذکر کنید. در چه مواردی ممکن است این سیستم عامل در context switch با مشکل مواجه شود؟
تذکر: هر فیلد PCB باید شامل یک عدد یا رشته یا آرایه‌ای از اعداد و رشته ها باشد، همچنین این اعداد/رشته ها نباید داده های multiplex شده باشند و باید داده خام و مستقیما قابل استفاده باشند، برای مثال نمیتوانید

program_counter * 10000 + pid

را به عنوان یک فیلد اختصاص دهید زیرا ۲ داده multiplex شدند تا یک داده را تشکیل دهند.

فیلد هایی که در یک PCB وجود دارند عبارتند از:

Process State, Process ID, Program Counter, CPU Registers, CPU-Scheduling Information, Memory-management Information, Accounting Information, I/O status information

از بین این فیلد ها برای اینکه بتوانیم context switch موفقیت آمیز بین دو پردازه داشته باشیم لازم است که فیلد های program counter, cpu registers , memory-management information و I/O status information را داشته باشیم (فیلد پنجم هر یک دیگر فیلد ها میتواند باشد و هر کدام بخش خاصی از فرایند را بهبود میدهند).

Program counter:

برای ادامه دادن یک برنامه باید بدانیم که دستور بعدی که باید اجرا کنیم چیست.

CPU registers:

مهم است که از دید برنامه در حال اجرا pre-empt شدن هیچ تاثیری نگذارد در نتیجه state cpu باید ذخیره و برای ادامه برنامه بازیابی شود

Memory-management information:

برنامه باید به متغیر های خود در ram دسترسی داشته باشد و محل آن ها را بداند.

I/O status information:

مانند متغیر های داخل رم برنامه باید به I/Oهایی که در دسترسش قرار دارند (مانند فایل های باز شده ویا سوکت های شبکه) دسترسی داشته باشد.

در راه‌حل بالا context switch به عنوان عملکرد اصلی‌ای که باید PCB به سیستم عامل ارائه دهد فرض شده، در صورتی که عملکرد اصلی‌ای که فرض کردید متفاوت باشد جواب نیز میتواند فرق کند.

سوال ۳ (بخش ۳.۴ کتاب)

۱) پردازه‌های independent و cooperating به چه پردازه‌های گفته می‌شود و چه ویژگی‌های دارند؟ توضیح دهید.

یک پردازه independent است اگر هیچ داده و اطلاعاتی با دیگر پردازه‌هایی که در سیستم در حال اجرا شدن هستند به اشتراک نگذارد.

یک پردازه cooperating است در صورتی که بتواند روی دیگر پردازه‌های در حال اجرا تاثیر بگذارد یا از آن‌ها تاثیر ببیند. به وضوح هر پردازه‌ای که با دیگر پردازه‌های موجود در سیستم اطلاعات به اشتراک بگذارد از نوع cooperating است.

۲) چرا به ارتباط میان پردازه‌ها نیاز داریم؟ ۳ دلیل برای آن بیاورید و آن‌ها را توضیح دهید.

۱) به اشتراک گذاری اطلاعات(Information sharing): از آنجایی که برنامه‌های متعددی ممکن است به یک اطلاعات خاص نیاز داشته‌باشند باید روشی برای به اشتراک گذاری این اطلاعات بین آن‌ها ایجاد کنیم.

۲) سرعت محاسبه(Computation speedup): یکی از روش‌های معمول افزایش سرعت انجام تسک‌ها شکاندن آن‌ها به تسک‌های کوچک‌تر است به شکلی که هر زیر مجموعه از تسک اصلی به صورت موازی با دیگر بخش‌ها اجرا شود. این اجرای موازی مستلزم به اشتراک گذاری اطلاعات میان تسک‌های کوچک است.

۳) ماژولار بودن(Modularity): اگر بخواهیم سیستم را به شکل ماژولار بسازیم(یعنی آن را به بخش‌های کوچک‌تر تقسیم کنیم) نیازمند کمک گرفتن از پردازه‌های cooperating هستیم.

۳) چگونه می‌توان با ایجاد ارتباط همکاری میان پردازه‌ها باعث سریع‌تر انجام شدن یک برنامه شد؟

اگر یک تسک را به تسک‌های کوچک‌تر(subtask) بشکنیم به شکلی که این بخش‌های کوچک به صورت موازی با هم اجرا شوند افزایش زیادی در سرعت اجرای برنامه ایجاد می‌شود. در این صورت هر subtask بخشی از کار را به شکل موازی با

subtaskهای دیگر انجام میدهد که به وضوح باعث سریع‌تر شدن برنامه می‌شود.
این مورد هم همانطور که گفته‌شد نیاز به پردازه‌های cooperating و ایجاد ارتباط میان پردازه‌ها دارد.

سوال ۴

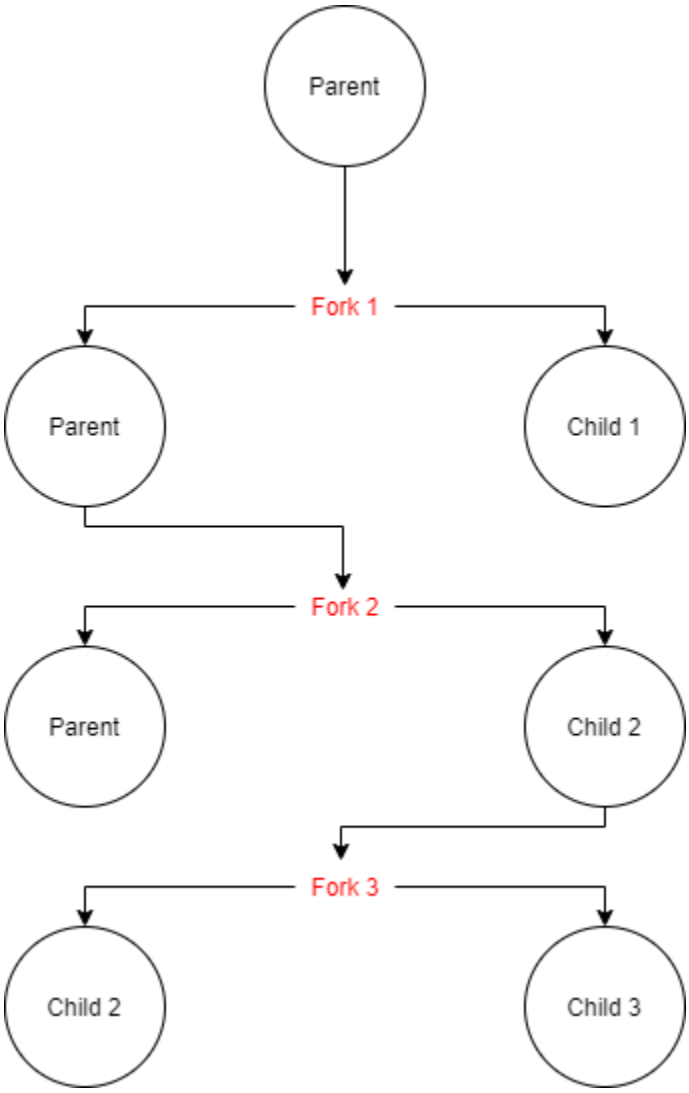
خروجی کدهای زیر را با رسم شکل درختواره پردازه‌ها توضیح دهید (سعی کنید ابتدا بر روی کاغذ تحلیل خود را انجام دهید (آنچه که در پاسخ تمرین ارسال خواهید کرد) و سپس برنامه را کامپایل و اجرا کنید):

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    if (fork()) {
        if (!fork()) {
            fork();
            printf("1 ");
        }
        else {
            printf("2 ");
        }
    }
    else {
        printf("3 ");
    }
    printf("4 ");
    return 0;
}
```

در ابتدا برای پردازه P یک فرزند C1 ساخته می‌شود. پردازه پدر وارد قسمت if و فرزند وارد else می‌شود. پردازه پدر فرزند دیگری به اسم C2 ساخته و وارد else می‌شود. پردازه C2 فرزندی به اسم

C3 ساخته و با پرینت ۱ و ۴ کار خود را تمام می‌کند. پردازه C3 نیز ۱ و ۴ را پرینت می‌کند. پردازه پدر اعداد ۲ و ۴ را پرینت کرده و پردازه C1 با پرینت کردن ۳ و ۴ به اتمام می‌رسد. دقت کنید هیچ تضمینی در ترتیب پرینت شدن این اعداد وجود ندارد.



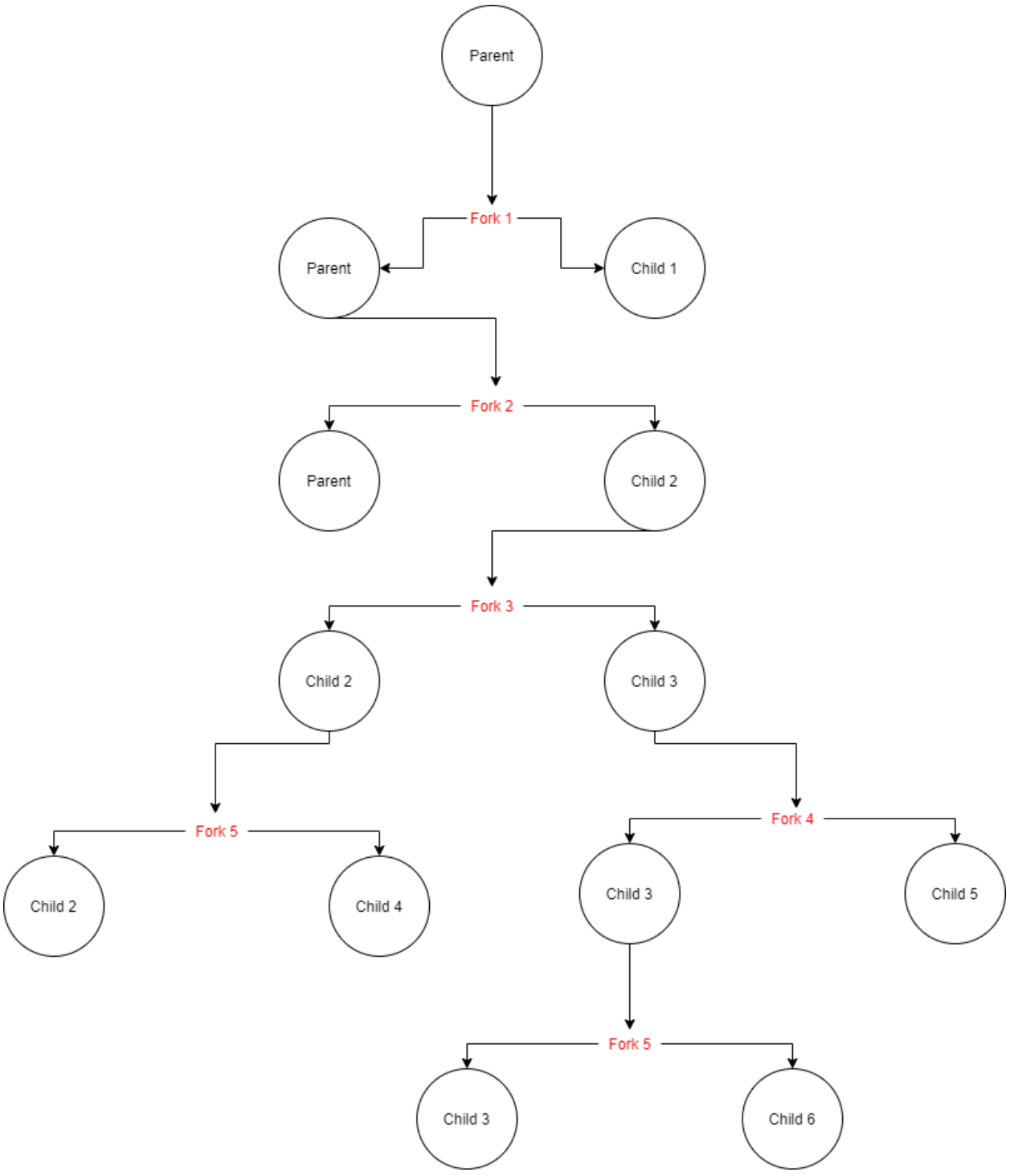
```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    if (fork() && (!fork())) {
        if (fork() || fork()) {
            fork();
        }
    }
    printf("2 ");
    return 0;
}
```

ابتدا پردازه پدر یک فرزند به اسم C1 می‌سازد. به دلیل وجود اپراتور && ادامه شرط دیگر برای فرزند اجرا نمی‌شود و تنها برای پردازه پدر بار دیگر اجرا شده و یک فرند دیگر به نام C2 ساخته می‌شود. در نتیجه پردازه پردازه C2 وارد شرط می‌شود و پردازه C1 و پردازه پدر از if خارج شده و دو بار عدد ۲ را چاپ می‌کنند. پردازه C2 وارد شرط شده و در اجرای اولین fork یک فرزند C3 را ایجاد می‌کند. سپس به دلیل true بودن و وجود اپراتور || وارد شرط می‌شود و در نتیجه با اجرای fork داخل شرط یک فرزند دیگر به اسم C4 ساخته و به کار پایان می‌دهد.

پردازه C3 وارد fork دوم شرط شده و فرزند C5 را ایجاد می‌کند و خود این بار وارد شرط می‌شود. در حالی که C5 خارج می‌شود. در ادامه تمامی آن‌ها عدد ۲ را پرینت کرده و اجرا تمام می‌شود.

نمی‌توان ترتیب اجرا شدن را تضمین کرد ولی به صورت کلی ۷ بار عدد ۲ پرینت می‌شود.


Parent
Fork 1
Parent
Child 1
Fork 2
Parent
Child 2
Fork 3
Child 2
Child 3
Fork 5
Fork 4
Child 2
Child 4
Child 3
Child 5
Fork 5
Child 3
Child 6

سوال ۵(بخش ۳.۵ کتاب)

در مورد shared memory به سوالات زیر پاسخ دهید

1) این فضای ذخیره‌سازی مشترک در کجا قرار داد؟
این فضا در فضای آدرسی(address space) پردازه‌ای که آن را میسازد قرار دارد.

2) پردازه‌های دیگری که تمایل به استفاده از این فضا دارند چگونه میتوانند به آن متصل شوند؟
پردازه‌های دیگری که میخواهند به از این فضای اشتراکی استفاده کنند باید آن را به فضای آدرسی خود متصل کنند. البته به طور معمول سیستم‌عامل اجازه‌ی دسترسی به فضای آدرسی یک پردازه را به پردازه‌های دیگر نمیدهد و به همین خاطر این پروتکل نیاز دارد تا پردازه‌های درگیر قبول کنند که این محدودیت را بردارند.

3) پس از اتصال چند پردازه به این فضای مشترک چگونه میتوانند داده‌ها را با هم به اشتراک بگذارند و ارتباط برقرار کنند؟
این پردازه‌ها پس از اتصال به فضای اشتراکی میتوانند با نوشتن و خواندن داده‌های موجود در محیط اشتراکی به رد و بدل اطلاعات و داده‌ها بپردازند. از آنجایی که محدودیت مربوط به سیستم عامل هنگام اتصال پردازه‌ها به محیط اشتراکی برداشته شده‌است، فرم داده و مکان آن و به طور کلی مسئولیت آن با خود پردازه‌هاست و سیستم عامل دیگر نقشی ندارد.

4) در پروسه‌ی ارتباط پردازه‌ها با روش shared memory سیستم‌عامل چه نقشی دارد و تا چه حد دسترسی دارد؟
همانطور که گفته‌شد سیستم‌عامل به طور معمول اجازه‌ی دسترسی به فضای آدرسی دیگر پردازه‌ها را به یک پردازه نمیدهد به همین خاطر قبل از اتصال پردازه به فضای اشتراکی باید با کمک فراخوانی‌های سیستمی(system call) به سیستم عامل اطلاع دهند که قصد ایجاد چنین فضایی را دارند تا محدودیت‌های مربوط به آن برداشته‌شود. پس از برداشته‌شدن محدودیت‌ها سیستم عامل دیگر در فرایند انتقال داده نقشی ندارد و مسئولیت آن به عهده پردازه‌هاست.

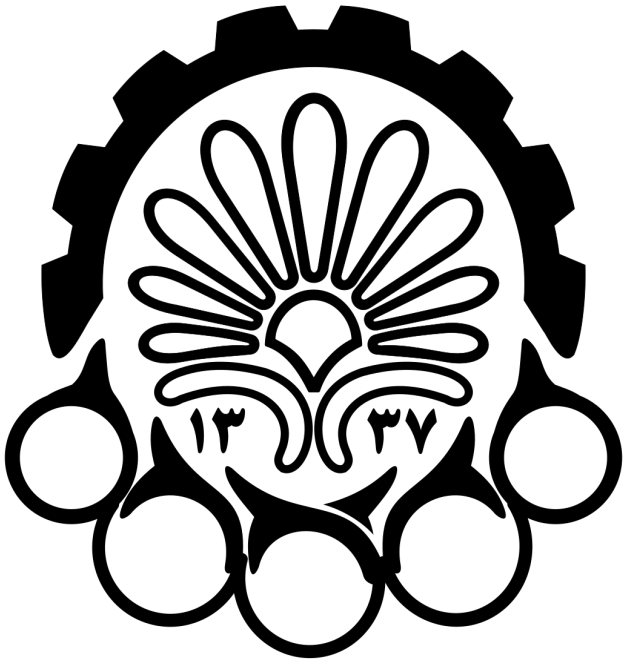
5) در حین ارتباط با روش shared memory چه مشکلاتی ممکن است به وجود بیاید و چه بخشی مسئول حل این مشکلات است؟
ممکن است چند پردازه بخواهند به شکل همزمان روی یک مکان اطلاعات بنویسند

و مشکلاتی مانند race condition به وجود بیاید و یا پردازه‌ای بخواهد داده‌ی خاصی را بخواند ولی پیش از آن پردازه‌ی دیگری آن را تغییر دهد. از آنجایی که محدودیت‌های اعمالی از سیستم‌عامل پیش از شروع اشتراک گذاری برداشته شده‌اند دیگر سیستم عامل نقشی ندارد و مسئول جلوگیری از مشکلات و حل آن‌ها خود پردازه‌ها هستند.

6) مشکل producer-consumer در حین استفاده از shared memory چیست و چگونه میتوان آن را حل کرد؟

این روش یک الگوی مشترک میان پردازه‌های cooperating است. به این شکل که پردازه‌ی producer اطلاعاتی را تولید میکند که پردازه‌ی consumer از آن‌ها استفاده میکند.(مانند اتفاقی که در پروتکل client-server رخ میدهد.) حال مشکلی که ممکن است رخ دهد این است که چگونه این فرایند ایجاد شود به طوری که مثلا زمانی که اطلاعاتی وجود ندارد پردازه‌ی consumer شروع به خواندن نکند یا زمانی که جایی برای اطلاعات وجود ندارد پردازه‌ی producer شروع به نوشتن آن‌ها نکند. برای حل این مشکل میتوان از یک بافر استفاده کرد به شکلی که producer در آن بنویسد و آن را پر کند و consumer از آن بخواند و آن را خالی کند. در این حالت باید producer و consumer به شکل سنکرون عمل کنند تا مشکلات ناشی از ناهماهنگی به وجود نیاید مثلا consumer قبل از نوشتن داده توسط producer شروع به خواندن آن نکند. حال ۲ نوع مختلف از بافر را میتوان استفاده کرد. نوع اول سایز محدودی ندارد. در این حالت دیگر تولید کننده نیازی به صبر کردن ندارد و هر زمان که خواست فارغ از میزان داده‌ی موجود در بافر میتواند در آن بنویسید زیرا محدودیت سایزی در بافر وجود ندارد اما از سمت دیگر مصرف کننده همچنان ممکن است در حالتی که بافر خالی است مجبور به صبر کردن شود. نوع دوم بافر سایز محدود دارد و هر دوی مصرف کننده و تولید کننده ممکن است مجبور به صبر کردن شوند زیرا در این حالت ممکن است بافر پر باشد و تولید کننده نتواند هر زمان که خواست در آن بنویسد. این بافر به صورت حلقوی پیاده‌سازی میشود و با داشتن دو متغیر in و out میتواند متوجه شد که بافر چقدر فضای خالی دارد و آیا امکان نوشتن روی آن وجود دارد یا نه.(اشکال ۳.۱۲ و ۳.۱۳)

به نام خدا



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

سیستم‌های عامل- گروه ۱ (نیم‌سال دوم ۱۳۹۹-۱۴۰۰)

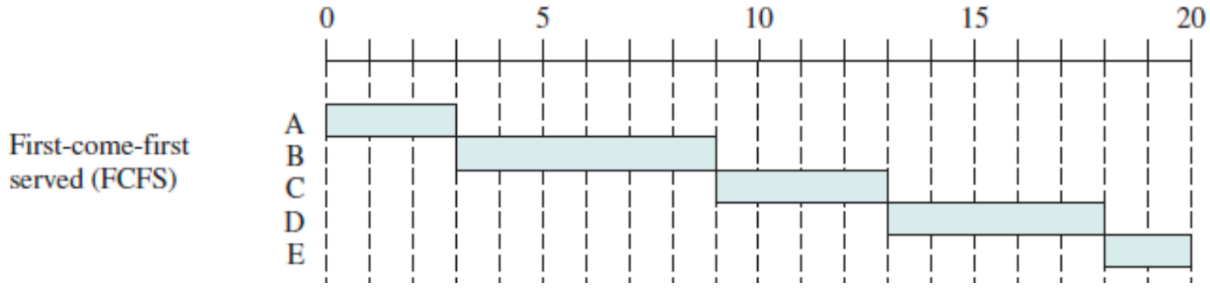
تمرین شماره ۳
زمان‌بندی و همگام سازی
(Scheduling & Synchornization)
آخرین تاریخ بارگذاری پاسخ در courses:
ساعت ۲۳:۵۹ روز۲۱ خرداد ۱۴۰۰

سوال ۱

سیستمی با شرایط زیر را در نظر بگیرید:

Process	Burst Time	Priority	Arrival Time
P1	50 ms	4	0 ms
P2	20 ms	1	20 ms
P3	100 ms	3	40 ms
P4	40 ms	2	60 ms

زمانبندی پردازنده را با استفاده از الگوریتم‌های non-preemptive priority، SJF، FCFS و RR (با کوانتوم زمانی ۳۰ میلی ثانیه) را همانند مثال زیر ترسیم کنید.



سوال ۲

مجموعه فرآیندهای زیر را با مدت زمان پردازنده (CPU burst time) بر حسب میلی ثانیه در نظر بگیرید:

Process	Burst Time	Priority
P_1	2	2
P_2	1	1
P_3	8	4
P_4	4	2
P_5	5	3

فرض میشود فرآیندها در لحظه ۳ به ترتیب p1، p2، p3، p4 و p5 وارد میشوند. الگوریتمهای non-preemptive priority، SJF، FCFS (شماره بزرگتر، نشان از اولویت بالاتر است) و نوبت چرخشی، را در نظر بگیرید:

الف) زمان رفت و برگشت (Turnaround time) هر فرآیند با هر الگوریتم زمانبندی فوق چقدر است؟

ب) زمان انتظار (Waiting time) هر فرآیند با هر الگوریتم زمانبندی فوق چقدر است؟

ج) کدام یک از الگوریتم‌های زمانبندی، میانگین زمان انتظار کمینه (برای کل فرآیندها) را دارد؟

سوال ۳

نشان دهید اگر عملیات (wait) و (signal) سمافور به صورت اتمیک اجرا نشوند، در این صورت امکان نقض انحصار متقابل وجود دارد.

سوال ۴

در راه حل زیر برای مسئله فیلسوفان غذاخور، فرض کنید روالهای برداشتن دو چنگال و گذاشتن هر چنگال از ابتدا تا انتها با رعایت کامل انحصار متقابل (Mutual Exclusion) انجام می‌شود. روال take_forks(i) دو چنگال راست و چپ را به ترتیب بررسی می‌کند و اگر هر دو موجود بودند بر می‌دارد، در غیر این صورت عمل بررسی تکرار می‌شود. روال put_fork(i) چنگال شماره i را می‌گذارد و از روال خارج میشود.

الف) بررسی نمایید آیا ممکن است بن بست پیش بیاید؟ اگر بله، مثال بیاورید.

ب) بررسی نمایید آیا ممکن است قحطی پیش بیاید؟ اگر بله، مثال بیاورید.

```
void philosopher(int i) {
    Think();
    take_forks(i);
    Eat();
    put_forks(i);
    put_forks((i+1)%n);
}
```

سوال ۵

فرض کنید یک نانوایی وجود دارد که در مقابل آن دو صف تشکیل شده است. در هر لحظه فقط یک نفر میتواند در مقابل نانوا قرار گرفته و از او نان بگیرد. با استفاده از سمافور ها راه حلی ارائه کنید که :

بیش از یک نفر مقابل نانوا قرار نگیرد و ثانیا هیچ یک از دو صف دچار قحطی نشوند.

همچنین می توانید پیاده سازی های توابع signal و wait را مطابق زیر فرض کنید:

```
wait(semaphore *S) {
  S->value--;
  if (S->value < 0) {
    add this process to S->list;
    block();
  } }
```

```
signal(semaphore *S) {
  S->value++;
  if (S->value <= 0) {
    remove a process P from S->list;
    wakeup(P);
  } }
```

سوال ۶

شروط انحصار متقابل، پیشرفت و انتظار محدود را برای الگوریتم زیر بررسی کنید و دلیل خود را بنویسید.

```
do {
        flag[i] = true;
        turn = j;
        while (!flag[j] || turn = = j);
                critical section
        flag[i] = false;
                remainder section
} while (true);
```

سوال ۷

انتظار مشغول (busy waiting) چیست؟ سایر انتظارهای موجود درسیستم‌عامل کدامند؟ آیا به صورت کلی می‌توان از انتظار مشغول اجتناب کرد؟ پاسخ خود را توضیح دهید.

نحوه تحویل تمرین

پاسخ به سوالات را در قالب یک فایل پی دی اف (اسکن یا تایپ شده) با نام «HW3_Student_ID» در صفحه درس اپلود کنید. پاسخ‌های شما بایستی دقیق و خوانا باشند.

جریمه دیرکرد

هر روز تاخیر در ارسال تمرین ٪۱۰ نمره منفی خواهد داشت. امکان اپلود تمرین تنها تا ۵ روز از تاریخ تعیین شده ممکن خواهد بود.

جریمه تقلب

۱. همه دانشجویان بایستی که خود تمرین را انجام دهند و هرگونه تقلب یا ارسال کار دیگران یا کارهای موجود در وب که تمرین را به شکل جزئی یا کلی انجام داده است، غیرقابل پذیرش و عواقب شدیدی خواهد داشت.
۲. بنده و گروه حل تمرین تمام تلاش خود را برای شناسایی تقلب‌های احتمالی خواهیم کرد تا در نهایت یک ارزیابی عادلانه از همه دانشجویان عزیز داشته باشیم. ما از Moss برای شناسایی فایل‌های مشابه (تمارین عملی) استفاده خواهیم کرد.
۳. در صورت شناسایی تقلبی که ۵۰ درصد یا پایینتر از کار را شامل می‌شود، دانشجوی مورد نظر اخطار اول را دریافت کرده و نمره «۰.۵- * بارم تمرین ۱» به ایشان تعلق می‌گیرد و در صورت شناسایی تقلبی که بیشتر از ۵۰ درصد کار را پوشش می‌دهد به دانشجوی مورد نظر اخطار دوم تعلق گرفته و نمره «۱- * بارم تمرین ۱» به ایشان تعلق می‌گیرد. علاوه بر این نمره منفی، گرفتن دو اخطار، به معنی لحاظ شدن نمره منفی برابر با بارم همه تمرینات خواهد بود.

در نهایت، هر گونه سوال در مورد تمرین و بخش‌های آنها را **تنها و تنها** از طریق سایت درس و ایجاد مباحثه با عناوین مرتبط مطرح بفرمایید.

تندرست و موفق باشید

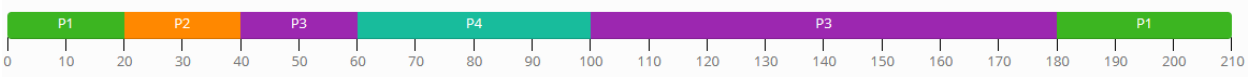
تیم درس سیستم‌های عامل

پاسخنامه تمرین سوم سیستم‌های عامل

سوال ۱)

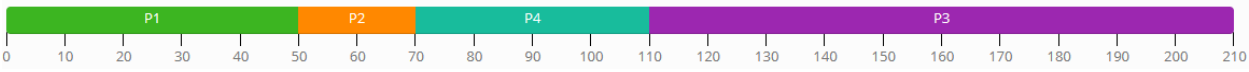
نمودار زمان‌بندی پردازنده برای پردازه‌های ذکر شده به شکل زیر خواهد بود:

Priority:

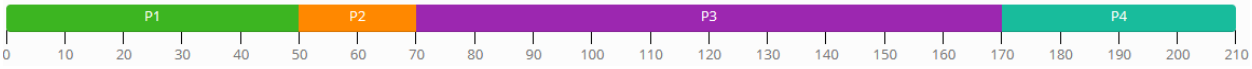


(در صورت non-preemptive بودن، پردازه‌های P1، P3، P4 و P2 به صورت متوالی و پشت سرهم تماما اجرا می‌شوند.)

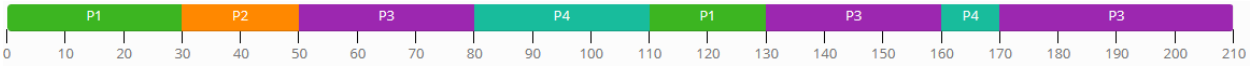
SJF:



FCFS:



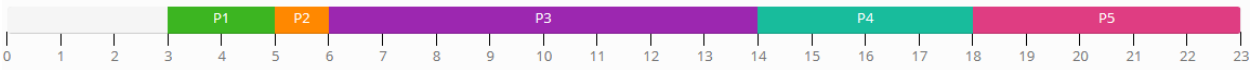
RR:



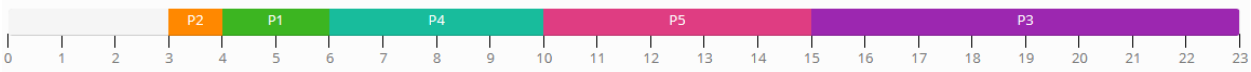
سوال ۲)

زمان‌بندی پردازنده برای هر الگوریتم مطابق موارد زیر خواهد بود:

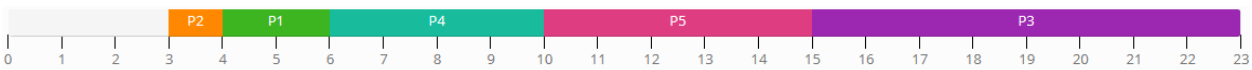
FCFS:



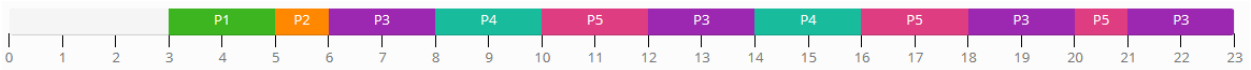
SJF:



Non-preemptive priority:



RR:



با توجه به زمان‌بندی‌ها فوق، به سوالات پاسخ می‌دهیم:

الف) مقدار Turnaround time هر پردازه:

	FCFS	SJF	Non-preemptive priority	RR
P1	2	3	3	2
P2	3	1	1	3
P3	11	20	20	20
P4	15	7	7	13
P5	20	12	12	18

ب) مقدارwaiting time هر پردازه:

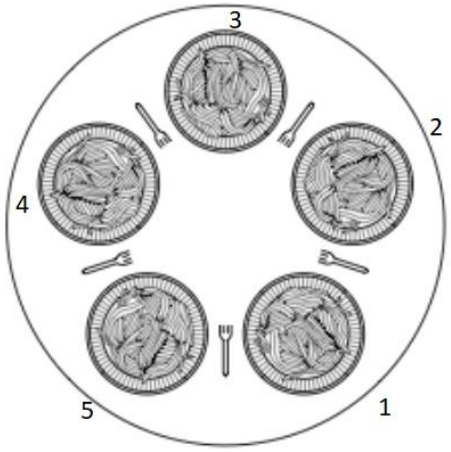
	FCFS	SJF	Non-preemptive priority	RR
P1	0	1	1	0
P2	2	0	0	2
P3	3	12	12	12
P4	11	3	3	9
P5	15	7	7	13
Average	6.2	4.6	4.6	7.2

ج) مطابق جدول قبل، زمان انتظار کمینه متعلق به الگوریتم SJF (و در اینجا Non-preemptive priority) می‌باشد.

سوال ۳) یک پیاده‌سازی بسیار ساده از wait و signal برای سمافور در زیر مشاهده می‌شود:

```
wait(Semaphore s){
    while(s<=0);
    s.value = s.value - 1;
}
signal(Semaphore s){
    s.value = s.value + 1;
}
```

اگر توابع فوق به صورت اتمی اجرا نشوند، امکان قبضه شدن (preempt) شدن در هر نقطه از مراحل هرکدام از توابع وجود دارد، بنابراین اگر یک سمافور دارای مقدار ۱ باشند و wait برای آن توسط دو پردازه به صورت همزمان اجرا شود، امکان دارد که هردو به صورت همزمان اقدام به کم‌کردن مقدار s.value کنند، که باعث نقض قاعده‌ی انحصار متقابل می‌شود.



سوال ۴) روش ذکر شده در این سوال، نوعی از روش Arbitrator solution برای حل مساله‌ی فیلسوف‌های خورنده می‌باشد. این روش با فرض اینکه تابع take_forks با استفاده از روش‌های قفل از race condition جلوگیری می‌کند، مشکل بن‌بست (Dead lock) را حل می‌کند؛ ولی مشکل قحطی همچنان پابرجاست.

الف) همانطور که ذکر شد، مشکل Dead lock وجود ندارد.

ب) بله، به دلیل به ترتیب گذاشتن چنگال‌ها امکان قحطی وجود دارد؛ یک مثال این حالت در ادامه بیان شده است:

فیلسوف‌های ۱ و ۳ در حال خوردن هستند، و فیلسوف ۲ منتظر خوردن است. حال ۳ چنگال‌هایش را می‌گذارد، ولی ۲ همچنان نمی‌تواند شروع به خوردن کند (چرا که ۱ نیز باید چنگال‌هایش را بگذارد). سپس قبل از اینکه ۱ چنگالش را بگذارد، ۳ دوباره شروع به خوردن می‌کند. این بار با اتمام خوردن ۱، فیلسوف ۲ باید منتظر ۳ بماند. اگر این چرخه تکرار شود، همواره فیلسوف ۲ از خوردن باز می‌ماند.

سوال ۵) یک راه حل ساده برای این مساله، می‌تواند به شکل زیر باشد:

```
line_1(){
    while (true) {
        wait(mutex);
        serve_bread();
        signal(mutex);
    }
}
```

```
line_2(){
    while (true) {
        wait(mutex);
        serve_bread();
        signal(mutex);
    }
}
```

سوال ۶)

```
do {
        flag[i] = true;
        turn = j;
        while (!flag[j] || turn = = j);
                critical section
        flag[i] = false;
                remainder section
} while (true);
```

این روش که به روش Peterson مشهور است، هر سه شرط انحصار متقابل، پیشرفت و انتظار محدود را ارضا می‌کند.

تنهل مشکل این شرط وجود انتظار مشغول است.

سوال ۷) انتظار مشغول تکنیکی است که در آن یک فرایند، مکرراً یک شرط خاص را بررسی می‌کند تا ببیند آیا آن شرط برقرار است یا خیر؛ مثلا یک حلقه while بدون اینکه قطعه کدی را اجرا کند، فقط در انتظار باطل شدن شرط تکرار برای خروج است. وجود این نوع انتظار باعث می‌شود که یک پردازه بدون اینکه کار مفیدی انجام دهد، منابع اجرایی سیستم را در اختیار داشته باشد و در ظاهر مشغول به کار باشد، در حالی که کاری انجام نمی‌شود.

علاوه بر انتظار مشغول، انتظار محدود (Bounded wait)، انتظار چرخش (Spin wait)، انتظار مسدود شده (blocked wait یا Sleep wait) و انتظارهایی مانند انتظار برای I/O و انتظار پردازه برای شروع به اجرا (تغییر وضعیت از ready به run) وجود دارد.

مشکل انتظار مشغول از طریق استفاده از sleep یا Block کردن یک پردازه قابل جلوگیری است، البته این روش‌ها نیز برای پردازه سربار خودشان را دارند.