

کتاب کوچک سما فورہا

آلن بی. دونی

مترجمین:

سید محمد جواد رضویان، محمد مهدی قاسمی نیا و سید علی آل طہ

نسخہ ۲/۲/۱

کتاب کوچک سمافورها

ویرایش دوم

نسخه ۲/۲/۱

حق نشر ۲۰۱۶ آلن بی. دونی

کپی، توزیع و/یا تغییر این سند تحت لایسنس زیر مجاز است:

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

(CC BY-NC-SA 4.0) <http://creativecommons.org/licenses/by-nc-sa/4.0>

فرم اصلی این کتاب یک سورس کد لاتک است. کامپایل این سورس لاتک سبب تولید یک نمایش کتاب بدون وابستگی به دستگاه خواهد شد که می‌تواند به دیگر فرمت‌ها تبدیل و چاپ گردد.

این کتاب توسط نویسنده با کمک لاتک، dvips و ps2pdf که همگی برنامه‌های کدباز هستند تایپ شده

است. سورس لاتک این کتاب در آدرس <http://greenteapress.com/semaphores> موجود

است.

پیشگفتار

غالب کتاب‌های درسی سیستم‌های عامل در دوره کارشناسی بخشی در همگام سازی دارند که به طور معمول شامل معرفی اجزای اولیه‌ای (موتکس، سمافور، ناظر و متغیرهای شرطی) و مسائل کلاسیک مثل خواننده نویسنده و تولیدکننده مصرف کننده. وقتی که من در برکلی کلاسی سیستم عامل را داشتم، و در کالج کالبی این درس را تدریس کردم، به این نتیجه رسیدم که بیشتر دانشجویان قادر به درک راه حل ارائه شده برای اینگونه مسائل هستند، اما تنها برخی از این دانشجویان توانایی [ارئه چنین راه حل‌هایی] [ارئه همان راه حل‌ها] و حل مسائل مشابه را دارند.

یکی از دلایلی که دانشجویان نمی توانند به طور عمیق این قبیل مسایل را بفهمند، این است که وقت و تلاش بیشتری می برند از آنچیزی که کلاس‌ها در اختیارشان می گذارد. همگام سازی یکی از ماژول‌هایی است که نسبت به دیگر ماژول‌ها وقت بیشتری نیاز دارد. و من مطمئن نیستم که بتوانم برای این منظور دلایلی را شرح دهم، منتها من فکر می کنم که سمافورها یکی از چالشی ترین، جالب ترین و سرگرمی ترین بخش‌های سیستم عامل می باشد. با هدف شناساندن اصطلاحات والگوهای همگام سازی به گونه ای که به صورت مستقل قابل درک باشد و بتوان از آنها برای حل مسائل پیچیده استفاده نمود، اولین ویرایش این کتاب نوشتم. نوشتن کدهمگام سازی چالش‌های مختص به خود را دارد زیرا که با افزایش تعداد اجزا و تعداد تعاملات به طور غیر قابل کنترلی افزایش می یابد.

با این وجود در بین راه حل‌هایی که دیدم، الگوهای یافتم و حداقل برخی رهیافت‌های روشمند درست برای ترکیب راه حل‌ها رسیدم. شانس این را داشتم که در زمانی که در کالج ویلسلی بودم، این کتاب را به همراه کتاب درسی استاندارد استفاده کردم و در زمان تدریس درس مبحث همگام سازی را به شکل موازی با درس تدریس می کردم. هر هفته به دانشجویان چند صفحه از کتاب را می دادم که با یک معما تمام می شد و گاهی اوقات به راهنمایی مختصر. و به آنها توصیه می کردم که به راهنمایی نگاه نکنند مگر اینکه گیر افتاده باشند. و همچنین ابزارهایی برای تست راه حل‌ها دادم، به تخته مغناطیسی کوچک که می توانستن کدهاشون رو بنویسند و یک بسته آهنربا برای نمایش تردهای در حال اجرا.

نتیجه بسیار چشمگیر بود، هر چه زمان بیشتری در اختیار دانشجویان می گذاشتم، عمق فهمشون

بیشتر می شد، مهمتر اینکه غالبشون قادر به حل بیشتر معماها بودند، و در برخی حالات همان راه حل های کلاسیک را می یافتند و یا راه حل جدیدی را ایجاد می کردند. وقتی که رفتم کالج گام بعدی را با ایجاد کلاس فوق برنامه همگام سازی برداشتم، که در آن کلاس این کتاب تدریس می شد و همچنین پیاده سازی دستورات اولیه همگام سازی در زبان اسمبلی x86 و پاسیکس و پیتون. دانشجویانی که این درس را گرفتند در یافتن خطاهای نسخه نخست کمک کردند و چندان از آنها راه حل هایی بهتر از راه حل های من ارائه دادند در پایان ترم از هر کدام آنها خواستم که یک مسائله جدید با ترجیحا با یک راه حل بنویسند. از این مشارکت ها در نسخه دوم استفاده کردم.

بخش باقی مانده از پیشگفتار: همچنین، پس از عرضه ی ویرایش اول، کنث ریک (Kenneth Reek) مقاله ی «الگوهای طراحی سمافورها» را در «گروه ویژه ی علاقمند به آموزش علوم کامپیوتر در ACM» ارائه داد. او در این مقاله مسأله ای را که من به آن «مسئله ی سوشی بار» می گویم معرفی و دو راه حل برای اثبات الگوهایی که وی آن ها را «دست به دست کردن باتوم» و «این کار را برای تو می کنم» نامید مطرح کرد. هنگامی که با این الگوها آشنا شدم، توانستم آن ها را در مسائل ویرایش اول کتاب به کار برم و راه حل هایی تولید کنم که به نظرم بهتر هستند. تغییر دیگر در نسخه دوم، نوع نگارش یا نحو آن است. بعد از آنی که نسخه اول را نوشتم، من زبان برنامه نویسی پی تون را که تنها یکی از عالیترین زبان های برنامه نویسی است بلکه یک زبان بسیار شبیه به شبه کد است را یاد گرفتم. در نتیجه من از یک شبه کد شبیه به C در ویرایش نخست به یک شبه کد شبیه به زبان پی تون تغییر دادم. در حقیقت، من یک شبیه ساز نوشتم که بسیاری از راه حل های ارائه شده در این کتاب را می تواند اجرا کند. خواننده هایی هم که با زبان پی تون آشنا نیستند نیز (ان شاء الله) می توانند آن را درک کنند. در مواردی که از ویژگی های زبان پی تون استفاده کردم، نحو زبان پی تون و نحوی کار کد را شرح داده ام. امیدوارم این تغییر زمینه خوانا تر کردن کتاب را بوجود آورده باشد. صفحه بندی این کتاب ممکن است کمی عجیب به نظر برسد! اما صفحات خالی نیز خود یک روش سودمند است، بعد از هر معما، یک فضای خالی را تا شبه راهنمایی که در صفحه بعد است، گذاشته ام و بعد از آن یک صفحه خالی دیگر برای حل مساله تا صفحه نمایش راه حل نهایی. زمانی که من از این کتاب در کلاس استفاده می کنم، برخی از صفحات را از کتاب جدا می کنم و دانشجویانم آنها را بعدا صحافی می کنند! این سیستم صفحه بندی امکان جدا کردن معما را بدون صفحات مربوط به راهنمایی ها، محقق می کند. بعضی اوقات بخش مربوط به راهنمایی را تا می کنم و از دیده شدن آن جلوگیری می کنم تا دانشجویان خود به حل مساله پرداخته و در زمان مناسب راه حل را ببینند. اگر کتاب را به شکل تک صفحه چاپ کنید (به شکل یک رو سفید! زمانی که پولتان زیادی کرده باشد! مترجم) می توانید از چاپ صفحه های سفید خودداری کنید (ظاهرا نویسنده برای ایالت های اصفهان نشین آمریکا هستند! مترجم). این کتاب یک کتاب رایگان است، این بدین معنی است که هر شخصی می تواند آنرا بخواند، رونوشت برداری کند، اصلاح کند و حتی بازپخش کند و اینها به دلیل نوع لایسنس مورد

استفاده برای این کتاب است. امیدوارم افراد این کتاب را مناسب و کارا ببینند، اما بیشتر از آن امیدوارم که آنها برای ادامه فرایند توسعه ایرادات و پیشنهادات خود و همینطور مطالب بیشتر خود را برایم ارسال کنند. با تشکر آلن دونی

نیدهام، ماساچوست
سه شنبه، ۱۲ خرداد ۱۳۸۳

لیست همکاران

در ادامه لیست برخی افرادی که در این کتاب مشارکت داشته‌اند آمده است:

- بسیاری از مسائل این کتاب گونه دیگری از مسائل کلاسیکی است که ابتدا در مقالات تخصصی آمده‌اند و سپس در کتب مرجع. هر کجا که منبع یک مسأله یا راه حل را بدانم در متن به آن اشاره خواهم داشت.
- همچنین از دانشجویان Wellesley College که با ویرایش اول این کتاب کار کرده‌اند تشکر می‌نمایم و نیز دانشجویان Olin College که با ویرایش دوم کتاب سر و کار داشتند.
- Se Won تصحیح کوچکی —لکن مهم— را در ارائه راه حل Tanenbaum نسب به مسأله فیلسوف‌های در حال غذا خوردن ارسال نموده است.
- Daniel Zingaro در مسأله Dancer نکته‌ای را متذکر گردید که سبب بازنویسی مجدد آن بخش گردید. امیدوارم اکنون با معنی‌تر شده باشد. علاوه بر این Daniel یک خطا را در نسخه قبلی راه حل مسأله H_2O نشان داده است و سال بعد از آن نیز تعدادی خطاهای تایپی را متذکر شده است.
- Thomas Hansen یک خطای تایپی را در مسأله Cigarette smokers یافته است.
- Pascal Rütten به چندین اشکال تایپی اشاره نموده است از جمله تلفظ نادرست Edsger Dijkstra.
- Marcelo Johann خطایی را در راه حل مسأله Dining Savages یافته و آن را اصلاح کرده است.
- Roger Shipman تمام اصلاحات به علاوه یک گونه جذاب از مسأله Barrier را ارسال نموده است.
- Jon Cass یک از قلم افتادگی را در مسأله فیلسوف‌های در حال غذا خوردن مشخص نموده است.

- Krzysztof Kościuszkiewicz چندین اصلاح از جمله از قلم افتادن خطی در تعریف کلاس Fifo را فرستاده است.
- Fritz Vaandrager از دانشگاه Radboud هلند و دانشجویانش Manuel, Marc Schoolderman و Lars Lockefeer و Stampe ابزاری بنام UPPAAL را به منظور بررسی چندین راه حل این کتاب بکار برده و خطاهایی را در راه حل‌های ارائه شده برای مساله‌های Room Party و Modus Hall یافته‌اند.
- Eric Gorr درست نبودن یک توضیح در فصل سوم را مشخص نموده است.
- Jouni Leppäjärvi در واضح نمودن مبدأ سمافورها کمک نموده است.
- Christoph Bartoschek خطایی در راه حل مساله رقص انحصاری را یافته است.
- Eus یک خطای تایپی در فصل سوم را پیدا کرده است.
- Tak-Shing Chan یک خطای خارج از محدوده¹ را در counter_mutex.c یافته است.
- Roman V. Kiseliiov چند پیشنهاد برای بهبود ظاهر کتاب ارائه داده و با چند نکته در \LaTeX مرا راهنمایی نموده است.
- Alejandro Céspedes در حال کار روی ترجمه اسپانیایی این کتاب است و چندین غلط تایپی را در آن یافته است.
- Erich Nahum مشکلی را در تطبیق راه حل Kenneth Reek نسبت به مساله Sushi Bar یافته است.
- Martin Storsjö تصحیحی در مساله generalized smokers را ارسال نموده است.
- Cris Hawkins به یک متغیر بدون استفاده اشاره نموده است.
- Adolfo Di Mare یک "and" از جا افتاده را یافته است.
- Simon Ellis یک خطای تایپی را یافته است.
- Benjamin Nash یک خطای تایپی و خطایی در یک راه حل و مشکل دیگری را یافته است.
- Alejandro Pulver مشکلی را در راه حل مساله Barbershop یافته است.

¹out-of-bounds

فهرست مطالب

آ	پیشگفتار
۱	۱ معرفی
۱	۱.۱ به‌هنگام سازی
۲	۲.۱ مدل اجرایی
۵	۳.۱ تسلل به کمک پیام‌دهی
۷	۴.۱ عدم قطعیت
۷	۵.۱ متغیرهای اشتراکی
۸	۱.۵.۱ نوشتن‌های همروند
۸	۲.۵.۱ updates Concurrent
۱۰	۳.۵.۱ messages with exclusion Mutual
۱۱	۲ Semaphores
۱۱	۱.۲ Definition
۱۲	۲.۲ Syntax
۱۴	۳.۲ semaphores? Why
۱۵	۳ الگوهای همگام سازی پایه
۱۵	۱.۳ علامت‌دهی
۱۶	۲.۳ Sync.py
۱۶	۳.۳ قرار ملاقات
۱۹	۱.۳.۳ اشاره ای در خصوص قرار ملاقات

۲۱ solution Rendezvous	۲.۳.۳
۲۱ #۱ Deadlock	۳.۳.۳
۲۲ Mutex	۴.۳
۲۳ hint exclusion Mutual	۱.۴.۳
۲۵ solution exclusion Mutual	۲.۴.۳
۲۶ Multiplex	۵.۳
۲۷ solution Multiplex	۱.۵.۳
۲۸ Barrier	۶.۳
۲۹ hint Barrier	۱.۶.۳
۳۱ non-solution Barrier	۲.۶.۳
۳۳ #۲ Deadlock	۳.۶.۳
۳۵ solution Barrier	۴.۶.۳
۳۷ #۳ Deadlock	۵.۶.۳
۳۷ barrier Reusable	۷.۳
۳۹ #۱ non-solution barrier Reusable	۱.۷.۳
۴۱ #۱ problem barrier Reusable	۲.۷.۳
۴۳ #۲ non-solution barrier Reusable	۳.۷.۳
۴۵ hint barrier Reusable	۴.۷.۳
۴۷ solution barrier Reusable	۵.۷.۳
۴۹ turnstile Preloaded	۶.۷.۳
۵۱ objects Barrier	۷.۷.۳
۵۲ صف	۸.۳
۵۳ راهنمایی برای معما	۱.۸.۳
۵۵ راه حل صف	۲.۸.۳
۵۷ راهنمای صف انحصاری	۳.۸.۳
۵۹ راه حل صف انحصاری	۴.۸.۳
۶۱ problems synchronization Classical	۴
۶۱ problem Producer-consumer	۱.۴
۶۳ hint Producer-consumer	۱.۱.۴

۶۵	solution Producer-consumer	۲.۱.۴
۶۷	#۴ Deadlock	۳.۱.۴
۶۷	buffer finite a with Producer-consumer	۴.۱.۴
۶۹	hint producer-consumer buffer Finite	۵.۱.۴
۷۱	solution producer-consumer buffer Finite	۶.۱.۴
۷۱	problem Readers-writers	۲.۴
۷۳	hint Readers-writers	۱.۲.۴
۷۵	solution Readers-writers	۲.۲.۴
۷۸	Starvation	۳.۲.۴
۸۱	hint readers-writers No-starve	۴.۲.۴
۸۳	solution readers-writers No-starve	۵.۲.۴
۸۵	hint readers-writers Writer-priority	۶.۲.۴
۸۷	solution readers-writers Writer-priority	۷.۲.۴
۸۹	mutex No-starve	۳.۴
۹۳	hint mutex No-starve	۱.۳.۴
۹۵	solution mutex No-starve	۲.۳.۴
۹۹	philosophers Dining	۴.۴
۱۰۳	#۵ Deadlock	۱.۴.۴
۱۰۵	#۱ hint philosophers Dining	۲.۴.۴
۱۰۷	#۱ solution philosophers Dining	۳.۴.۴
۱۰۹	#۲ solution philosopher's Dining	۴.۴.۴
۱۱۱	solution Tanenbaum's	۵.۴.۴
۱۱۳	Tanenbaums Starving	۶.۴.۴
۱۱۵	problem smokers Cigarette	۵.۴
۱۱۹	#۶ Deadlock	۱.۵.۴
۱۲۱	hint problem Smokers	۲.۵.۴
۱۲۳	solution problem Smoker	۳.۵.۴
۱۲۴	Problem Smokers Generalized	۴.۵.۴
۱۲۵	Hint Problem Smokers Generalized	۵.۵.۴
۱۲۷	Solution Problem Smokers Generalized	۶.۵.۴

۱۲۹	problems synchronization classical Less ۵
۱۲۹	problem savages dining The ۱.۵
۱۳۱	hint Savages Dining ۱.۱.۵
۱۳۳	solution Savages Dining ۲.۱.۵
۱۳۵	problem barbershop The ۲.۵
۱۳۷	hint Barbershop ۱.۲.۵
۱۳۹	solution Barbershop ۲.۲.۵
۱۴۱	barbershop FIFO The ۳.۵
۱۴۳	hint barbershop FIFO ۱.۳.۵
۱۴۵	solution barbershop FIFO ۲.۳.۵
۱۴۷	problem Barbershop Hilzer's ۴.۵
۱۴۸	hint barbershop Hilzer's ۱.۴.۵
۱۵۱	solution barbershop Hilzer's ۲.۴.۵
۱۵۵	problem Claus Santa The ۵.۵
۱۵۷	hint problem Santa ۱.۵.۵
۱۵۹	solution problem Santa ۲.۵.۵
۱۶۱	O _r H Building ۶.۵
۱۶۳	hint O _r H ۱.۶.۵
۱۶۵	solution O _r H ۲.۶.۵
۱۶۶	problem crossing River ۷.۵
۱۶۹	hint crossing River ۱.۷.۵
۱۷۱	solution crossing River ۲.۷.۵
۱۷۳	problem coaster roller The ۸.۵
۱۷۵	hint Coaster Roller ۱.۸.۵
۱۷۷	solution Coaster Roller ۲.۸.۵
۱۷۹	problem Coaster Roller Multi-car ۳.۸.۵
۱۸۱	hint Coaster Roller Multi-car ۴.۸.۵
۱۸۳	solution Coaster Roller Multi-car ۵.۸.۵

۱۸۵	problems Not-so-classical ۶
۱۸۵	problem search-insert-delete The ۱.۶
۱۸۷	hint Search-Insert-Delete ۱.۱.۶
۱۸۹	solution Search-Insert-Delete ۲.۱.۶
۱۹۰	problem bathroom unisex The ۲.۶
۱۹۱	hint bathroom Unisex ۱.۲.۶
۱۹۳	solution bathroom Unisex ۲.۲.۶
۱۹۵	problem bathroom unisex No-starve ۳.۲.۶
۱۹۷	solution bathroom unisex No-starve ۴.۲.۶
۱۹۷	problem crossing Baboon ۳.۶
۱۹۸	Problem Hall Modus The ۴.۶
۲۰۱	hint problem Hall Modus ۱.۴.۶
۲۰۳	solution problem Hall Modus ۲.۴.۶
۲۰۷	problems classical remotely Not ۷
۲۰۷	problem bar sushi The ۱.۷
۲۰۹	hint bar Sushi ۱.۱.۷
۲۱۱	non-solution bar Sushi ۲.۱.۷
۲۱۳	non-solution bar Sushi ۳.۱.۷
۲۱۵	#۱ solution bar Sushi ۴.۱.۷
۲۱۷	#۲ solution bar Sushi ۵.۱.۷
۲۱۸	problem care child The ۲.۷
۲۱۹	hint care Child ۱.۲.۷
۲۲۱	non-solution care Child ۲.۲.۷
۲۲۳	solution care Child ۳.۲.۷
۲۲۳	problem care child Extended ۴.۲.۷
۲۲۵	hint care child Extended ۵.۲.۷
۲۲۷	solution care child Extended ۶.۲.۷
۲۲۹	problem party room The ۳.۷
۲۳۱	hint party Room ۱.۳.۷

۲۳۳	solution party Room	۲.۳.۷
۲۳۷	problem Bus Senate The	۴.۷
۲۳۹	hint problem Bus	۱.۴.۷
۲۴۱	#۱ solution problem Bus	۲.۴.۷
۲۴۳	#۲ solution problem Bus	۳.۴.۷
۲۴۵	problem Hall Faneuil The	۵.۷
۲۴۷	Hint Problem Hall Faneuil	۱.۵.۷
۲۴۹	solution problem Hall Faneuil	۲.۵.۷
۲۵۳	Hint Problem Hall Faneuil Extended	۳.۵.۷
۲۵۵	solution problem Hall Faneuil Extended	۴.۵.۷
۲۵۹	problem Hall Dining	۶.۷
۲۶۱	hint problem Hall Dining	۱.۶.۷
۲۶۳	solution problem Hall Dining	۲.۶.۷
۲۶۴	problem Hall Dining Extended	۳.۶.۷
۲۶۵	hint problem Hall Dining Extended	۴.۶.۷
۲۶۷	solution problem Hall Dining Extended	۵.۶.۷

۲۶۹ **Python in Synchronization** ۸

۲۷۰	problem checker Mutex	۱.۸
۲۷۳	hint checker Mutex	۱.۱.۸
۲۷۵	solution checker Mutex	۲.۱.۸
۲۷۷	problem machine coke The	۲.۸
۲۷۹	hint machine Coke	۱.۲.۸
۲۸۱	solution machine Coke	۲.۲.۸

۲۸۳ **C in Synchronization** ۹

۲۸۳	exclusion Mutual	۱.۹
۲۸۴	code Parent	۱.۱.۹
۲۸۴	code Child	۲.۱.۹
۲۸۵	errors Synchronization	۳.۱.۹
۲۸۷	hint exclusion Mutual	۴.۱.۹

۲۸۹	solution exclusion Mutual	۵.۱.۹
۲۹۱	semaphores own your Make	۲.۹
۲۹۳	hint implementation Semaphore	۱.۲.۹
۲۹۵	implementation Semaphore	۲.۲.۹
۲۹۷	detail implementation Semaphore	۳.۲.۹
۳۰۱		threads Python up Cleaning	آ
۳۰۱	methods Semaphore	۱.آ
۳۰۲	threads Creating	۲.آ
۳۰۳	interrupts keyboard Handling	۳.آ
۳۰۷		threads POSIX up Cleaning	ب
۳۰۷	code Pthread Compiling	۱.ب
۳۰۸	threads Creating	۲.ب
۳۱۰	threads Joining	۳.ب
۳۱۱	Semaphores	۴.ب

فصل ۱

معرفی

۱.۱ به‌هنگام‌سازی

اصطلاحاً همگام‌سازی به معنی وقوع هم‌زمان دو چیز است. در سیستم‌های کامپیوتری همگام‌سازی کلی‌تر است. این به معنی رابطه مابین رویدادهاست، در هر تعداد از رویدادها و هر نوع رابطه (قبل، حین، بعد). غالباً برنامه‌نویسان با محدودیت‌های همگام‌سازی مواجه‌اند، که این محدودیت‌ها الزاماتی در ارتباط با ترتیب این رخدادها می‌باشد.

تسلسل: رخداد الف پیش از رخداد ب اتفاق می‌افتد.

انحصار متقابل: رخداد الف و ب نباید در یک زمان رخ دهد.

در زندگی واقعی غالباً محدودیت‌های همگامی‌سازی را با کمک یک ساعت بررسی و اعمال می‌کنیم. چگونه می‌فهمیم که رخداد الف قبل از رخداد ب رخ داده است؟ با دانستن زمان رخداد هر دو واقعه را بدانیم، می‌توانیم زمان‌ها را با هم مقایسه کنیم. در سیستم‌های کامپیوتری غالباً نمی‌توانیم از ساعت در محدودیت‌های همگام‌سازی‌های کامپیوتری را برآورده کنیم، زیرا که هیچ ساعت جهانی به دلیل اینکه زمان دقیق وقوع رویدادها را نمی‌دانیم. این کتاب درباره تکنیک‌های نرم‌افزار برای اعمال‌های محدودیت‌های همگام‌سازی در کامپیوتر است.

۲.۱ مدل اجرایی

به منظور درک همگام‌سازی نرم‌افزاری، باید مدلی از چگونگی اجرای برنامه‌های کامپیوتری داشته باشید. در ساده‌ترین مدل، کامپیوترها دستورات را به ترتیب یکی پس از دیگری اجرا می‌نمایند. در این مدل، همگام‌سازی بدیهی است؛ ترتیب وقایع را با نگاه به برنامه می‌توان بیان نمود. اگر دستور A قبل از دستور B آمده باشد، اول اجرا می‌گردد.

در دو صورت همگام‌سازی پیچیده خواهد شد. ممکن است کامپیوتر موازی باشد بدین معنی که چندین پردازنده در یک زمان در حال اجرا باشد. در این حالت نمی‌توان به سادگی فهمید که دستوری در یک پردازنده قبل از دستور دیگری در پردازنده دیگر اجرا شده است.

و یا ممکن است یک پردازنده چندین نخ اجرایی داشته باشد. نخ دنباله‌ای از دستورات است که به به ترتیب اجرا می‌شوند. اگر چندین نخ وجود داشته باشد آنگاه پردازنده می‌تواند برای مدتی بر روی یکی از نخ‌ها کار کند و سپس به نخ دیگری منتقل شود و به همین ترتیب ادامه دهد.

به طور کلی برنامه‌نویس هیچ کنترلی روی اجرای نخ‌ها ندارد؛ در واقع سیستم عامل (بخصوص زمان‌بند) در این باره تصمیم می‌گیرد. در نتیجه برنامه‌نویس نمی‌تواند بگوید که دستورات چه زمانی در نخ‌های مختلف اجرا خواهد شد.

در همگام‌سازی، تفاوتی بین مدل موازی و مدل چند نخ وجود ندارد. مساله یکی است—در یک پردازنده (یا یک نخ) ترتیب اجرا مشخص است اما بین پردازنده‌ها (یا نخ‌ها) بیان این ترتیب غیر ممکن است.

یک مثال واقعی این مساله را روشن‌تر می‌نمایند. تصور کنید که شما و دوستان Bob در شهرهای متفاوتی زندگی می‌کنید. یک روز نزدیک وقت ناهار، شما به این فکر می‌افتید که چه کسی امروز زودتر ناهار خواهد خورد، شما یا Bob. چگونه این را در می‌یابید؟

به سادگی می‌توانید به او زنگ بزنید و بپرسید که چه زمانی ناهار خورده است. اما اگر شما با ساعت خودتان در ۱۱/۵۹ غذا را شروع نموده باشید و Bob با ساعت خودش در ۱۲/۰۱، آن وقت چه؟ آیا می‌توانید مطمئن باشید که چه کسی زودتر شروع نموده است؟ تنها در صورتی ممکن است که هر دوی شما نسبت به دقیق بودن ساعت‌هایتان حساس بوده باشید.

سیستم‌های کامپیوتری با مشکل مشابهی مواجه هستند زیرا با وجود اینکه معمولاً ساعت‌هایشان دقیق است اما همیشه در میزان دقت ساعت‌ها محدودیت وجود دارد. به علاوه، در بیشتر وقت‌ها کامپیوتر زمان وقوع رخدادها را دنبال نمی‌نماید. چرا که تعداد بسیار زیادی رخداد آن هم با سرعتی بسیار در حال وقوع است که ذخیره زمان دقیق همه آن‌ها ممکن نیست.

معملاً: با فرض اینکه Bob می‌خواهد دستورات ساده‌ای را دنبال نماید آیا راهی وجود دارد که تضمین

نمایید فردا شما زودتر از او ناهار خواهید خورد؟

۳.۱ تسلل به کمک پیام‌دهی

یک راه آن است که به Bob بگویید تا شما به او زنگ نزده‌اید ناهار نخورد. شما نیز اطمینان دهید پس از ناهار زنگ می‌زنید. اگر چه این راهکار بدیهی به نظر می‌رسد لکن ایده پایه آن، تبادل پیام^۱، راه حل واقعی برای بسیاری از مسائل همگام‌سازی می‌باشد. جدول زمانی زیر را در نظر بگیرید.

نخ A (شما)	نخ B (Bob)
1 Eat breakfast	1 Eat breakfast
2 Work	2 Wait for a call
3 Eat lunch	3 Eat lunch
4 Call Bob	

اولین ستون لیست اعمالی است که شما انجام می‌دهید؛ به عبارت دیگر نخ اجرای شما. ستون دوم نیز نخ اجرای Bob است. درون یک نخ همیشه می‌توانیم ترتیب اجرای وقایع را بگوییم. ترتیب وقایع را به این صورت می‌توانیم نشان دهیم

$$a_1 < a_2 < a_3 < a_4$$

$$b_1 < b_2 < b_3$$

که رابطه $a_1 < a_2$ به معنای وقوع a_1 پیش از a_2 است. ولی در کل هیچ راهی برای مقایسه رخدادهای نخ‌های مختلف نداریم؛ برای مثال ایده‌ای از اینکه چه کسی ابتدا صبحانه می‌خورد نداریم (آیا $b_1 < a_1$ است؟). اما با کمک تبادل پیام (تماس تلفنی) می‌توانیم بگوییم چه کسی زودتر ناهار خورده است ($a_3 < b_3$). با فرض اینکه باب هیچ دوست دیگری نداشته باشد هیچ تماسی جز از شما دریافت نخواهد کرد بنابراین ($b_2 > a_4$). با ترکیب تمامی روابط، داریم

$$b_3 > b_2 > a_4 > a_3$$

که ثابت می‌کند شما قبل از باب ناهار خورده‌اید. در این حالت، می‌گوییم شما و باب به صورت متوالی^۲ ناهار خورده‌اید زیرا ترتیب وقایع را می‌دانیم. از طرف دیگر صبحانه را به صورت همروند^۳ خورده‌اید زیرا که ترتیب مشخص نیست. مواقعی که درباره رخدادهای همروند صحبت می‌کنیم، اینکه بگوییم آن‌ها در یک زمان یا به صورت

¹message passing

²sequential

³concurrent

همزمان رخ می‌دهد بی‌راه نیست هر چند که دقیق هم نیست. تعبیر فوق تا زمانی که تعریف دقیق زیر را در خاطر دارید بلامانع است:

دو واقعه، همروند هستند اگر با نگاه به برنامه نتوانیم بگوییم کدامیک زودتر رخ می‌دهد.

گاهی اوقات پس از اجرای برنامه می‌توانیم بگوییم که کدامیک ابتدا رخ داده است اما غالباً ممکن نیست و حتی اگر هم بتوانیم باز هم تضمینی نیست که مرتبه بعد نتیجه‌ای یکسان بگیریم.

۴.۱ عدم قطعیت

برنامه‌های همروند اغلب **غیر قطعی**^۴ هستند به این معنی که با نگاه به برنامه امکان اینکه بگوییم با اجرای آن چه چیزی رخ خواهد داد، وجود ندارد. در ادامه یک برنامه ساده غیر قطعی آمده است:

نخ A <div style="border: 1px solid black; padding: 5px; display: inline-block;"> <code>print "yes"</code> </div>	نخ B <div style="border: 1px solid black; padding: 5px; display: inline-block;"> <code>print "no"</code> </div>
---	--

از آنجایی که دو نخ به صورت همروند اجرا می‌شوند، ترتیب اجرا بستگی به زمان‌بند دارد. در هر اجرای این برنامه، خروجی ممکن است "yes no" یا "no yes" باشد.

عدم قطعیت یکی از مواردی است که اشکال‌زدایی برنامه‌های همروند را مشکل می‌سازد. برنامه‌ای ممکن است ۱۰۰۰ بار بر روی یک سطر به درستی کار کرده و سپس در اجرای ۱۰۰۱ام بسته به تصمیمات خاص زمان‌بند با مشکل مواجه شده و اجرای برنامه متوقف شود. تقریباً پیدا کردن این نوع خطاها با بررسی کد ناممکن است؛ این نوع خطاها تنها از طریق دقت در برنامه‌نویسی قابل اجتناب هستند.

۵.۱ متغیرهای اشتراکی

بیشتر مواقع، غالب متغیرها در اکثر نخ‌ها **محلی**^۵ هستند، بدین معنی که تنها به یک نخ تعلق دارند و سایر نخ‌ها نمی‌توانند به آن‌ها دسترسی داشته باشند. تا زمانی‌که این نکته برقرار است، مشکلات همگام‌سازی کمی وجود خواهد داشت زیرا که نخ‌ها دخالتی در آن متغیرها ندارند.

اما گاهی اوقات برخی متغیرها بین دو یا چند نخ به صورت **اشتراکی**^۶ هستند؛ این یکی از شیوه‌های تعامل نخ‌ها با یکدیگر است. برای مثال، یک راه تبادل اطلاعات بین نخ‌ها، این است که نخ‌ی مقداری را بخواند و نخ دیگر آن را بنویسد.

اگر نخ‌ها ناهمگام باشند آنگاه با نگاه کردن به کد نمی‌توانیم بگوییم که آیا نخ خواننده مقداری را که نویسنده نوشته است می‌بیند یا همان مقدار قبلی را خواهد دید. لذا بسیاری از برنامه‌ها محدودیت‌هایی را بر روی خواننده‌ها اعمال می‌نمایند تا زمانی‌که نویسنده مقدار را ننوشته است چیزی را نخواند. این دقیقاً همان مساله تسلسل است که در بخش ۳.۱ آمده است. نوشتن همروند (دو یا بیشتر نویسنده) و روزرسانی همروند (دو یا بیشتر نخ که خواندنی پس از نوشتن دارند)، شیوه‌های دیگری از تعامل نخ‌ها با یکدیگر

^۴non-determinism

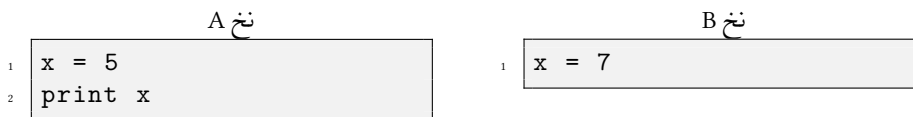
^۵local

^۶shared

است. دو بخش بعدی با این تعاملات سر و کار خواهد داشت. خواندن همروند متغیرهای اشتراکی که گونه دیگری از این تعامل است عموماً مشکل همگام‌سازی تولید نمی‌نماید.

۱.۵.۱ نوشتن‌های همروند

در این مثال، x یک متغیر اشتراکی است که دو خواننده به آن دسترسی دارند.



کدام مقدار x چاپ خواهد شد؟ در پایان اجرای تمام این دستورات، مقدار x چیست؟ این بستگی به ترتیب اجرای هر یک از دستورات، که به آن مسیر اجرا^۷ گفته می‌شود، دارد. $a_1 < a_2 < b_1$ یکی از مسیرهای ممکن است که در آن خروجی برنامه ۵ است، درحالی‌که مقدار نهایی ۷ خواهد بود.

معما: چه مسیری منجر به خروجی و مقدار نهایی ۵ می‌شود؟

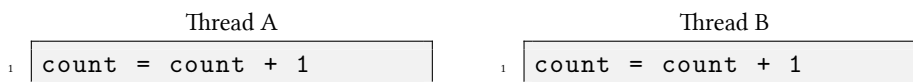
معما: چه مسیری منجر به خروجی و مقدار نهایی ۷ می‌شود؟

معما: آیا مسیری وجود دارد که منجر به خروجی ۷ و مقدار نهایی ۵ شود؟ می‌توانید جواب خود را ثابت کنید؟

پاسخ به چنین سؤالاتی یکی از بخش‌های مهم برنامه‌نویسی همروند است: مسیرهای ممکن کدام‌ها هستند و هر یک از این مسیرها چه تأثیراتی دارند؟ آیا می‌توان ثابت نمود که اثری (خواسته) ضروری است و یا اینکه اثری (ناخواسته) غیر ممکن است.

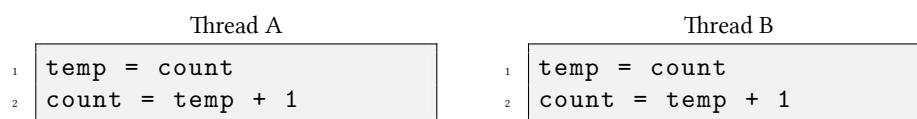
۲.۵.۱ updates Concurrent

An update operation is that reads the value of a variable, computes a new value, and writes the new value back to the variable. The most common kind of update is increment, in which the new value is the old value plus one. The following example shows a shared variable, count, being updated concurrently by two threads.



^۷execution path

here. problem synchronization a is there that obvious not is it glance. first At
 result. same the yield they and paths. execution two only are There
 before language machine into translated are operations these that is problem The
 write. a and read a steps. two takes update the language machine in and execution.
 variable. temporary a with code the rewrite we if obvious more is problem The
 .temp



path execution following the consider Now

$$a\backslash < b\backslash < b\text{?} < a\text{?}$$

both Because value? final its is what ,0 is x of value initial the that Assuming
 only is variable The value. same the write they value. initial same the read threads
 mind. in had programmer the what not probably is which once. incremented
 at looking tell. to possible always not is it because subtle is problem of kind This
 which and step single a in performed are operations which program. high-level a
 that instruction increment an provide computers some fact. In interrupted. be can
 cannot that operation An interrupted. be cannot and hardware in implemented is
 .atomic be to said is interrupted be

operations which know don't we if programs concurrent write we can how So
 operation each about information specific collect to is possibility One atomic? are
 obvious. are approach this of drawbacks The platform. hardware each on
 all that assumption conservative the make to is alternative common most The
 con- to constraints synchronization use to and atomic. not are writes all and updates
 variables. shared to access concurrent trol

men- I which mutex. or exclusion. mutual is constraint common most The
 accesses thread one only that guarantees exclusion Mutual .۱.۱ Section in tioned
 this in errors synchronization of kinds the eliminating time. a at variable shared a
 section.

(if concurrently program following the run threads ۱۰۰ that Suppose Puzzle:

times.): ۱۰۰ update the runs loop for the Python. with familiar not are you

```

1 for i in range(100):
2     temp = count
3     count = temp + 1

```

completed? have threads all after count of value possible largest the is What

value? possible smallest the is What

not. is second the easy? is question first the Hint:

messages with exclusion Mutual ۳.۵.۱

For passing. message using implemented be can exclusion mutual serialization. Like from monitor you that reactor nuclear a operate Bob and you that imagine example. but lights. warning for watching are you of both time. the of Most stations. remote lunch eats who matter doesn't It lunch. for break a take to allowed both are you the leaving time. same the at lunch eat don't you that important very is it but first.

unwatched! reactor

these enforces that calls) (phone passing message of system a out Figure Puzzle:

start will lunch when predict cannot you and clocks. no are there Assume restraints.

required? is that messages of number minimum the is What last. will it long how or

فصل ٢

Semaphores

usually visually, communicate to used signals of system a is semaphore a life real In
data a is semaphore a software. In mechanism, other some or lights, flags, with
problems, synchronization of variety a solving for useful is that structure
computer eccentric famously a Dijkstra, Edsger by invented were Semaphores
basic the but design, original the since changed have details the of Some scientist.
same, the is idea

Definition ١.٢

differences: three with integer, an like is semaphore A

integer, any to value its initialize can you semaphore, the create you When .١
increment are perform to allowed are you operations only the that after but
the read cannot You one). by (decrease decrement and one) by (increase
semaphore, the of value current

the negative, is result the if semaphore, the decrements thread a When .٢
the increments thread another until continue cannot and itself blocks thread
semaphore.

waiting, threads other are there if semaphore, the increments thread a When .3
 unblocked, gets threads waiting the of one
 the notifies it that say to is “blocks”) simply (or itself blocks thread a that say To
 run– from thread the prevent will scheduler The proceed, cannot it that scheduler
 tra– the In unblocked, become to thread the causes that occurs event an until ning
 “wak– called often is unblocking science, computer in metaphors mixed of dition
 ing”.

def– the of consequences some are there but definition, the to is there all That’s
 about, think to want might you inition

semaphore a decrements thread a before know to way no is there general. In •
 that prove to able be might you cases specific (in not or block will it whether
 not), will or will it

up, woken gets thread another and semaphore a increments thread a After •
 which know to way no is There concurrently, running continue threads both
 immediately, continue will either, if thread,

another whether know necessarily don’t you semaphore, a signal you When •
 one, or zero be may threads unblocked of number the so waiting, is thread

means, semaphore the of value the what about think to want might you Finally,
 decrement can that threads of number the represents it then positive, is value the If
 that threads of number the represents it then negative, is it If blocking, without
 threads no are there means it zero, is value the If waiting, are and blocked have
 block, will it decrement, to tries thread a if but waiting,

Syntax 2.2

available is semaphores of implementation an environments, programming most In
 implemen– Different system, operating the or language programming the of part as
 different require usually and capabilities, different slightly offer sometimes tations
 syntax.

how demonstrate to pseudo-language simple a use will I book this In
is it initializing and semaphore new a creating for syntax The work. semaphores

Semaphore initialization syntax

```
1 fred = Semaphore(1)
```

new a returns and creates it constructor: a is Semaphore function The
con- the to parameter a as passed is semaphore the of value initial The Semaphore.
structor.

environments. different in names different by go operations semaphore The
are alternatives common most The

Semaphore operations

```
1 fred.increment()
2 fred.decrement()
```

and

Semaphore operations

```
1 fred.signal()
2 fred.wait()
```

and

Semaphore operations

```
1 fred.V()
2 fred.P()
```

reason. a is there but names. many so are there that surprising be may It
de- wait and signal .do operations the what describe decrement and increment
proposed names original the were P and V And .for used often are they what scribe
misleading a than better is name meaningless a that realized wisely who Dijkstra. by
.\name

neglect decrement and increment because misleading pairs other the consider I
used often are semaphores and waking. and blocking of possibility the mention to
.wait and signal with do to nothing have that ways in
these: suggest would I then names. meaningful on insist you If

meaningless. completely aren't P and V Dutch. speak you If¹

Semaphore operations

```

1 fred.increment_and_wake_a_waiting_process_if_any()
2 fred.decrement_and_block_if_the_result_is_negative()

```

the In soon, names these of either embrace to likely is world the think don't I
 .wait and signal use to arbitrarily less or (more choose I meantime.

semaphores? Why ۳.۲

use- are they why obvious all at not is it semaphores, of definition the at Looking
 but problems, synchronization solve to semaphores *need* don't we that true It's ful.
 them: using to advantages some are there

er- avoid programmers help that constraints deliberate impose Semaphores •
 rors.

to easy it making organized, and clean often are semaphores using Solutions •
 correctness, their demonstrate

solutions so systems, many on efficiently implemented be can Semaphores •
 efficient, usually and portable are semaphores use that

فصل ۳

الگوهای همگام سازی پایه

در این فصل تعدادی از مسائل همگام سازی پایه ارائه شده است و نشان داده می شود چگونه با استفاده از سمافورها آن ها را حل کنیم. این مسائل شامل موضوعات مختلفی می شود از جمله تسلسل و انحصار متقابل — که قبلاً با آن ها آشنا شده ایم —.

۱.۳ علامت دهی

ساده ترین شکل استفاده از یک سمافور احتمالاً مکانیزم **علامت دهی**^۱ است، و به این معناست که یک نخپیمای به نخ دیگر می فرستد تا وقوع رخدادی را اعلام کند. علامت دهی این امکان را فراهم می آورد تا مطمئن شویم که یک قطعه کد از یک نخ، حتماً قبل از قطعه کدی دیگر در نخ دیگری اجرا خواهد شد؛ به عبارت دیگر، مسئله تسلسل را حل می کند. فرض کنید یک سمافور با نام `sem` و مقدار اولیه ۰ داریم، و نخ های `A` و `B` هر دو به آن دسترسی دارند.

Thread A		Thread B	
1	<code>statement a1</code>	1	<code>sem.wait()</code>
2	<code>sem.signal()</code>	2	<code>statement b1</code>

کلمه `statement` نشان دهنده یک عبارت دلخواه در برنامه است. برای اینکه مثال مشخص تر شود، فرض کنید `a1` یک خط از یک فایل را می خواند، و `b1` آن خط را در صفحه نمایش نشان می دهد. سمافور در این برنامه تضمین می کند که نخ `A` عملیات `a1` را، قبل از آنکه نخ `B` عملیات `b1` را شروع کند، به طور

¹signaling

کامل انجام داده است.

روش کار به این صورت است: اگر اول نخ B به عبارت wait برسد، با مقدار اولیه، یعنی صفر، مواجه می شود و بلاک [مسدود] خواهد شد. سپس هر زمان نخ A علامت دهد، نخ B ادامه خواهد داد. به طور مشابه، اگر اول نخ A به عبارت signal برسد، مقدار سمافور افزایش می یابد، و هنگامی که نخ B به wait برسد، بدون وقفه ادامه می یابد. بهرحال ترتیب a1 و b1 تضمین می شود. این شیوه استفاده از سمافورها، پایه و اساس نام های signal و wait است، و در این مورد، این اسامی به راحتی به خاطر سپرده می شوند. اما متأسفانه، موارد دیگری را خواهیم دید که این اسم ها کمتر به ما کمک می کنند.

حالا که از اسامی بامعنی صحبت می کنیم، باید بدانیم که sem دارای این شرایط نیست. اگر امکان داشته باشد، ایده ی خوب این است که به یک سمافور نامی دهیم که مشخص کند به چه دلالت دارد. در این مثال نام a1done می تواند خوب باشد، چرا که a1done.signal() به این معنی است که «علامت بده a1 انجام شده است»، و a1done.wait() به این معنی است که «صبر کن تا اینکه a1 انجام شود».

۲.۳ Sync.py

تمرین: درباره ی استفاده از sync بنویسید، از signal.py شروع کنید.

چرا نخ B به initComplete علامت می دهد؟

۳.۳ قرار ملاقات

معمّاً: الگوی علامت دهی را طوری تعمیم دهید که بتواند در دو جهت کار کند. نخ A باید منتظر نخ B بماند و بالعکس. به عبارت دیگر اگر کد زیر را داشته باشیم

Thread A		Thread B	
1	statement a1	1	statement b1
2	statement a2	2	statement b2

می خواهیم مطمئن شویم که a1 پیش از b2 رخ می دهد و نیز b1 قبل از a2 اتفاق می افتد. هنگام نوشتن راه حل خود، نام و مقدار اولیه سمافورها را حتماً مشخص نمایید (اشاره کوچکی وجود دارد). راه حل شما نباید قید و بندهای زیادی داشته باشد. مثلاً، ترتیب a1 و b1 برای ما اهمیتی ندارد. در راه حل شما، باید امکان هر ترتیبی وجود داشته باشد.

نام این مسئله همگام سازی، قرار ملاقات است. ایده آن به این صورت است که دو نخ در یک نقطه از اجرا با یکدیگر قرار ملاقات می‌گذارند، و تا زمانی که هر دو نرسیده باشند، دیگری حق ادامه ندارد.

۱.۳.۳ اشاره ای در خصوص قرار ملاقات

اگر خوش شانس باشید می‌توانید به یک راه حل برسید، ولی اگر هم نرسیدید، این اشاره برای شماست. دو سمافور به نام‌های aArrived و bArrived ایجاد کنید، و به هر دو مقدار اولیه صفر بدهید. همان طور که از نام‌ها مشخص است، aArrived نشان می‌دهد که آیا نخ A به محل ملاقات رسیده است، و به همین صورت bArrived نیز در مورد نخ B می‌باشد.

solution Rendezvous ٢.٣.٣

hint: previous the on based solution, my is Here

Thread A	Thread B
<pre> 1 statement a1 2 aArrived.signal() 3 bArrived.wait() 4 statement a2 </pre>	<pre> 1 statement b1 2 bArrived.signal() 3 aArrived.wait() 4 statement b2 </pre>

like something tried have might you problem, previous the on working While

this:

Thread A	Thread B
<pre> 1 statement a1 2 bArrived.wait() 3 aArrived.signal() 4 statement a2 </pre>	<pre> 1 statement b1 2 bArrived.signal() 3 aArrived.wait() 4 statement b2 </pre>

might it since efficient, less probably is it although works, also solution This

necessary, than more time one B and A between switch to have

proceed might and A wakes it arrives, B When B, for waits it first, arrives A If

,signal its reach to A allowing blocks, it case which in wait its to immediately

proceed, can threads both which after

yourself convince and code this through paths possible other the about Think

arrived, have both until proceed can thread neither cases all in that

#١ Deadlock ٣.٣.٣

something tried have might you problem, previous the on working while Again,

this: like

Thread A	Thread B
<pre> 1 statement a1 2 bArrived.wait() 3 aArrived.signal() 4 statement a2 </pre>	<pre> 1 statement b1 2 aArrived.wait() 3 bArrived.signal() 4 statement b2 </pre>

Assuming problem, serious a has it because quickly, it rejected you hope I so. If

since block, also will it arrives. B When .wait its at block will it first, arrives A that and proceed, can thread neither point. this At .aArrived signal to able wasn't A will. never solution successful a not is it obviously, and, **deadlock** a called is situation This the often but obvious, is error the case, this In problem, synchronization the of later, examples more see will We subtle, more is deadlock of possibility

Mutex ۴.۳

have We exclusion, mutual enforce to is semaphores for use common second A shared to access concurrent controlling exclusion, mutual for use one seen already variable shared the accesses thread one only that guarantees mutex The variables. time, a at one allowing another, to thread one from passes that token a like is mutex A of group a *Flies the of Lord The* in example, For proceed, to time a at thread As conch, the hold to have you speak, to order In mutex, a as conch a use children .^۴ speak can one only conch, the holds child one only as long the “get” to has it variable, shared a access to thread a for order in Similarly, the hold can thread one Only mutex, the “releases” it done, is it when mutex: time, a at mutex exclusion mutual enforce to example following the to semaphores Add Puzzle:

.count variable shared the to

	Thread A		Thread B
1	<code>count = count + 1</code>	1	<code>count = count + 1</code>

^{۴.۵} Section in see will you as misleading, be also can it now, for helpful is metaphor this Although^۴

hint exclusion Mutual ۱.۴.۳

that means one of value A .\ to initialized is that mutex named semaphore a Create
it that means zero of value a variable: shared the access and proceed may thread a
mutex. the release to thread another for wait to has

solution exclusion Mutual ٢.٢.٣

solution: a is Here

Thread A		Thread B	
1	<code>mutex.wait()</code>	1	<code>mutex.wait()</code>
2	<code># critical section</code>	2	<code># critical section</code>
3	<code>count = count + 1</code>	3	<code>count = count + 1</code>
4	<code>mutex.signal()</code>	4	<code>mutex.signal()</code>

to able be will first wait the to gets thread whichever \ initially is mutex Since effect the has semaphore the on waiting of act the course. Of immediately. proceed first the until wait to have will arrive to thread second the so it. decrementing of signals.

the within contained is it that show to operation update the indented have I mutex.

sometimes is This code. same the running are threads both example. this In solution the code. different run to have threads the If solution. **symmetric** a called the case. this In generalize. to easier often are solutions Symmetric. **asymmetric** is modifica- without threads concurrent of number any handle can solution mutex after. signals and update an performing before waits thread every as long As tion.

concurrently. count access will threads two no then sup- I **section critical** the called is protected be to needs that code the Often

access. concurrent prevent to important critically is it because pose several are there metaphors. mixed and science computer of tradition the In been have we metaphor the In mutexes. about talk sometimes people ways other another. to thread one from passed is that token a is mutex the far. so using

only and room. a as section critical the of think we metaphor. alternative an In are mutexes metaphor. this In time. a at room the in be to allowed is thread one it unlock and entering before mutex the lock to said is thread a and locks. called about talk and metaphors the mix people though. Occasionally. exiting. while sense. much make doesn't which lock. a "releasing" or "getting" work you As misleading. potentially and useful potentially are metaphors Both

to you leads one which see and thinking of ways both out try problem. next the on
solution. a

Multiplex ۵.۳

in run to threads multiple allows it that so solution previous the Generalize Puzzle:
of number the on limit upper an enforces it but time. same the at section critical the
critical the in run can threads n than more no words. other In threads. concurrent
time. same the at section
at occurs problem multiplex the life. real In **multiplex** a called is pattern This
build- the in allowed people of number maximum a is there where nightclubs busy
exclusivity. of illusion the create to or safety fire maintain to either time. a at ing
by constraint synchronization the enforces usually bouncer a places such At
is room the when arrivals barring and inside people of number the of track keeping
enter. to allowed is another leaves person one whenever Then. capacity. at
almost is it but difficult. sound may semaphores with constraint this Enforcing
trivial.

solution Multiplex ١.٥.٣

semaphore the initialize just section. critical the in run to threads multiple allow To
 allowed. be should that threads of number maximum the is which .n to
 additional of number the represents semaphore the of value the time. any At
 until block will thread next the then zero. is value the If enter. may that threads
 value the exited have threads all When signals. and exits inside threads the of one
 .n to restored is semaphore the of
 the of copy one only show to conventional it's symmetric. is solution the Since
 in concurrently running code the of copies multiple imagine should you but code.
 threads. multiple

Multiplex solution

```

1 multiplex.wait()
2   critical section
3 multiplex.signal()

```

ar- thread one than more and occupied is section critical the if happens What
 does solution This wait. to arrivals the all for is want we what course. Of rives?
 decremented. is semaphore the queue. the joins arrival an time Each that. exactly
 in threads of number the represents (negated) semaphore the of value the that so
 queue.

al- and value its incrementing semaphore. the signals it leaves. thread a When
 proceed. to threads waiting the of one lowing
 the of think to useful it find I case this in metaphors. of again Thinking
 .wait invokes thread each As lock). a than (rather tokens of set a as semaphore
 thread a Only one. releases it signal invokes it when tokens: the of one up picks it
 thread a when available are tokens no If room. the enter can token a holds that
 one. releases thread another until waits it arrives.

out hand They this. like system a use sometimes windows ticket life. real In
 holder the allows token Each line. in customers to chips) poker (sometimes tokens
 ticket. a buy to

Barrier ۶.۳

Consider the again problem Rendezvous from Section ۳.۳. A limitation of the solution we presented is that it does not work with more than two threads. Generalize the solution. Every thread should be able to follow the run of the puzzle.

lowing code:

Barrier code

```
1 rendezvous
2 critical point
```

The synchronization requirement is that no thread executes `critical point` until all threads have executed `rendezvous`. You can assume that there are n threads and that this value is stored in a variable. When the first of $n - 1$ threads arrives, the block until should they arrive threads $n - 1$ first the When proceed. may threads the all point which at

hint Barrier ۱.۶.۳

vari- the presenting by hints provide will I book this in problems the of many For
roles. their explaining and solution my in used I ables

Barrier hint

```

1 n = the number of threads
2 count = 0
3 mutex = Semaphore(1)
4 barrier = Semaphore(0)

```

exclusive provides mutex arrived. have threads many how of track keeps count
safely. it increment can threads that so count to access
be should it then arrive: threads all until negative) or (zero locked is barrier
more). or \) unlocked

non-solution Barrier 9.3

examine to useful is it because right, quite not is that solution a present will I First
wrong, is what out figure and solutions incorrect

Barrier non-solution

```

1 rendezvous
2
3 mutex.wait()
4     count = count + 1
5 mutex.signal()
6
7 if count == n: barrier.signal()
8
9 barrier.wait()
10
11 critical point

```

pass, that threads of number the counts it mutex, a by protected is count Since
locked, initially is which barrier, the to get they when wait threads $n - 1$ first The
barrier, the unlocks it arrives, thread th_n the When
solution? this with wrong is What Puzzle:

#۲ Deadlock ۳.۶.۳

deadlock. a is problem The
 barrier. the at waiting are threads \forall that and $n = \Delta$ that imagine example. an An
 is which negated. queue. in threads of number the is semaphore the of value The
 .۴-
 allowed is threads waiting the of one barrier. the signals thread Δ th the When
 .۳- to incremented is semaphore the and proceed. to
 can threads other the of none and again semaphore the signals one no then But
 deadlock. a of example second a is This barrier. the pass
 execution an find you Can deadlock? a create always code this Does Puzzle:
 deadlock? a cause *not* does that code this through path
 problem. the Fix Puzzle:

solution Barrier ۶.۶.۳

barrier: working a is here Finally.

Barrier solution

```

1 rendezvous
2
3 mutex.wait()
4     count = count + 1
5 mutex.signal()
6
7 if count == n: barrier.signal()
8
9 barrier.wait()
10 barrier.signal()
11
12 critical point

```

each as Now barrier. the at waiting after signal another is change only The
 pass. can thread next the that so semaphore the signals it passes. thread
 that enough often occurs succession. rapid in signal a and wait a pattern. This
 time. a at pass to thread one allows it because **turnstile** a called it's name. a has it
 threads. all bar to locked be can it and
 and it unlocks thread thn The locked. is turnstile the (zero). state initial its In
 through. go threads n all then
 this In mutex. the outside count of value the read to dangerous seem might It
 will We idea. good a not probably is it general in but problem. a not is it case
 these consider to want might you meantime. the in but pages. few a in up this clean
 the way any there Is in? turnstile the is state what thread. thn the After questions:
 once? than more signaled be might barrier

#3 Deadlock 0.6.3

thread one only and mutex, the through pass can time a at thread one only Since
turnstile the put to reasonable seen might it turnstile, the through pass can time a at
this: like mutex, the inside

Bad barrier solution

```

1 rendezvous
2
3 mutex.wait()
4     count = count + 1
5     if count == n: barrier.signal()
6
7     barrier.wait()
8     barrier.signal()
9 mutex.signal()
10
11 critical point

```

deadlock, a cause can it because idea bad a be to out turns This
reaches it when blocks then and mutex the enters thread first the that Imagine
condi- the so enter, can threads other no locked, is mutex the Since turnstile, the
turnstile, the unlock ever will one no and true be never will ,count==n tion,
source common a demonstrates it but obvious, fairly is deadlock the case this In
mutex, a holding while semaphore a on blocking deadlocks: of

barrier Reusable V.3

syn- and loop a in steps of series a perform will threads cooperating of set a Often
barrier reusable a need we application this For step, each after barrier a at chronize
through, passed have threads the all after itself locks that
passed have threads the all after that so solution barrier the Rewrite Puzzle:
again, locked is turnstile the through,

#\ non-solution barrier Reusable ۱.V.۳

improve gradually and solution a at attempt simple a with start will we again. Once it:

Reusable barrier non-solution

```

1 rendezvous
2
3 mutex.wait()
4     count += 1
5 mutex.signal()
6
7 if count == n: turnstile.signal()
8
9 turnstile.wait()
10 turnstile.signal()
11
12 critical point
13
14 mutex.wait()
15     count -= 1
16 mutex.signal()
17
18 if count == 0: turnstile.wait()

```

before code the as same the much pretty is turnstile the after code the that Notice
 .count variable shared the to access protect to mutex the use to have we Again. it.
 correct. quite not is code this though. Tragically,
 problem? the is What Puzzle:

#\ problem barrier Reusable ٧.٧.٣

code. previous the of \forall Line at spot problem a is There comes thread th_n the then and point. this at interrupted is thread $th_n - 1$ the If sig- will threads both and $count == n$ that find will threads both mutex. the through turnstile. the signal will threads the *all* that possible even is it fact. In turnstile. the nal cause will which wait to threads multiple for possible is it $\forall \wedge$ Line at Similarly.

deadlock. a

problem. the Fix Puzzle:

#१ non-solution barrier Reusable ३.V.३

remains. problem subtle a but error. previous the fixes attempt This

Reusable barrier non-solution

```

1 rendezvous
2
3 mutex.wait()
4     count += 1
5     if count == n: turnstile.signal()
6 mutex.signal()
7
8 turnstile.wait()
9 turnstile.signal()
10
11 critical point
12
13 mutex.wait()
14     count -= 1
15     if count == 0: turnstile.wait()
16 mutex.signal()

```

interrupted be cannot thread a that so mutex the inside is check the cases both In

it. checking before and counter the changing after

inside be will barrier this that Remember correct. not *still* is code this Tragically,

rendezvous. the to back go will thread each line. last the executing after So, loop. a

problem. the fix and Identify Puzzle:

hint barrier Reusable ५.V.३

the through pass to thread precocious a allows code this written. currently is it As
 turnstile. the and mutex first the through pass and around loop then mutex. second
 lap. a by threads other the of ahead getting effectively
 turnstiles. two use can we problem this solve To

Reusable barrier hint

```

1 turnstile = Semaphore(0)
2 turnstile2 = Semaphore(1)
3 mutex = Semaphore(1)

```

arrive threads the all When open. is second the and locked is first the Initially
 at arrive threads the all When first. the unlock and second the lock we first. the at
 around loop to threads the for safe it makes which first. the relock we second the
 second. the open then and beginning. the to

solution barrier Reusable ०.V.३

Reusable barrier solution

```

1 # rendezvous
2
3 mutex.wait()
4     count += 1
5     if count == n:
6         turnstile2.wait()      # lock the second
7         turnstile.signal()     # unlock the first
8 mutex.signal()
9
10 turnstile.wait()              # first turnstile
11 turnstile.signal()
12
13 # critical point
14
15 mutex.wait()
16     count -= 1
17     if count == 0:
18         turnstile.wait()      # lock the first
19         turnstile2.signal()   # unlock the second
20 mutex.signal()
21
22 turnstile2.wait()              # second turnstile
23 turnstile2.signal()

```

the all forces it because **barrier two-phase** a called sometimes is solution This threads the all for again and arrive to threads the all for once twice: wait to threads section. critical the execute to code: synchronization non-trivial most of typical is solution this Unfortunately, a that way subtle a is there Often correct. is solution a that sure be to difficult is it error. an cause can program the through path particular much not is solution a of implementation an testing worse. matters make To it causes that path particular the because rarely very occur might error The help. errors Such circumstances. of combination unlucky spectacularly a require might means. conventional by debug and reproduce to impossible almost are cor- is it that “prove” and carefully code the examine to is alternative only The you that necessarily. mean. don’t I because marks quotation in “prove” put I rect.

lu- such encourage who zealots are there (although proof formal a write to have nacy).

of advantage take can We informal. more is mind in have I proof of kind The then and assert. to developed. have we idioms the and code. the of structure the ex- For program. the about claims intermediate-level of number a demonstrate. ample:

turnstiles. the unlock or lock can thread thn the Only .۱

and second. the close to has it turnstile. first the unlock can thread a Before .۲ others the of ahead get to thread one for impossible is it therefore versa: vice turnstile. one than more by

sometimes can you prove. and assert to statements of kinds right the finding By is code your that colleague) skeptical a (or yourself convince to way concise a find bulletproof.

turnstile Preloaded ٩.V.٣

in use can you component versatile a is it that is turnstile a about thing nice One through go to threads forces it that is drawback one But solutions. of variety a necessary. than switching context more cause may which sequentially. that thread the if solution the simplify can we solution. barrier reusable the In num- right the let to signals enough with turnstile the preloads turnstile the unlocks . through threads of ber spec- that parameter a take can signal that assumes here using am I syntax The to easy be would it but feature. non-standard a is This signals. of number the ifies signals multiple the that is mind in keep to thing only The loop. a with implement But loop. the in interrupted be might thread signaling the is. that atomic. not are problem. a not is that case this in

Reusable barrier solution

```

1 # rendezvous
2
3 mutex.wait()
4     count += 1
5     if count == n:
6         turnstile.signal(n)      # unlock the first
7 mutex.signal()
8
9 turnstile.wait()                # first turnstile
10
11 # critical point
12
13 mutex.wait()
14     count -= 1
15     if count == 0:
16         turnstile2.signal(n)    # unlock the second
17 mutex.signal()
18
19 turnstile2.wait()              # second turnstile

```

for signal one with turnstile first the preloads it arrives. thread tn the When and token” last the “takes it turnstile. the passes thread tn the When thread. each

solution! this for Tesch Matt to Thanks”

again. locked turnstile the leaves
the when unlocked is which turnstile. second the at happens thing same The
mutex. the through goes thread last

objects Barrier V.V.3

syntax Python the borrow will I object. an in barrier a encapsulate to natural is It
class: a defining for

Barrier class

```

1 class Barrier:
2     def __init__(self, n):
3         self.n = n
4         self.count = 0
5         self.mutex = Semaphore(1)
6         self.turnstile = Semaphore(0)
7         self.turnstile2 = Semaphore(0)
8
9     def phase1(self):
10        self.mutex.wait()
11        self.count += 1
12        if self.count == self.n:
13            self.turnstile.signal(self.n)
14        self.mutex.signal()
15        self.turnstile.wait()
16
17    def phase2(self):
18        self.mutex.wait()
19        self.count -= 1
20        if self.count == 0:
21            self.turnstile2.signal(self.n)
22        self.mutex.signal()
23        self.turnstile2.wait()
24
25    def wait(self):
26        self.phase1()
27        self.phase2()

```

ini- and object. Barrier new a create we when runs method `__init__` The
have that threads of number the is `n` parameter The variables. instance the tializes
opens. Barrier the before `wait` invoke to
each Since on. operating is method the object the to refers `self` variable The
specific the to refers `self.mutex` turnstiles. and mutex own its has object barrier
object. current the of mutex
it: on waits and object Barrier a creates that example an is Here

Barrier interface

```

1 barrier = Barrier(n)           # initialize a new barrier
2 barrier.wait()                 # wait at a barrier

```

if separately, phase2 and phase1 call can barrier a uses that code Optionally,

between. in done be should that else something is there

۸.۳ صف

سمافورها می توانند به عنوان یک صف نیز استفاده شوند. در MASHKOOK این مورد مقدار اولیه • و کد مربوطه از قبل نوشته شده است، بنابراین سیگنال دادن به نخ دیگر عملاً غیر ممکن است مگر اینکه یک ترد دیگر در حال انتظار باشد در نتیجه مقدار سمافور هرگز مثبت نخواهد شد. به عنوان مثال، تصور کنید که نخ‌ها نمایش دهنده اتاق رقص باشند و دو نوع رقص، به شکل جلودار و دنباله‌رو باشد، که قبل از ورود به صحن در دو صف مجزا قرار خواهند گرفت. زمانی که یک جلودار می‌رسد، چک می‌کند که آیا هیچ دنباله‌روای در حالت انتظار است یا خیر. اگر چنین باشد، هر دو می‌توانند عمل کنند (به رقصند!) در غیر اینصورت می‌بایست منتظر باشند. به شکل مشابهی، زمانی که یک دنباله‌رو می‌رسد، او ابتدا چک می‌کند که آیا هیچ جلوداری فعال یا منتظر شده است یا خیر، او نیز چنین خواهد کرد. معما: یک کدی بنویسید برای جلودارها و دنباله‌روها که این شرایط در آن صدق کند.

۱.۸.۳ راهنمایی برای معما

اینجا یک متغیر I هست که در راه حل ما استفاده شده است:

Queue hint

```
1 leaderQueue = Semaphore(0)
2 followerQueue = Semaphore(0)
```

leaderQueue صف حاوی جلودارهای منتظر هست! و followerQueue صف حاوی دنباله‌روهای منتظر هست!

۲.۸.۳ راه حل صف

کد مربوط به جلودارها را در اینجا مشاهده می‌کند:

Queue solution (leaders)

```
1 followerQueue.signal()
2 leaderQueue.wait()
3 dance()
```

و در این قسمت کد مربوط به دنباله‌روها گذاشته شده است:

Queue solution (followers)

```
1 leaderQueue.signal()
2 followerQueue.wait()
3 dance()
```

این راه حل ساده تر از آن است که آن را بگیرید! صرفاً یک کلید است. هر سیگنال از جلودار دقیقاً یک دنباله‌رو دارد و هر سیگنال دنباله‌رو یک جلودار دارد، لذا این ضمانت می‌کند که جلودار و دنباله‌رو هر دو به شکل همزمان وارد عملیات می‌شوند (می‌رقصند!!!). اگرچه آنها هر دو وارد عملیات شده اند اما این ممکن است که نخ‌ها قبل از اجرای dance انباشته شوند، و در نتیجه امکان دارد که تعدادی از جلودارهای dance قبل از دنباله‌روها وارد عمل شوند. بر اساس مفهوم dance که ممکن است گیج کننده باشد! برای اینکه جالب تر باشد، فرض کنید یک شرط اضافه را به هر کدام از جلودارها که می‌توانند dance را به شکل همزمان صدا بزنند با تنها یک دنباله‌رو و برعکس، اضافه کنیم. به معنی دیگر شما می‌توانید برقصید با کسی که شما را آورده است^۴. معماً: یک راه حل برای این مساله صف انحصاری ارائه کنید.

^۴ترانه توسط شاینا تیواین

۳.۸.۳ راهنمای صف انحصاری

متغیرهای راهنمایی را در زیر می بیند:

Queue hint

```
1 leaders = followers = 0
2 mutex = Semaphore(1)
3 leaderQueue = Semaphore(0)
4 followerQueue = Semaphore(0)
5 rendezvous = Semaphore(0)
```

leaders و followers شمارنده هایی هستند که تعداد رقاص ها را از هر نوع که در حال انتظار هستند، نگه می دارند. موتکس دسترسی انحصاری به شمارنده ها را ضمانت می کند. leaderQueue و followerQueue صف هایی هستند که رقاص های منتظر شده را نگه داری می کند. rendezvous نیز برای چک کردن اینکه هر دو نخ در حال رقصیدن هستند می باشد.

۴.۸.۳ راه حل صف انحصاری

قطعه کد مربوط به رقاص جلودار:

Queue solution (leaders)

```

1 mutex.wait()
2 if followers > 0:
3     followers--
4     followerQueue.signal()
5 else:
6     leaders++
7     mutex.signal()
8     leaderQueue.wait()
9
10 dance()
11 rendezvous.wait()
12 mutex.signal()

```

زمانی که یک جلودار می‌رسد، یک میوتکس می‌گیرد تا leaders و followers محافظت شود. اگر یک دنباله‌رو منتظر باشد، جلودار مقدار followers را کاهش می‌دهد، به یک جلودار سیگنال می‌دهد و سپس dance را فراخوانی می‌کند، تمامی قبلی‌ها mutex را آزاد می‌کنند. این ضمانت می‌کند که تنها یک نخ دنباله‌رو را dance به شکل همزمان اجرا می‌کند. اگر هیچ دنباله‌روی در حالت انتظار نباشد، جلودار میوتکس را قبل از تغییر به حالت انتظار، بر روی leaderQueue فعال می‌کند. کدمربوط به دنباله‌رو به این شکل است:

Queue solution (followers)

```

1 mutex.wait()
2 if leaders > 0:
3     leaders--
4     leaderQueue.signal()
5 else:
6     followers++
7     mutex.signal()
8     followerQueue.wait()
9
10 dance()
11 rendezvous.signal()

```

زمانی که یک دنباله‌رو می‌رسد، آن چک می‌کند که جلودار در حالت انتظار هست. اگر یکی دیگر وجود داشته باشد، دنباله‌رو مقدار leaders را کاهش می‌دهد، به جلودار سیگنال می‌دهد، و dance را اجرا می‌کند، و mutex بقیه را آزاد می‌کند. در حقیقت در این مورد دنباله‌رو هرگز mutex را رها نمی‌کند.

جلودار اینکار را می‌کرد. ما نمی‌توانیم از چنین نخ موتکسی، پرهیز کنیم چرا که ما می‌دانیم که یکی از آنها می‌توانند وارد عمل شوند و هر یک از آن دو می‌توانند رها کنند. البته در این راه حل آن همیشه جلودار است! زمانی که یک سمافور به عنوان یک صف استفاده می‌شود^۵، من آنرا سودمند یافتیم برای بررسی وضعیت “انتظار” مانند “انتظار برای صف” و سیگنال می‌دهد مانند “یک فرد وارد صف می‌شود.” در این کد ما هرگز سیگنال به صف نمی‌دهیم، مگر چنین فردی در حالت انتظار می‌باشد، در نتیجه مقدار صف سمافور به ندرت مثبت می‌باشد. این ممکن است، لذا ببینید چطور انجام می‌شود.

^۵ استفاده از سمافور به عنوان یک صف بسیار شبیه به متغیر شرطی است. تفاوت اصلی در این است که نخ‌ها مجبور به رها کردن موتکس هستند قبل از ورود به حالت انتظار و لازم است تا مجدداً بدست آورده شود (اما تنها زمانی که به آن نیاز است)

synchronization Classical problems

op- every nearly in appear that problems classical the examine we chapter this In
 prob- real-world of terms in presented usually are They textbook. systems erating
 bring can students that so and clear is problem the of statement the that so lems.
 bear. to intuition their
 or world. real the in happen not do problems these though, part, most the For
 synchronization of kind the like much not are solutions real-world the do, they if
 with. working are we code
 to analogous are they that is problems these in interested are we reason The
 solve. to need applications) some (and systems operating that problems common
 explain also and formulation. classical the present will I problem classical each For
 problem. OS corresponding the to analogy the

problem Producer-consumer 1.4

one In threads. between labor of division a often is there programs multithreaded In Producers consumers. are some and producers are threads some pattern. common

the remove consumers structure: data a to them add and kind some of items create them. process and items

hap- that something is “event” An example. good a are programs Event-driven the moves or key a presses user the respond: to program the requires that pens a network. the from arrives packet a disk. the from arrives data of block a mouse. completes. operation pending

adds and object event an creates thread producer a occurs. event an Whenever buffer the of out events take threads consumer Concurrently. buffer. event the to it handlers.” “event called are consumers the case. this In them. process and make to enforce to need we that constraints synchronization several are There correctly: work system this

is buffer the buffer. the from removed or to added being is item an While • the to access exclusive have must threads Therefore. state. inconsistent an in buffer.

pro- a until blocks it empty. is buffer the while arrives thread consumer a If • item. new a adds ducer

over: and over operations following the perform producers that Assume

Basic producer code

```
1 event = waitForEvent()
2 buffer.add(event)
```

operations: following the perform consumers that assume Also.

Basic consumer code

```
1 event = buffer.get()
2 event.process()
```

waitForEvent but exclusive. be to has buffer the to access above. specified As concurrently. run can event.process and to code consumer and producer the to statements synchronization Add Puzzle: constraints. synchronization the enforce

hint Producer-consumer ١.١.٤

use: to want might you variables the are Here

Producer-consumer initialization

```

1 mutex = Semaphore(1)
2 items = Semaphore(0)
3 local event

```

items When buffer. the to access exclusive provides mutex surprisingly. Not
it negative. is it When buffer. the in items of number the indicates it positive. is
queue. in threads consumer of number the indicates
has thread each that means context this in which **variable local** a is event
all to access have threads all that assuming been have we far So version. own its
thread. each to variable a attach to useful it find sometimes will we but variables.
environments: different in implemented be can this ways of number a are There

the on allocated variables any then stack. run-time own its has thread each If ●
thread-specific. are stack

thread each to attribute an add can we objects. as represented are threads If ●
object.

or array an into index an as IDs the use can we IDs. unique have threads If ●
there. data per-thread store and table. hash

this in but otherwise. declared unless local are variables most programs. most In
unless shared are variables that that assume will we so shared. are variables most book
.local declared explicitly are they

solution Producer-consumer ٢.١.٤

solution. my from code producer the is Here

Producer solution

```

1 event = waitForEvent()
2 mutex.wait()
3     buffer.add(event)
4     items.signal()
5 mutex.signal()

```

an gets it until buffer the to access exclusive get to have doesn't producer The
concurrently. `waitForEvent` run can threads Several event.
Each buffer. the in items of number the of track keeps semaphore `items` The
one. by it incrementing `items` signals it item. an adds producer the time
similar. is code consumer The

Consumer solution

```

1 items.wait()
2 mutex.wait()
3     event = buffer.get()
4 mutex.signal()
5 event.process()

```

consumer the before but `mutex`. a by protected is operation buffer the Again.
consumer the negative. or zero is `items` If `items` decrement to has it. to gets
signals. producer a until blocks
small one make to opportunity an is there correct. is solution this Although
in consumer one least at is there that Imagine performance. its to improvement
run. to consumer the allows scheduler the If `items` signals producer a when queue
the by held (still) is that `mutex` the on blocks immediately It next^٩ happens what
producer.
performing operations: expensive moderately are up waking and Blocking
probably would it So program. a of performance the impair can unnecessarily them
this: like producer the rearrange to better be

Improved producer solution

```

1 event = waitForEvent()

```

```

2 mutex.wait()
3   buffer.add(event)
4 mutex.signal()
5 items.signal()

```

proceed can it know we until consumer a unblocking bother don't we Now

mutex). the to it beats producer another that case rare the in (except
the In stickler. a bother might that solution this about thing other one There's
items of number the of track keeps semaphore items the that claimed I section hint
several that possibility the see we code. consumer the at looking But queue. in
removes and mutex the gets them of any before items decrement could consumers
inaccurate. be would items while. little a for least At buffer. the from item an
mutex: the inside buffer the checking by that address to try might We

Broken consumer solution

```

1 mutex.wait()
2   items.wait()
3   event = buffer.get()
4 mutex.signal()
5 event.process()

```

idea. bad a is This

why? Puzzle:

#٤ Deadlock ٣.١.٤

code this running is consumer the If

Broken consumer solution

```

1 mutex.wait()
2     items.wait()
3     event = buffer.get()
4 mutex.signal()
5
6 event.process()

```

arrives. consumer A empty. is buffer the that Imagine deadlock. a cause can it
blocks it arrives. producer the When .items on blocks then and mutex. the gets
halt. grinding a to comes system the and mutex on
a for wait you time any code: synchronization in error common a is This
are you When deadlock. of danger a is there mutex. a holding while semaphore
kind this for check should you problem. synchronization a to solution a checking
deadlock. of

buffer finite a with Producer-consumer ٢.١.٤

usu- is buffer shared the threads. event-handling above. described I example the In
mem- physical like resources system by bounded is it accurately. (more infinite ally
space). swap and ory
space. available on limits are there though. system. operating the of kernel the In
In size. fixed usually are packets network and requests disk like things for Buffers
constraint: synchronization additional an have we these. like situations

removes consumer a until blocks it full. is buffer the when arrives producer a If •
item. an

have we Since .bufferSize it Call buffer. the of size the know we that Assume
write to tempting is it items. of number the of track keeping is that semaphore a
like something

Broken finite buffer solution

```
1 if items >= bufferSize:  
2     block()
```

semaphore: a of value current the check can't we that Remember can't. we But

.signal and wait are operations only the

con- finite-buffer the handles that code producer-consumer write Puzzle:

straint.

hint producer-consumer buffer Finite ٥.١.٢

the in spaces available of number the of track keep to semaphore second a Add
buffer.

Finite-buffer producer-consumer initialization

```

1 mutex = Semaphore(1)
2 items = Semaphore(0)
3 spaces = Semaphore(buffer.size())

```

producer a When .spaces signal should it item an removes consumer a When
next the until block might it point which at ,spaces decrement should it arrives
signals. consumer

solution producer-consumer buffer Finite ٩.١.٤

solution. a is Here

Finite buffer consumer solution

```

1 items.wait()
2 mutex.wait()
3     event = buffer.get()
4 mutex.signal()
5 spaces.signal()
6
7 event.process()

```

way: a in symmetric. is code producer The

Finite buffer producer solution

```

1 event = waitForEvent()
2
3 spaces.wait()
4 mutex.wait()
5     buffer.add(event)
6 mutex.signal()
7 items.signal()

```

before availability check consumers and producers deadlock. avoid to order In
signaling. before mutex the release they performance. best For mutex. the getting

problem Readers-writers ٢.٤

sit- any to pertains Problem. Reader-Writer the called problem. classical next The
con- by modified and read is system file or database. structure. data a where uation
often is it modified or written being is structure data the While threads. current
in- from reader a prevent to order in reading. from threads other bar to necessary
data. invalid or inconsistent reading and progress in modification a interrupting
Readers asymmetric. is solution the problem. producer-consumer the in As
syn- The section. critical the entering before code different execute writers and
are: constraints chronization

simultaneously. section critical the in be can readers of number Any .١

section. critical the to access exclusive have must Writers .۲

thread other any while section critical the enter cannot writer a words, other In
 enter. may thread other no there. is writer the while and there. is writer) or (reader
 A .**exclusion mutual categorical** called be might here pattern exclusion The
 the but threads. other exclude necessarily not does section critical the in thread
 categories. other excludes section critical the in category one of presence
 readers allowing while constraints. these enforce to semaphores Use Puzzle:
 deadlock. of possibility the avoiding and structure. data the access to writers and

hint Readers-writers ١.٢.٤

problem. the solve to sufficient is that variables of set a is Here

Readers-writers initialization

```

1 int readers = 0
2 mutex = Semaphore(1)
3 roomEmpty = Semaphore(1)

```

mutex room. the in are readers many how of track keeps readers counter The
 counter. shared the protects
 section. critical the in writers) or (readers threads no are there if \ is roomEmpty
 semaphores for use I convention naming the demonstrates This otherwise. • and
 the for “wait means usually “wait” convention. this In condition. a indicate that
 true”. is condition the that “signal means “signal” and true” be to condition

solution Readers-writers ٢.٢.٢

enter, may writer a empty, is section critical the If simple, is writers for code The
threads: other all excluding of effect the has entering but

Writers solution

```
1 roomEmpty.wait()
2     critical section for writers
3 roomEmpty.signal()
```

because Yes, empty? now is room the that sure be it can exits, writer the When
there, was it while entered have can thread other no that knows it
section, previous the in saw we code barrier the to similar is readers for code The
special a give can we that so room the in readers of number the of track keep We
leave, to last the and arrive to first the to assignment
empty, is room the If .roomEmpty for wait to has arrives that reader first The
readers Subsequent writers, bars time, same the at and, proceeds reader the then
.roomEmpty on wait to try will them of none because enter still can
.roomEmpty on waits it room, the in writer a is there while arrives reader a If
.mutex on queue readers subsequent any mutex, the holds it Since

Readers solution

```
1 mutex.wait()
2     readers += 1
3     if readers == 1:
4         roomEmpty.wait()      # first in locks
5 mutex.signal()
6
7 # critical section for readers
8
9 mutex.wait()
10    readers -= 1
11    if readers == 0:
12        roomEmpty.signal()    # last out unlocks
13 mutex.signal()
```

room the leave to reader last The similar, is section critical the after code The
waiting a allowing possibly .roomEmpty signals it is, lights—that the out turns
enter, to writer

demon- and assert to useful is it correct. is code this that demonstrate to Again.
convince you Can behave. must program the how about claims of number a strate
true? are following the that yourself

might writers several but ,roomEmpty for waiting queue can reader one Only •
queued. be

empty. be must room the roomEmpty signals reader a When •

sec- a into thread first the common: are code reader this to similar Patterns
so is it fact. In it. unlocks out one last the and queues) (or semaphore a locks tion
object. an in up it wrap and name a it give should we common
the where pattern the with analogy by ,**Lightswitch** is pattern the of name The
out one last the and mutex) the (locks light the on turns room a into person first
Lightswitch: a for definition class a is Here mutex). the (unlocks off it turns

Lightswitch definition

```

1 class Lightswitch:
2     def __init__(self):
3         self.counter = 0
4         self.mutex = Semaphore(1)
5
6     def lock(self, semaphore):
7         self.mutex.wait()
8         self.counter += 1
9         if self.counter == 1:
10            semaphore.wait()
11        self.mutex.signal()
12
13    def unlock(self, semaphore):
14        self.mutex.wait()
15        self.counter -= 1
16        if self.counter == 0:
17            semaphore.signal()
18        self.mutex.signal()

```

If hold. possibly and check will it that semaphore a parameter. one takes lock
subsequent all and semaphore on blocks thread calling the locked. is semaphore the
waiting first the unlocked. is semaphore the When .self.mutex on block threads
proceed. threads waiting all and again it locks thread

subsequent all and it locks thread first the unlocked. initially is semaphore the If
 proceed. threads
 When .unlock calls also lock called that thread every until effect no has unlock
 semaphore. the unlocks it .unlock calls thread last the

simply: more code reader the rewrite can we functions: these Using

Readers-writers initialization

```
1 readLightswitch = Lightswitch()
2 roomEmpty = Semaphore(1)
```

initially is counter whose object Lightswitch shared a is readLightswitch

zero.

Readers-writers solution (reader)

```
1 readLightswitch.lock(roomEmpty)
2 # critical section
3 readLightswitch.unlock(roomEmpty)
```

unchanged. is writers for code The

the of attribute an as roomEmpty to reference a store to possible be also would It

alternative This .unlock and lock to parameter a as it pass than rather Lightswitch.

of invocation each if readability improves it think I but error-prone. less be would

on. operates it semaphore the specifies unlocks and lock

Starvation ۳.۲.۴

deadlock a for order In deadlock? of danger any there is solution: previous the In

holding while semaphore a on wait to thread a for possible be must it occur: to

signaled. being from itself prevent thereby and another.

is that problem related a is there but possible. not is deadlock example. this In

starve. to writer a for possible is it bad: as almost

in wait might it section. critical the in readers are there while arrives writer a If

before arrives reader new a as long As go. and come readers while forever queue

the in reader one least at be always will there departs. readers current the of last the

room.

but progress. making are threads some because deadlock. a not is situation This

the on load the as long as work might this like program A desirable. exactly not is it

But writers. the for opportunities of plenty are there then because low. is system

least (at quickly deteriorate would system the of behavior the increases load the as
writers). of view of point the from
readers existing the arrives. writer a when that so solution this Extend Puzzle:
enter. may readers additional no but finish. can

hint readers-writers No-starve 7.4.4

lock to writers allow and readers the for turnstile a add can You hint. a Here's the check should they but turnstile. same the through pass to have writers The it. in stuck gets writer a If turnstile. the inside are they while semaphore roomEmpty Then turnstile. the at queue to readers the forcing of effect the has it turnstile the one least at that guaranteed are we section. critical the leaves reader last the when proceed). can readers queued the of any (before next enters writer

No-starve readers-writers initialization

```

1 readSwitch = Lightswitch()
2 roomEmpty = Semaphore(1)
3 turnstile = Semaphore(1)

```

locks it room: the in are readers many how of track keeps readSwitch exits. reader last the when it unlocks and enters reader first the when roomEmpty writers. for mutex a and readers for turnstile a is turnstile

solution readers-writers No-starve 5.2.4

code: writer the is Here

No-starve writer solution

```

1 turnstile.wait()
2   roomEmpty.wait()
3   # critical section for writers
4 turnstile.signal()
5
6 roomEmpty.signal()

```

Line at block will it room, the in readers are there while arrives writer a If entering from readers bar will This locked. be will turnstile the that means which code: reader the is Here queued. is writer a while

No-starve reader solution

```

1 turnstile.wait()
2 turnstile.signal()
3
4 readSwitch.lock(roomEmpty)
5   # critical section for readers
6 readSwitch.unlock(roomEmpty)

```

waiting the unblocking ,roomEmpty signals it leaves. reader last the When waiting the of none since section. critical its enters immediately writer The writer. turnstile. the pass can readers thread. waiting a unblocks which ,turnstile signals it exits writer the When one least at that guarantees solution this Thus. writer. a or reader a be could which are there while enter to reader a for possible still is it but proceed. to gets writer queued. writers to priority more give to idea good a be might it application. the on Depending structure. data a to updates time-critical making are writers if example. For writers. writer the before data old the see that readers of number the minimize to best is it proceed. to chance a has choose to programmer. the not scheduler. the to up is it though. general. In queue. first-in-first-out a use schedulers Some unblock. to thread waiting which

Other queued. they order same the in unblocked are threads that means which
prop- the on based scheme priority a to according or random. at choose schedulers
threads. waiting the of erties
pri- threads some give to possible it makes environment programming your If
will you not. If issue. this address to way simple a is that then others. over ority
way. another find to have
to priority gives that problem readers-writers the to solution a Write Puzzle:
until enter to allowed be should readers no arrives. writer a once is. That writers.
system. the left have writers all

hint readers-writers Writer-priority 9.2.4

solution. the in used variables of form the in is hint the usual. As

Writer-priority readers-writers initialization

```
1 readSwitch = Lightswitch()  
2 writeSwitch = Lightswitch()  
3 noReaders = Semaphore(1)  
4 noWriters = Semaphore(1)
```


solution readers-writers Writer-priority 7.4.4

code: reader the is Here

Writer-priority reader solution

```

1 noReaders.wait()
2   readSwitch.lock(noWriters)
3 noReaders.signal()
4
5   # critical section for readers
6
7 readSwitch.unlock(noWriters)
```

hold doesn't it but `noWriters` holds it section, critical the in is reader a If
 sub- cause will which `noReaders` lock can it arrives writer a if Thus `noReaders`
 queue, to readers sequent
 writers queued any allowing `noWriters` signals it exits, reader last the When
 proceed, to
 code: writer The

Writer-priority writer solution

```

1 writeSwitch.lock(noReaders)
2   noWriters.wait()
3   # critical section for writers
4   noWriters.signal()
5 writeSwitch.unlock(noReaders)
```

`noWriters` and `noReaders` both holds it section critical the in is writer a When
 no and readers no are there that insuring of effect obvious) (relatively the has This
 obvious) (less the has `writeSwitch` addition, In section, critical the in writers other
`noReaders` keeping but `noWriters` on queue to writers multiple allowing of effect
 section critical the through pass can writers many Thus, there, are they while locked
 readers the can exits writer last the when Only `noReaders` signaling ever without
 enter.

to *readers* for possible is it now that is solution this of drawback a course, Of
 get to better be might it applications some For delays), long face least at (or starve
 times, turnaround predictable with data obsolete

mutex No-starve 3.4

in **starvation categorical** call I'll what addressed we section. previous the In
starve. to (writers) category another allows (readers) threads of category one which
is which **starvation thread** of issue the address to have we level. basic more a At
proceed. others while indefinitely wait might thread one that possibility the
the enforce we so unacceptable. is starvation applications. concurrent most For
a on waits thread a time the that means which **waiting bounded** of requirement
finite. provably be to has matter) that for else. anywhere (or semaphore
multiple Whenever scheduler. the of responsibility the is starvation part. In
processor. parallel a on or. one which decides scheduler the run. to ready are threads
starve. will it then scheduled. never is thread a If run. to gets threads of set which
semaphores. with do we what matter no
as- some with start to have we starvation. about anything say to order in So
we assumption. strong a make to willing are we If scheduler. the about sumptions
proven be can that algorithms many the of one uses scheduler the that assume can
uses. scheduler the algorithm what know don't we If waiting. bounded enforce to
assumption: weaker a with by get can we then
scheduler the run. to ready is that thread one only is there if : \forall Property
run. it let to has
of free provably is that system a build can we then \exists Property assume can we If
program any then threads. of number finite a are there if example. For starvation.
be will one but threads all eventually since starve. cannot barrier a contains that
run. to has thread last the point which at barrier. the at waiting
star- from free are that programs write to non-trivial is it though. general. In
assumption: stronger the make we unless vation
runs it until waits it time the then run. to ready is thread a if : \forall Property
bounded. is
we and implicitly. \forall Property assuming been have we far. so discussion our In

systems existing many that know should you hand. other the On to. continue will strictly. property this guarantee not do that schedulers use introduce we when again head ugly its rears starvation. \forall Property with Even exe- thread one when that said we semaphore. a of definition the In semaphores. which said never we But up. woken gets threads waiting the of one. signal cutes about assumptions make to have we starvation. about anything say to order In one. semaphores. of behavior the is: starvation avoid to possible it makes that assumption weakest The

thread a when semaphore a on waiting threads are there if \forall Property woken. be to has threads waiting the of one then. signal executes

effect the has It trivial. not is it but obvious. seem may requirement This a signals that thread a is which behavior. problematic of form one barring of the on waits running. keeps then and waiting. are threads other while semaphore be would there possible. were that If signal! own its gets and semaphore. same starvation. prevent to do could we nothing some- for even but starvation. avoid to possible becomes it. \forall Property With run- threads three imagine example. For easy. not is it mutex. a as simple as thing code: following the ning

Mutex loop

```

1 while True:
2     mutex.wait()
3     # critical section
4     mutex.signal()
```


thread a as soon as words. other in loop: infinite an is statement while The again. mutex the get to tries and top the to loops it section. critical the leaves A When wait. C and B Thread and mutex the gets A Thread that Imagine queue. the in C joins and around loops A leaves. B before but enters. B leaves. and next. goes A if fact. In next. goes C that guarantee no is there leaves. B When the repeat can we and position. starting the to back are we then queue. the joins B starves. C forever. cycle

starvation. to vulnerable is mutex the that proves pattern this of existence The
so semaphore the of implementation the change to is problem this to solution One
property: stronger a guarantees it that

of number the then semaphore. a at waiting is thread a if : \forall Property
bounded. is it before woken be will that threads

Prop- then queue. first-in-first-out a maintains semaphore the if example. For
ahead threads of number the queue. the joins thread a when because holds \forall erty
it. of ahead go can later arrive that threads no and finite. is it of

one :**semaphore strong** a called sometimes is \forall Property has that semaphore A
with that shown have We .**semaphore weak** a called is \forall Property only has that
fact. In starvation. to vulnerable is solution mutex simple the semaphores. weak
star- without problem mutex the solve to possible not is it that conjectured Dijkstra
semaphores. weak only using vation

assuming problem. the solving by conjecture the refuted Morris J.M. , 1979 In
the problem. this in interested are you If .[assume just can you fun. of idea your not is this If solution. his presents section next

. \forall . \forall Section to on go and \forall Property have semaphores that

weak using problem exclusion mutual the to solution a write Puzzle:
thread a once guarantee: following the provide should solution Your semaphores.
threads of number the on bound a is there mutex. the enter to attempts and arrives
is threads of number total the that assume can You it. of ahead proceed can that
finite.

hint mutex No-starve 1.3.4

turn- two uses It .V.3 Section in barrier reusable the to similar is solution Morris's works mechanism The section. critical the before rooms waiting two create to stiles is second the and open is turnstile first the phase. first the During phases. two in the phase. second the During room. second the in accumulate threads so closed. the so open. is second the and enter. can threads new no so closed. is turnstile first

section. critical the to get can threads existing room. waiting the in threads of number arbitrary an be may there Although

arrivals. future any before section critical the enter to guaranteed is one each used. Morris names the changed (I solution the in used I variables the are Here

clearer). structure the make to trying

No-starve mutex hint

```

1 room1 = room2 = 0
2 mutex = Semaphore(1)
3 t1 = Semaphore(1)
4 t2 = Semaphore(0)

```

rooms. waiting the in are threads many how of track keep room2 and room1

turnstiles. the are t2 and t1 counters. the protect helps mutex

solution mutex No-starve २.२.२

solution. Morris's is Here

Morris's algorithm

```

1 mutex.wait()
2     room1 += 1
3 mutex.signal()
4
5 t1.wait()
6     room2 += 1
7     mutex.wait()
8     room1 -= 1
9
10    if room1 == 0:
11        mutex.signal()
12        t2.signal()
13    else:
14        mutex.signal()
15        t1.signal()
16
17 t2.wait()
18     room2 -= 1
19
20    # critical section
21
22    if room2 == 0:
23        t1.signal()
24    else:
25        t2.signal()

```

These turnstiles, two pass to has thread a section, critical the entering Before is २ Room . 1-2 Lines is 1 Room “rooms”. three into code the divide turnstiles room2 and room1 counters the loosely, Speaking rest, the is ३ Room . 11-12 Lines room, each in threads of number the of track keep for duty guard but way, usual the in mutex by protected is room1 counter The to access exclusive for responsibility Similarly, .t2 and t1 between split is room2 a section, critical the enter to order In .t2 and t1 both involves section critical the up gives it exiting, before Then, both, not but other, the or one hold to has thread has, it one whichever all thread single a following by start works, solution this how understand To

checks it Once τ_1 and mutex holds it, \wedge Line to gets it When through, way the
 As τ_2 turnstile, second the open then and mutex release can it, \cdot is which room1
 the enter and room2 decrement safely can it and \vee Line at wait doesn't it result, a
 Leaving τ_1 on queued be to have threads following any because section, critical
 the to back us brings which τ_1 releases and $0 = \text{room2}$ finds it section, critical the
 state, starting

In thread, one than more is there if interesting more is situation the course. Of
 have threads other, \wedge Line to gets thread lead the when that possible is it case, that
 thread lead the $0 > \text{room1}$ Since τ_1 on queued and room waiting the entered
 Room enter to thread following a allow to τ_1 signals instead and locked τ_2 leaves
 τ_2 Room enter can thread neither locked, still is τ_2 Since τ_2

to get will thread a threads, of number finite a are there (because Eventually
 allowing τ_2 open will it case which in \vee Room enters thread another before \wedge Line
 hold to continues τ_2 opens that thread The τ_2 Room into move to there threads any
 Δ Line at block will they around, loop threads lead the of any if so τ_1

leave to thread another allowing τ_2 signals it, τ_2 Room leaves thread each As
 τ_1 opens and locked τ_2 leaves it, τ_2 Room leaves thread last the When τ_2 Room
 state, starting the to back us brings which

in operation its of think to helps it starvation, avoids solution this how see To
 and room1 increment \vee Room into check threads phase, first the In phases, two
 is locked τ_2 a keep to way only The time, a at one τ_2 Room into cascade then
 finite a are there Because \vee Room through threads of procession a maintain to
 stays τ_1 point which at eventually, end to has procession the threads, of number
 opens, τ_2 and locked

finite a are there Because τ_2 Room into cascade threads phase, second the In
 last the eventually enter, can threads new no and τ_2 Room in threads of number
 opens, τ_1 and locks τ_2 point which at leaves, thread
 τ_1 at waiting threads no are there that know we phase, first the of end the At
 are there that know we phase, second the of end the at And $0 = \text{room1}$ because
 $0 = \text{room2}$ because τ_2 at waiting threads no

can thread one if possible only is starvation threads. of number finite a With
that makes mechanism two-turnstile the But another. overtake and around loop
impossible. is starvation so impossible.
pre- to difficult very is it semaphores. weak with that is story this of moral The
this of rest the In problems. synchronization simplest the for even starvation. vent
semaphores. strong assume will we starvation. discuss we when book.

philosophers Dining 4.4

di- when 1965 in Dijkstra by proposed was Problem Philosophers Dining The standard the but variations. of number a in appears It [3] earth the ruled nosaurs of bowl big a and chopsticks) (or forks five plates. five with table a are features table the to come threads. interacting represent who philosophers. Five spaghetti.

loop: following the execute and

Basic philosopher loop

```

1 while True:
2     think()
3     get_forks()
4     eat()
5     put_forks()

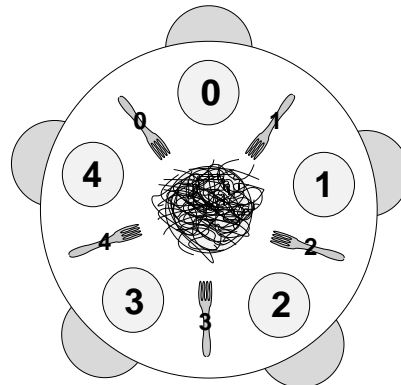
```

order in exclusively hold to have threads the that resources represent forks The and unrealistic. interesting. problem the makes that thing The progress. make to philosopher hungry a so eat. to forks two need philosophers the that is unsanitary.

fork. a down put to neighbor a for wait to have might

philoso- each identifies that i variable local a have philosophers the that Assume that so 4 to 0 from numbered are forks the Similarly. (0,4) in value a with pher of diagram a is Here left. the on $i + 1$ fork and right the on i fork has i Philosopher

situation: the



write to is job our eat and think to how know philosophers the that Assuming constraints: following the satisfies that put_forks and get_forks of version a

time. a at fork a hold can philosopher one Only •

occur. to deadlock a for impossible be must It •

fork. a for waiting starve to philosopher a for impossible be must It •

time. same the at eat to philosopher one than more for possible be must It •

efficient: be should solution the that saying of way one is requirement last The

concurrency. of amount maximum the allow should it is. that

eat that except take. think and eat long how about assumption no make We

a impossible—if is constraint third the Otherwise. eventually. terminate to has

from neighbors the prevent can nothing forever. forks the of one keeps philosopher

starving.

functions the use can we forks. their to refer to philosophers for easy it make To

:right and left

Which fork?

```
1 def left(i): return i
2 def right(i): return (i + 1) % 5
```

.0 = 5 % 1) + (4 so .0 to gets it when around wraps operator % The

list a use to natural is it forks. the to access exclusive enforce to have we Since

available. are forks the all Initially. fork. each for one Semaphores. of

Variables for dining philosophers

```
1 forks = [Semaphore(1) for i in range(5)]
```

use don't who readers to unfamiliar be might list a initializing for notation This

this of element each for elements: 0 with list a returns function range The Python.

a in result the assembles and \ value initial with Semaphore a creates Python list.

.forks named list

:put_fork and get_fork at attempt initial an is Here

Dining philosophers non-solution

```
1 def get_forks(i):
2     fork[right(i)].wait()
```

```
3     fork[left(i)].wait()
4
5 def put_forks(i):
6     fork[right(i)].signal()
7     fork[left(i)].signal()
```

sure pretty be can we but constraint. first the satisfies solution this that clear It's
interesting an be wouldn't this did. it if because two. other the satisfy doesn't it
.⚠ Chapter reading be would you and problem
wrong? what's Puzzle:

#۵ Deadlock ۱.۴.۴

a up pick can philosopher each result. a As round. is table the that is problem The
 Deadlock! fork. other the for forever wait then and fork
 deadlock. prevents that problem this to solution a write Puzzle:
 make that conditions the about think to is deadlock avoid to way one Hint:
 dead- the case. this In conditions. those of one change then and possible deadlock
 it. breaks change small very fragile—a fairly is lock

#\ hint philosophers Dining ٢.٢.٢

impossible. is deadlock time. a at table the at allowed are philosophers four only If
the limits that code write then true. is claim this that yourself convince First.

table. the at philosophers of number

#1 solution philosophers Dining 3.4.4

picks one each case worst the in then table. the at philosophers four only are there If
 neigh- two has fork that and table. the on left fork a is there then. Even fork. a up
 neighbors these of either Therefore. fork. another holding is which of each bors.
 eat. and fork remaining the up pick can
 named Multiplex a with table the at philosophers of number the control can We
 this: like looks solution the Then .4 to initialized is that footman

Dining philosophers solution #1

```

1 def get_forks(i):
2     footman.wait()
3     fork[right(i)].wait()
4     fork[left(i)].wait()
5
6 def put_forks(i):
7     fork[right(i)].signal()
8     fork[left(i)].signal()
9     footman.signal()

```

philoso- no that guarantees also solution this deadlock. avoiding to addition In
 are neighbors your of both and table the at sitting are you that Imagine starves. pher
 neigh- right your Eventually fork. right your for waiting blocked are You eating.
 thread only the are you Since forever. run can't eat because down. it put will bor
 you argument. similar a By next. it get necessarily will you fork. that for waiting
 fork. left your for waiting starve cannot
 im- That bounded. is table the at spend can philosopher a time the Therefore.
 has footman as long as bounded. also is room the into get to time wait the that plies

.(3.4 Section (see 4 Property

can we philosophers. of number the controlling by that shows solution This
 which in order the change to is deadlock avoid to way Another deadlock. avoid
 are philosophers the non-solution. original the In forks. up pick philosophers the
 Philosopher if happens what But first. fork right the up pick they is. that "righties":
 leftie? a is •
 then rightie. one least at and leftie one least at is there if that prove Puzzle:

possible. not is deadlock
 fork one holding are philosophers Δ all when occur only can deadlock Hint:
 forks. both get could them of one Otherwise. other. the for forever. waiting. and
 leave. and eat.
 Then possible. is deadlock that assume First. contradiction. by works proof The
 can you leftie. a she's If philosophers. deadlocked supposedly the of one choose
 she's if Similarly. contradiction. a is which lefties. all are philosophers the that prove
 contradiction: a get you way Either righties. all are they that prove can you rightie. a
 possible. not is deadlock therefore.

#2 solution philosopher's Dining 4.4.4

at be to has there problem. philosophers Dining the to solution asymmetric the In
 impos- is deadlock case. that In table. the at rightie one least at and leftie one least
 details. the are Here proof. the outlines hint previous The sible.
 one holding are philosophers ϕ all when occurs it possible. is deadlock if Again.
 she then leftie. a is j Philosopher that assume we If other. the for waiting and fork
 neighbor her Therefore right. her for waiting and fork left her holding be must
 right his for waiting and fork left his holding be must $\neg k$ Philosopher right. the to
 ar- same the Repeating leftie. a be must k Philosopher words. other in neighbor:
 the contradicts which lefties. all are philosophers the that prove can we gument.
 possible. not is deadlock Therefore rightie. one least at is there that claim original
 starvation that proves also solution previous the for used we argument same The
 solution. this for impossible is

solution Tanenbaum's 4.4.4

let's completeness, for just but solutions, previous the with wrong nothing is There Tanen- in appears that one the is known best the of One alternatives, some at look a is there philosopher each For .[12] textbook systems operating popular baum's waiting or eating, thinking, is philosopher the whether indicates that variable state start can philosopher the whether indicates that semaphore a and ("hungry") eat to variables: the are Here eating.

Variables for Tanenbaum's solution

```
1 state = ['thinking'] * 5
2 sem = [Semaphore(0) for i in range(5)]
3 mutex = Semaphore(1)
```

Δ of list a is sem . 'thinking' of copies Δ of list a is state of value initial The code: the is Here . • value initial the with semaphores

Tanenbaum's solution

```
1 def get_fork(i):
2     mutex.wait()
3     state[i] = 'hungry'
4     test(i)
5     mutex.signal()
6     sem[i].wait()
7
8 def put_fork(i):
9     mutex.wait()
10    state[i] = 'thinking'
11    test(right(i))
12    test(left(i))
13    mutex.signal()
14
15 def test(i):
16     if state[i] == 'hungry' and
17        state[left(i)] != 'eating' and
18        state[right(i)] != 'eating':
19         state[i] = 'eating'
20         sem[i].signal()
```

which eating, start can philosopher thi the whether checks function test The signals test the so. If eating, are neighbors his of neither and hungry is he if can he

.i semaphore

philosopher the case, first the In eat. to gets philosopher a ways two are There
 the In immediately. proceeds and available. forks the finds ,get_forks executes
 own its on blocks philosopher the and eating is neighbors the of one case, second
 executes it point which at finish. will neighbors the of one Eventually, semaphore.
 which in succeed. will tests both that possible is It neighbors. its of both on test
 matter. doesn't tests two the of order The concurrently. run can neighbors the case
 the Thus. .mutex hold to has thread a ,test invoke or state access to order In
 can philosopher a Since atomic. is array the updating and checking of operation
 forks the to access exclusive available. are forks both know we when proceed only
 guaranteed. is
 more by accessed is that semaphore only the because possible. is deadlock No
 .mutex holding while wait executes thread no and ,mutex is philosopher one than
 tricky. is starvation again. But
 starvation prevents solution Tanenbaum's that yourself convince Either Puzzle:
 make threads other while starve to thread a allows that pattern repeating a find or
 progress.

Tanenbaums Starving 4.4.4

there that demonstrated Gengras starvation-free. not is solution this Unfortunately.
come threads other while forever wait to thread a allow that patterns repeating are

. $[P]$ go and

the at are P and Q Initially. \bullet Philosopher starve to trying are we that Imagine
gets P then down: sit \backslash and up gets Q that Imagine hungry. are Q and \backslash and table
position. starting the of image mirror the in are we Now down. sits Q and up
back are we down. sits Q and up gets \backslash then and down. sits P and up gets Q If
would \bullet Philosopher and indefinitely cycle the repeat could We started. we where
starve.

requirements. the all satisfy doesn't solution Tanenbaum's So.

problem smokers Cigarette 5.4

who, [A] Patil Suhas by presented originally was problem smokers cigarette The some with comes claim That semaphores, with solved be cannot it that claimed challenging, and interesting is problem the case any in but qualifications, for- loop smokers The smokers, three and agent an involved: are threads Four in- The cigarettes, smoking and making then ingredients, for waiting first ever, matches, and paper, tobacco, are gredients and ingredients, three all of supply infinite an has agent the that assume We has smoker one is, that ingredients: the of one of supply infinite an has smoker each tobacco, has third the and paper, has another matches, makes and random at ingredients different two chooses repeatedly agent The the chosen, are ingredients which on Depending smokers, the to available them and resources both up pick should ingredient complementary the with smoker proceed, the with smoker the paper, and tobacco out puts agent the if example, For the signal then and cigarette, a make ingredients, both up pick should matches agent, allocates that system operating an represents agent the premise, the explain To problem The resources, need that applications represent smokers the and resources, applications more one allow would that available are resources if that sure make to is avoid to want we Conversely, up, woken be should applications those proceed, to proceed, cannot it if application an waking appear often that problem this of versions three are there premise, this on Based textbooks: in

solution, the on restrictions imposes version Patil's **version: impossible** The an represents agent the If code, agent the modify to allowed not are you First, it modify to want don't you that assume to sense makes it system, operating you that is restriction second The along, comes application new a time every con- these With semaphores, of array an or statements conditional use can't

second the out. points Parnas as but solved. be cannot problem the strains.
 problems of lot a these. like constraints With .[✓] artificial pretty is restriction
 unsolvable. become

can't restriction—you first the keeps version This **version: interesting** The
 others. the drops it code—but agent the change

agent the that specifies problem the textbooks. some In **version: trivial** The
 ingredients the to according next. go should that smoker the signal should
 it because uninteresting is problem the of version This available. are that
 Also. irrelevant. cigarettes. the and ingredients the premise. whole the makes
 know to agent the require to idea good a not probably is it matter. practical a as
 of version this Finally. for. waiting are they what and threads other the about
 easy. too just is problem the

statement the complete To version. interesting the on focus will we Naturally.
 following the uses agent The code. agent the specify to need we problem. the of
 semaphores:

Agent semaphores

```
1 agentSem = Semaphore(1)
2 tobacco = Semaphore(0)
3 paper = Semaphore(0)
4 match = Semaphore(0)
```

B Agent A. Agent threads. concurrent three of up made actually is agent The
 the of one signaled. is agentSem time each :agentSem on waits Each C. Agent and
 semaphores. two signaling by ingredients provides and up wakes Agents

Agent A code

```
1 agentSem.wait()
2 tobacco.signal()
3 paper.signal()
```

Agent B code

```
1 agentSem.wait()
2 paper.signal()
3 match.signal()
```

Agent C code

```
1 agentSem.wait()
2 tobacco.signal()
3 match.signal()
```

tempting is It work. not does solution natural the because hard is problem This

like: something write to

Smoker with matches

```
1 tobacco.wait()
2 paper.wait()
3 agentSem.signal()
```

Smoker with tobacco

```
1 paper.wait()
2 match.wait()
3 agentSem.signal()
```

Smoker with paper

```
1 tobacco.wait()
2 match.wait()
3 agentSem.signal()
```

solution? this with wrong What's

#6 Deadlock 1.5.4

that Imagine deadlock. of possibility the is solution previous the with problem The
 waiting is matches with smoker the Since paper. and tobacco out puts agent the
 on waiting is tobacco with smoker the But unblocked. be might it .tobacco on
 first the Then unblocked. be also will it that likely) (even possible is it so .paper
 Deadlock! .match on block will second the and paper on block will thread

hint problem Smokers ٢.٥.٢

re- that “pushers” called threads helper three uses Parnas by proposed solution The
and ingredients. available the of track keep agent. the from signals the to spond
smoker. appropriate the signal
are semaphores and variables additional The

Smokers problem hint

```
1 isTobacco = isPaper = isMatch = False
2 tobaccoSem = Semaphore(0)
3 paperSem = Semaphore(0)
4 matchSem = Semaphore(0)
```

table. the on is ingredient an not or whether indicate variables boolean The
other the and tobacco. with smoker the signal to tobaccoSem use pushers The
likewise. semaphores

solution problem Smoker ۳.۵.۴

pushers: the of one for code the is Here

Pusher A

```

1 tobacco.wait()
2 mutex.wait()
3     if isPaper:
4         isPaper = False
5         matchSem.signal()
6     elif isMatch:
7         isMatch = False
8         paperSem.signal()
9     else:
10        isTobacco = True
11 mutex.signal()

```

isPaper finds it If table. the on tobacco is there time any up wakes pusher This with smoker the signal can it so run. already has B Pusher that knows it true. with smoker the signal can it table. the on match a finds it if Similarly. matches. paper.

It false. isMatch and isPaper both find will it then first. runs A Pusher if But .isTobacco sets it so smokers. the of any signal cannot smoker the work. real the all do pushers the Since similar. are pushers other The trivial: is code

Smoker with tobacco

```

1 tobaccoSem.wait()
2 makeCigarette()
3 agentSem.signal()
4 smoke()

```

bitwise. variables. boolean the assembles that solution similar a presents Parnas semaphores. of array an into index an as integer the uses then and integer. an into The constraints). artificial the of (one conditionals using avoid can he way That obvious. as not is function its but concise. more bit a is code resulting

Problem Smokers Generalized ۴.۵.۴

the modify we if difficult more becomes problem smokers the that suggested Parnas ingredients. out putting after wait agent the that requirement the eliminating agent.

table. the on ingredient an of instances multiple be might there case. this In variation. this with deal to solution previous the modify Puzzle:

Hint Problem Smokers Generalized 5.5.4

table. the on accumulate might ingredients smokers. the for wait don't agents the If
to integers need we ingredients. of track keep to values boolean using of Instead
them. count

Generalized Smokers problem hint

```
numTobacco = numPaper = numMatch = 0
```


Solution Problem Smokers Generalized ۶.۵.۴

A: Pusher for code modified the is Here

Pusher A

```

1 tobacco.wait()
2 mutex.wait()
3     if numPaper:
4         numPaper -= 1
5         matchSem.signal()
6     elif numMatch:
7         numMatch -= 1
8         paperSem.signal()
9     else:
10        numTobacco += 1
11 mutex.signal()

```

it runs. Agent an when that imagine to is problem this visualize to way One room a in them puts and ingredient, one them of each gives pushers, two creates where room a into file pushers the mutex, the of Because pushers. other the all with the enters pusher each time, a at One table, a and smokers sleeping three are there set complete a assemble can he If table, the on ingredients the checks and room If smoker, corresponding the wakes and table the off them takes he ingredients, of anyone, waking without leaves and table the on ingredient his leaves he not.

score- a call I which times, several see will we pattern a of example an is This state the of track keep numMatch and numTobacco ,numPaper variables The **.board** looking ifas state the checks it mutex, the through files thread each As system, the of accordingly, reacts and scoreboard, the at

Less classical synchronization problems

١.٥ The dining savages problem

This problem is from Andrews's *Programming Concurrent* [١].

A tribe of savages eats communal dinners from a large pot that can hold M servings of stewed missionary. When a savage wants to eat, he holds the pot if it is empty, or helps himself from the pot unless it is empty. If the pot is not empty, he waits until the cook has refilled the pot.

Any number of threads run following the code:

This problem is based on a cartoonish representation of the history of Western missionaries among hunter-gatherer societies. Some humor is intended by the allusion to the Dining Philosophers problem—previous to the more realistic than any be intended isn't here “savages” of representation the but lem. Jared recommend I societies. hunter-gatherer in interested are you If philosophers. of representation O'Hanlon's Redmond and *Yanomamo* The Chagnon's Napoleon. *Steel and Germs Guns*. Diamond's unreliable. is believe I which *Dorado El in Darkness* Tierney's not but *Again Trouble In*

Unsynchronized savage code

```
1 while True:
2     getServingsFromPot()
3     eat()
```

code: this runs thread cook one And

Unsynchronized cook code

```
1 while True:
2     putServingsInPot(M)
```

are: constraints synchronization The

empty. is pot the if `getServingsFromPot` invoke cannot Savages •

empty. is pot the if only `putServingsInPot` invoke can cook The •

synchronization the satisfies that cook the and savages the for code Add Puzzle:

constraints.

hint Savages Dining ۱.۱.۵

in as servings. of number the of track keep to semaphore a use to tempting is It
 is pot the when cook the signal to order in But problem. producer-consumer the
 whether semaphore the decrementing before know to have would thread a empty.

that. do can't just we and wait. to have would it

If servings. of number the of track keep to scoreboard a use to is alternative An
 the that signal a for waits and cook the wakes he zero. at counter the finds savage a

used: I variables the are Here full. is pot

Dining Savages hint

```

1 servings = 0
2 mutex = Semaphore(1)
3 emptyPot = Semaphore(0)
4 fullPot = Semaphore(0)

```

indi- fullPot and empty is pot the that indicates emptyPot surprisingly. Not

full. is pot the that cates

solution Savages Dining ۲.۱.۵

is Here rendezvous. a with pattern scoreboard the of combination a is solution My
cook: the for code the

Dining Savages solution (cook)

```
1 while True:
2     emptyPot.wait()
3     putServingsInPot(M)
4     fullPot.signal()
```

passes savage each As complicated. more little a only is savages the for code The
waits. and cook the signals he empty. is it If pot. the checks he mutex. the through
pot. the from serving a gets and servings decrements he Otherwise.

Dining Savages solution (savage)

```
1 while True:
2     mutex.wait()
3     if servings == 0:
4         emptyPot.signal()
5         fullPot.wait()
6         servings = M
7         servings -= 1
8         getServicingFromPot()
9     mutex.signal()
10
11     eat()
```

.M = servings sets cook. the than rather savage. the that odd seem might It
that know we .putServingsInPot runs cook the when necessary: really not That's
access could cook the So .fullPot on waiting is mutex the holds that savage the
clear is it that so it do savage the let to decided I case. this in But safely. servings
mutex. the inside are servings to accesses all that code the at looking from
when comes deadlock for opportunity only The deadlock-free. is solution This
savages other waiting. is he While .fullPot for waits mutex holds that savage the
which .fullPot signal and run will cook the eventually But .mutex on queued are
mutex. the release and resume to savage waiting the allows
that guarantee it does or thread-safe. is pot the that assume solution this Does
exclusively? executed are getServicingFromPot and putServingsInPot

problem barbershop The ۲.۵

ap- it of variation A Dijkstra. by proposed was problem barbershop original The
 .[۱۰] *Concepts Systems Operating* Galvin's and Silberschatz in pears

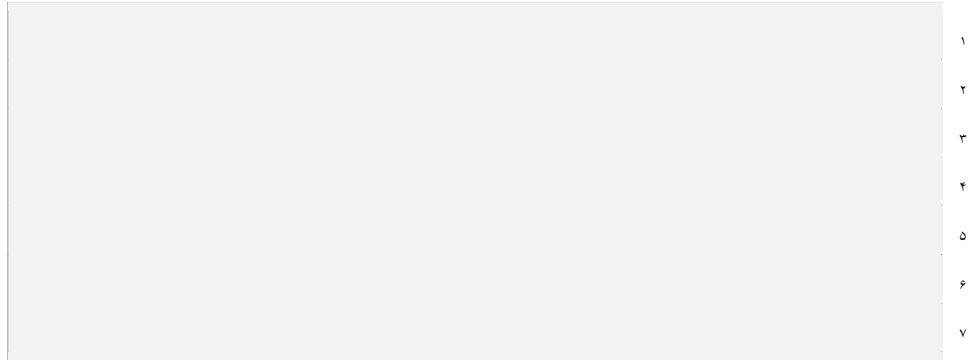
bar- the and chairs. n with room waiting a of consists barbershop A
 be to customers no are there If chair. barber the containing room ber
 and barbershop the enters customer a If sleep. to goes barber the served.
 barber the If shop. the leaves customer the then occupied. are chairs all
 free the of one in sits customer the then available. are chairs but busy. is
 Write barber. the up wakes customer the asleep. is barber the If chairs.
 customers. the and barber the coordinate to program a

information: following the added I concrete. more little a problem the make To

- `.getHairCut` named function a invoke should threads Customer
- which `.balk` invoke can it full. is shop the when arrives thread customer a If
 return. not does
- `.cutHair` invoke should thread barber The
- in- thread one exactly be should there `cutHair` invokes barber the When
 concurrently. `getHairCut` voking
- constraints. these guarantees that solution a Write

hint Barbershop ۱.۲.۵

hint Barbershop

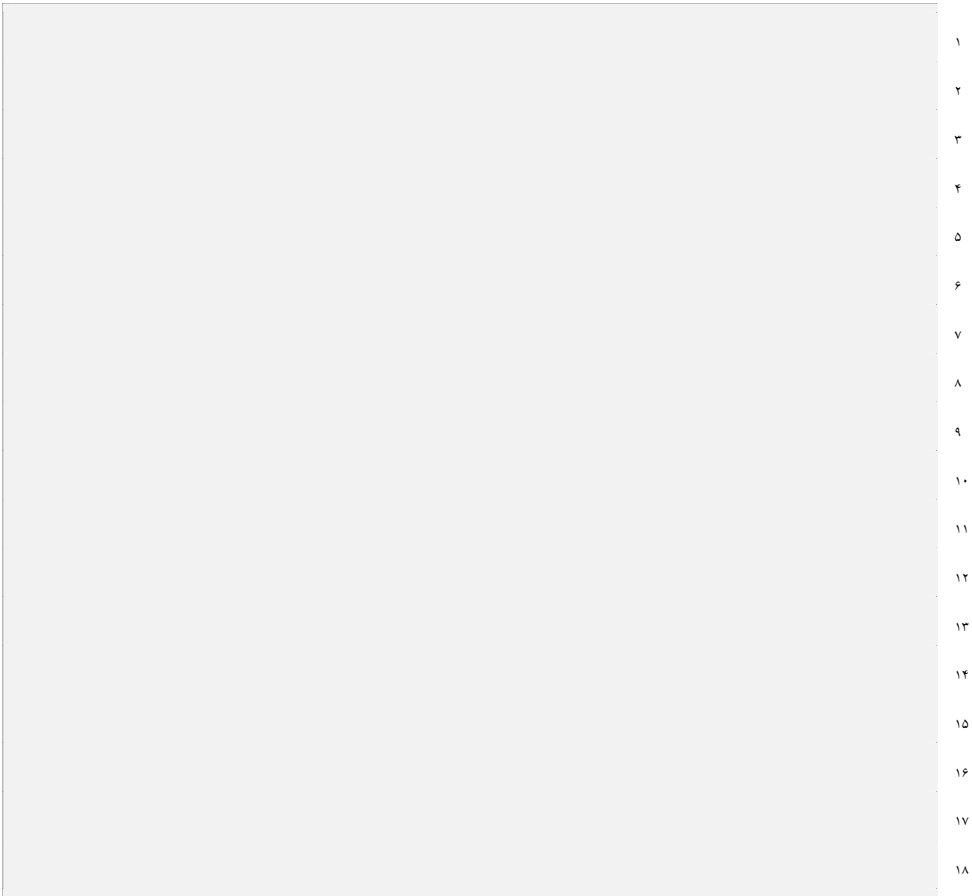


waiting the in three shop: the in be can that customers of number total the is n
 chair. the in one and room
 by protected is it shop: the in customers of number the counts customers
 .mutex
 cus- the then shop. the enters customer a until customer on waits barber The
 seat. a take to him signals barber the until barber on waits tomer
 on waits and customerDone signals customer the haircut. the After
 .barberDone

solution Barbershop ۲.۲.۵

for code the is Here . rendezvous two and scoreboard a combines solution This customers:

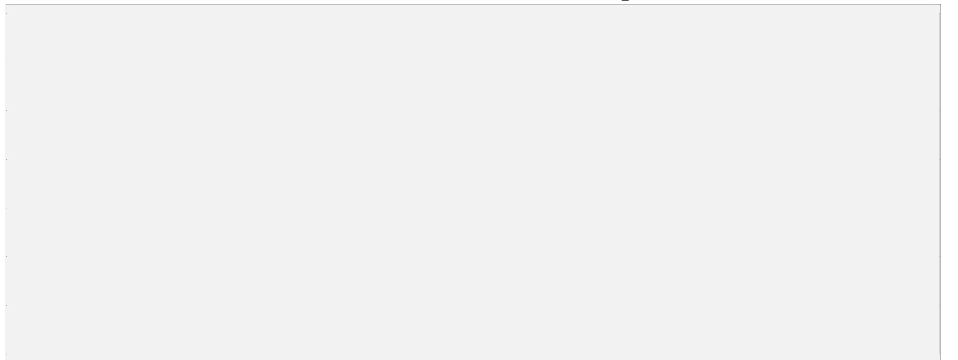
(customer) solution Barbershop



in- immediately arrive that customers any shop the in customers n are there If
.barber on waits and customer signals customer each Otherwise .balk voke
barbers: for code the is Here

possibility Another is. it what about agree dictionaries all not and rare. is rendezvous of plural The
pronounced. is “s” final the but “rendezvous.” spelled also is plural the that is

(barber) solution Barbershop



one gives and .barber signals wakes. barber the signals. customer a time Each
 next the on then busy. is barber the while arrives customer another If cut. hair
 sleeping. without semaphore customer the pass will barber the iteration
 for convention naming the on based are barber and customer for names The
 “customers not customer.” a for “wait means customer.wait() so rendezvous. a
 here.” wait
 that ensures .barberDone and customerDone using rendezvous. second The
 the into customer next the let to around loops barber the before done is cut hair the
 section. critical

.(2.3 (see sync_code/barber.py in is solution This

barbershop FIFO The ۳.۵

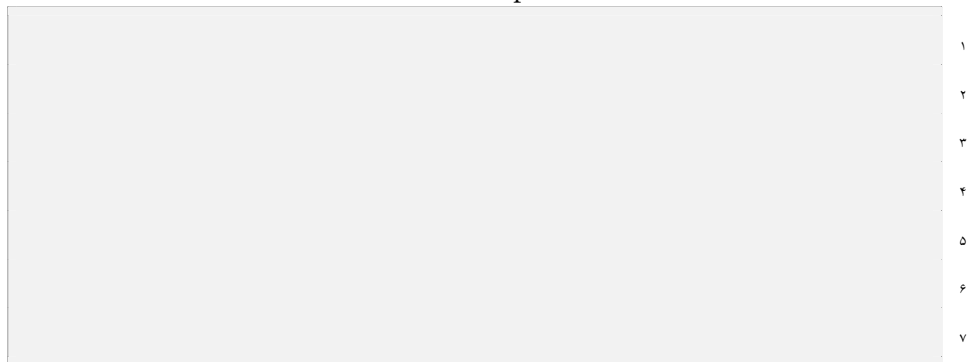
order the in served are customers that guarantee no is there solution previous the In
 on wait and customer signal turnstile. the pass can customers n to Up arrive. they
 proceed. might customers the of any ,barber signal barber the When .barber
 turn- the pass they order the in served are customers that so solution this Modify
 stile.

= self.sem write you if so ,self as thread current the to refer can you Hint:
 semaphore. own its gets thread each ,Semaphore(0)

hint barbershop FIFO ۱.۳.۵

.queue named semaphores of list a uses solution My

hint barbershop FIFO



queue. the in it puts and thread a creates it turnstile. the passes thread each As
 When semaphore. own its on waits thread each ,barber on waiting of Instead
 it. signals and queue the from thread a removes he up. wakes barber the

solution barbershop FIFO ۲.۳.۵

customers: for code modified the is Here

(customer) solution barbershop FIFO

۱

۲

۳

۴

۵

۶

۷

۸

۹

۱۰

۱۱

۱۲

۱۳

۱۴

۱۵

۱۶

۱۷

۱۸

۱۹

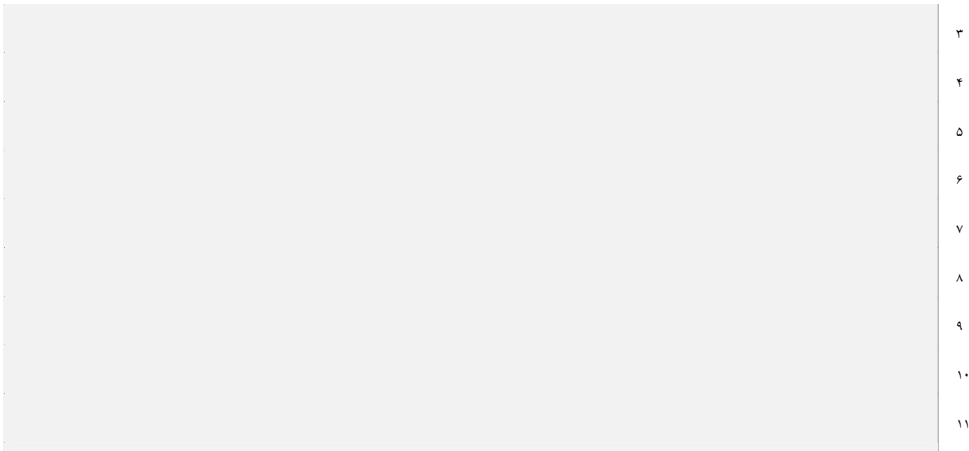
۲۰

barbers: for code the And

(barber) solution barbershop FIFO

۱

۲



queue. the access to mutex get to has barber the that Notice
 .(۲.۳ (see sync_code/barber2.py in is solution This

problem Barbershop Hilzer's 4.5

prob- barbershop the of version complicated more a presents [11] Stallings William Chico. at University State California the at Hilzer Ralph to attributes he which lem.

area waiting a and barbers. three chairs. three has barbershop Our standing has that and sofa a on customers four accommodate can that of number total the limit codes Fire customers. additional for room . 20 to shop the in customers other with capacity to filled is it if shop the enter not will customer A stands or sofa the on seat a takes customer the inside. Once customers. on been has that customer the free. is barber a When filled. is sofa the if customers. standing any are there if and. served is longest the sofa the sofa. the on seat a takes longest the shop the in been has that one the payment. accept can barber any finished. is haircut customer's a When one for accepted is payment register. cash one only is there because but hair. cutting among time their divide barbers The time. a at customer customer. a for waiting chair their in sleeping and payment. accepting

apply: constraints synchronization following the words. other In

.sitOnSofa, enterShop order: in functions following the invoke Customers •

.pay, getHairCut

.acceptPayment and cutHair invoke Barbers •

capacity. at is shop the if enterShop invoke cannot Customers •

.sitOnSofa invoke cannot customer arriving an full. is sofa the If •

bar- corresponding a be should there getHairCut invokes customer a When •

versa. vice and concurrently. cutHair executing ber

con- getHairCut execute to customers three to up for possible be should It •

concurrently. cutHair execute to barbers three to up and currently.

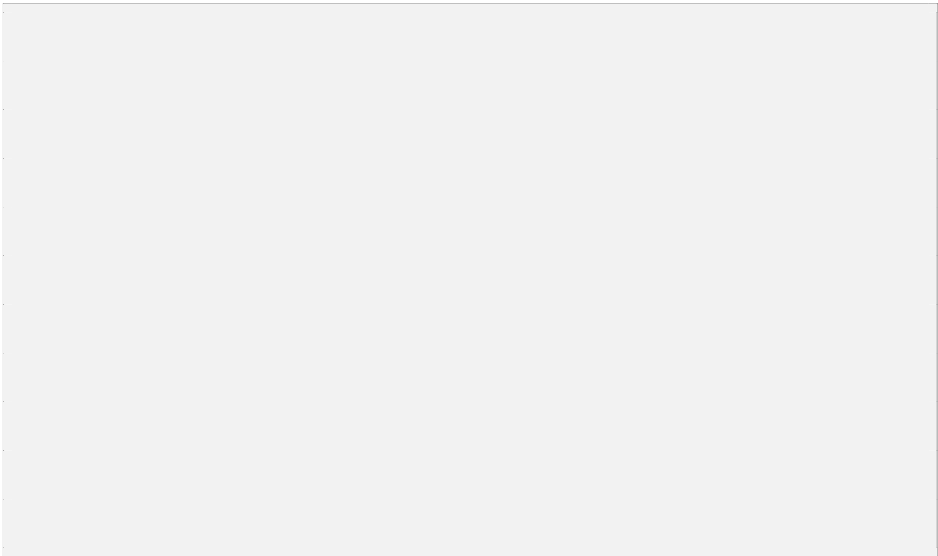
- The customer has to pay to the barber before .acceptPayment can
- The barber must acceptPayment before the customer can exit.

Write a puzzle: Hilzer's enforces that the synchronization constraints for Hilzer's barbershop.

1.4.5 Hilzer's barbershop hint

Here are the variables I used in my solution:

Hilzer's barbershop hint



mutex protects .customers which keeps track of the number of customers in the shop. the queue1 and list a is of semaphores for waiting threads on a seat sofa. the queue2 protects .customers which keeps track of the number of customers in the shop. the chair. the sofa. the chair.

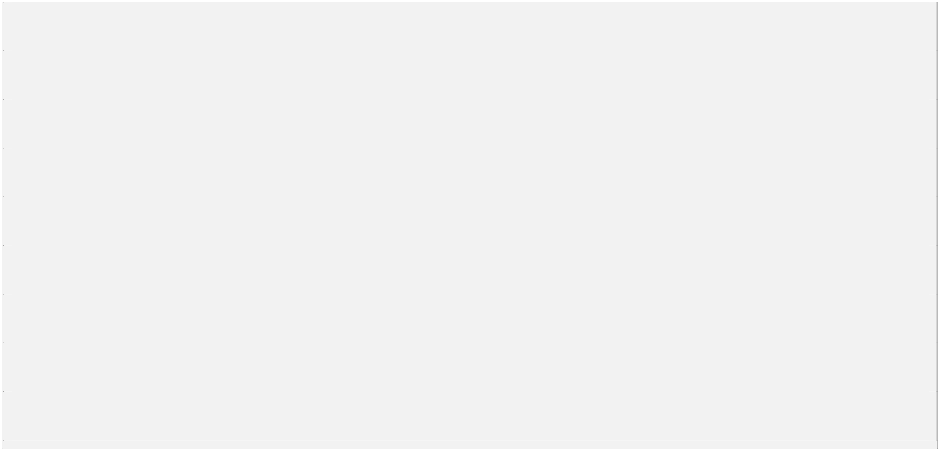
signals customer2 and .queue1 in customer a is there that signals customer1
 .queue2 in customer a is there that
 has barber a that signals receipt and paid. has customer a that signals payment
 payment. accepted

solution barbershop Hilzer’s ۲.۴.۵

had Hilzer if not am I expected. I than complex more considerably is solution This
do. could I best the is here but mind. in simpler something

(customer) solution barbershop Hilzer’s

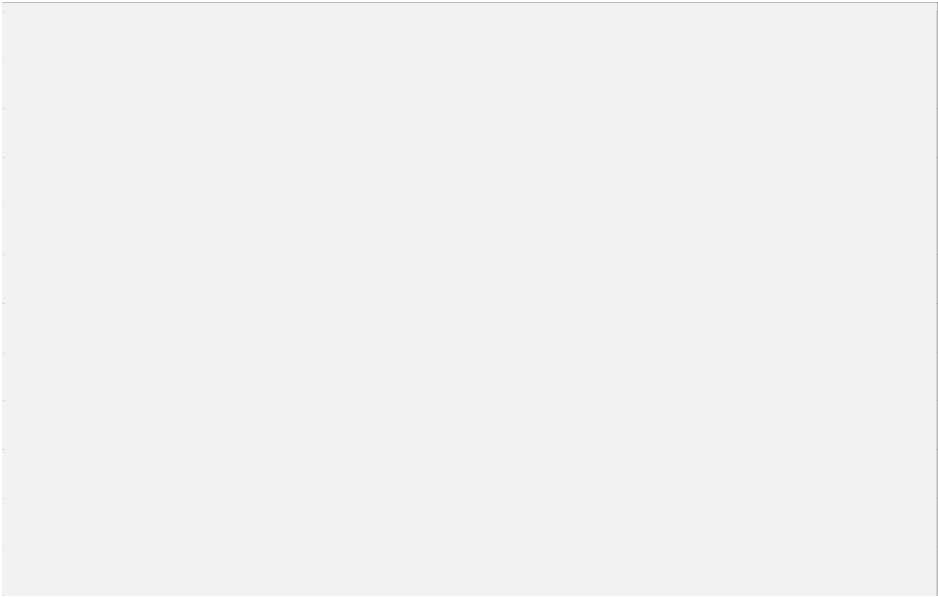
	۱
	۲
	۳
	۴
	۵
	۶
	۷
	۸
	۹
	۱۰
	۱۱
	۱۲
	۱۳
	۱۴
	۱۵
	۱۶
	۱۷
	۱۸
	۱۹
	۲۰
	۲۱
	۲۲
	۲۳
	۲۴
	۲۵



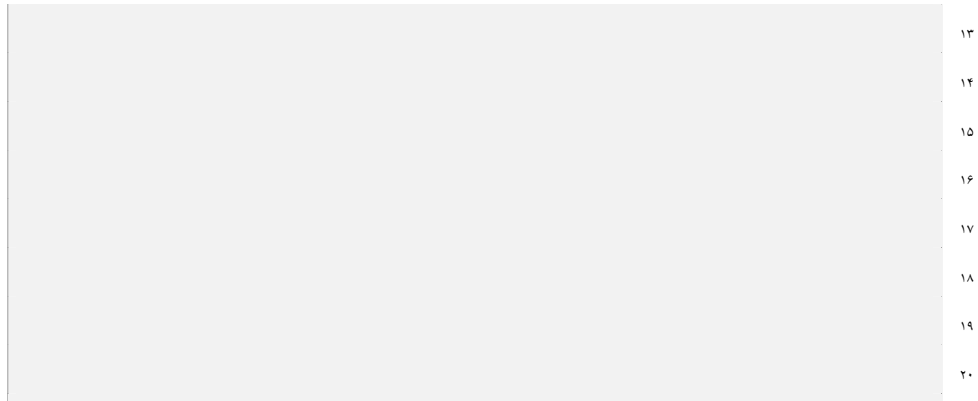
26
27
28
29
30
31
32
33
34

customer a When solution. previous the in as same the is paragraph first The
it Then queue. the to itself adds or balks either and counter the checks it arrives.
barber. a signals
couch the on sits multiplex. the enters it queue. of out gets customer the When
queue. second the to itself adds and
exits. then and pays. haircut. a gets it queue. that of out gets it When

(barber) solution barbershop Hilzer's



1
2
3
4
5
6
7
8
9
10
11
12



to semaphore customer's the signals enter. to customer a for waits barber Each
the enforces This sofa. the on seat a claim to it for waits then queue. of out it get
requirement. FIFO
it. signals then and queue second the join to customer the for waits barber The
chair. a claim to customer the allowing
con- three to up by can there so chair. the to customer one admits barber Each
get to has customer the register. cash one only is there Because haircuts. current
exit. both then register. cash the at rendezvous barber and customer The .mutex
un- sofa the leaves it but constraints. synchronization the satisfies solution This
than more be never can there barbers. three only are there Because derutilized.
unnecessary. is multiplex the so sofa. the on customers three
.(٢.٣ (see sync_code/barber3.py in is solution This
of kind third a create to is problem this solve to of think can I way only The
manage barbers the and queue1 manage ushers The usher. an can I which thread.
utilized. fully be can sofa the barbers. ٣ and ushers ٤ are there If .queue2
.(٢.٣ (see sync_code/barber4.py in is solution This

problem Claus Santa The 0.0

attributes he but `[\] Systems Operating Stallings's William from is problem This`
 Vermont. in College Michael's St. of Trono John to it

be only can and Pole North the at shop his in sleeps Claus Santa
 vacation their from back being reindeer nine all (\) either by awakened
 making difficulty having elves the of some (\) or Pacific. South the in
 him wake only can elves the sleep. some get to Santa allow to toys:
 their having are elves three When problems. have them of three when
 for wait must Santa visit to wishing elves other any solved. problems
 at waiting elves three find to up wakes Santa If return. to elves those
 the from back come having reindeer last the with along door. shop's his
 Christmas. after until wait can elves the that decided has Santa tropics.
 that assumed is (It ready. sleigh his get to important more is it because
 stay they therefore and tropics. the leave to want not do reindeer the
 must arrive to reindeer last The moment.) possible last the until there
 harnessed being before hut warming a in wait others the while Santa get
 sleigh. the to

specifications: addition some are Here

then and `.prepareSleigh` invoke must Santa arrives. reindeer ninth the After •
`.getHitched` invoke must reindeer nine all

all Concurrently. `.helpElves` invoke must Santa arrives. elf third the After •
`.getHelp` invoke should elves three

(in- enter elves additional any before `getHelp` invoke must elves three All •
 counter). elf the ccrement

assume can We elves. of sets many help can he so loop a in run should Santa
 elves. of number any be may there but reindeer. 9 exactly are there that

hint problem Santa 1.0.0

Santa problem hint

```
1 elves = 0
2 reindeer = 0
3 santaSem = Semaphore(0)
4 reindeerSem = Semaphore(0)
5 elfTex = Semaphore(1)
6 mutex = Semaphore(1)
```

reindeer and Elves .mutex by protected both counters: are reindeer and elves
them. check to it gets Santa counters: the modify to mutex get
him. signals reindeer a or elf an either until santaSem on waits Santa
pad- the enter to them signals Santa until reindeerSem on wait reindeer The
hitched. get and dock
elves three while entering from elves additional prevent to elfTex use elves The
helped. being are

solution problem Santa 1.0.0

loop. a in runs it that Remember straightforward. pretty is code Santa's

Santa problem solution (Santa)

```

1  santaSem.wait()
2  mutex.wait()
3      if reindeer >= 9:
4          prepareSleigh()
5          reindeerSem.signal(9)
6          reindeer -= 9
7      else if elves == 3:
8          helpElves()
9  mutex.signal()

```

ei- and holds conditions two the of which checks he up, wakes Santa When
wait- reindeer nine are there If elves. waiting the or reindeer the with deals ther
allow- times. nine reindeerSem signals then .prepareSleigh invokes Santa ing,
in- just Santa waiting, elves are there If .getHitched invoke to reindeer the ing
signal they once Santa! for wait to elves the for need no is There .helpElves vokes
immediately. getHelp invoke can they ,santaSem
on it do elves the because counter elves the decrement to have doesn't Santa
out. way their

reindeer: for code the is Here

Santa problem solution (reindeer)

```

1  mutex.wait()
2      reindeer += 1
3      if reindeer == 9:
4          santaSem.signal()
5  mutex.signal()
6
7  reindeerSem.wait()
8  getHitched()

```

on waiting reindeer other the joins then and Santa signals reindeer ninth The
.getHitched execute all reindeer the signals. Santa When .reindeerSem
sub- bar to has it arrives elf third the when that except similar, is code elf The
.getHelp executed have three first the until arrivals sequent

Santa problem solution (elves)

```
1 elfTex.wait()
2 mutex.wait()
3     elves += 1
4     if elves == 3:
5         santaSem.signal()
6     else
7         elfTex.signal()
8 mutex.signal()
9
10 getHelp()
11
12 mutex.wait()
13     elves -= 1
14     if elves == 0:
15         elfTex.signal()
16 mutex.signal()
```

but .mutex the release they time same the at elfTex release elves two first The
have elves three all until entering from elves other barring .elfTex holds elf last the
.getHelp invoked
enter. to elves of batch next the allowing .elfTex releases leave to elf last The

O₂H Building ٩.٥

Berkeley U.C. at class Systems Operating the of staple a been has problem This
Concurrent Andrews's in exercise an on based be to seems It decade. a least at for
 .[١] *Programming*

assemble to order In hydrogen. and oxygen threads. of kinds two are There
 each makes that barrier a create to have we molecules. water into threads these
 proceed. to ready is molecule complete a until wait thread
 guarantee must You .bond invoke should it barrier. the passes thread each As
 from threads the of any before bond invoke molecule one from threads the all that
 do. molecule next the
 words: other In

are threads hydrogen no when barrier the at arrives thread oxygen an If •
 threads. hydrogen two for wait to has it present.

present. are threads other no when barrier the at arrives thread hydrogen a If •
 thread. hydrogen another and thread oxygen an for wait to has it

the is. that explicitly: up threads the matching about worry to have don't We
 The with. up paired are they threads other which know necessarily not do threads
 the examine we if thus. sets: complete in barrier the pass threads that just is key
 each three. of groups into them divide and bond invoke that threads of sequence
 threads. hydrogen two and oxygen one contain should group
 that molecules hydrogen and oxygen for code synchronization Write Puzzle:
 constraints. these enforces

hint O₂H ۱.۶.۵

solution: my in used I variables the are Here

Water building hint

```

1 mutex = Semaphore(1)
2 oxygen = 0
3 hydrogen = 0
4 barrier = Barrier(3)
5 oxyQueue = Semaphore(0)
6 hydroQueue = Semaphore(0)

```

each where is barrier .mutex by protected counters, are hydrogen and oxygen
of set next the allowing before and bond invoking after meets threads three of set
proceed. to threads
the is hydroQueue on: wait threads oxygen semaphore the is oxyQueue
conven- naming the using am I on. wait threads hydrogen semaphore
and queue” oxygen the “join means oxyQueue.wait() so queues. for tion
queue.” the from thread oxygen an “release means oxyQueue.signal()

solution O₂H ۲.۶.۵

it arrives thread oxygen an When locked. are oxyQueue and hydroQueue Initially
oxygen the Then proceed. to hydrogens two allowing twice. hydroQueue signals
arrive. to threads hydrogen the for waits thread

Oxygen code

```

1 mutex.wait()
2 oxygen += 1
3 if hydrogen >= 2:
4     hydroQueue.signal(2)
5     hydrogen -= 2
6     oxyQueue.signal()
7     oxygen -= 1
8 else:
9     mutex.signal()
10
11 oxyQueue.wait()
12 bond()
13
14 barrier.wait()
15 mutex.signal()

```

If scoreboard. the checks and mutex the gets it enters. thread oxygen each As
and itself and them of two signals it waiting. threads hydrogen two least at are there
waits. and mutex the releases it not. If bonds. then
bonded. have threads three all until barrier the at wait threads bonding. After
oxygen one only is there Since mutex. the releases thread oxygen the then and
once. mutex signal to guaranteed are we three. of set each in thread
similar: is hydrogen for code The

Hydrogen code

```

1 mutex.wait()
2 hydrogen += 1
3 if hydrogen >= 2 and oxygen >= 1:
4     hydroQueue.signal(2)
5     hydrogen -= 2
6     oxyQueue.signal()
7     oxygen -= 1
8 else:
9     mutex.signal()

```

```

10
11 hydroQueue.wait()
12 bond()
13
14 barrier.wait()

```

am- is mutex the of point exit the that is solution this of feature unusual An the exit and counter, the update mutex, the enter threads cases, some In biguous. mutex the keep to has it set, complete a forms that arrives thread a when But mutex.

.bond invoked have set current the until threads subsequent bar to order in opens, barrier the When barrier, a at wait threads three the .bond invoking After the holds them of one that and bond invoked have threads three all that know we long as matter doesn't it but mutex, the holds thread *which* know don't We mutex, we thread, oxygen one only is there know we Since it, releases them of one only as work, the do it make

a that true been generally has it now until because wrong, seem might This has that says that rule no is there But it, release to order in lock a hold to has thread mutex a of think to misleading be can it where cases those of one is This true, be to release, and acquire threads that token a as

problem crossing River V.5

don't I but Berkeley, U.C. at Joseph Anthony by written set problem a from is This that sense the in problem O_YH the to similar is It author, original the is he if know combinations, certain in pass to threads allows only that barrier of sort peculiar a is it both by used is that rowboat a is there Washington Redmond, near Somewhere hold can ferry The river, a cross to (serfs) employees Microsoft and hackers Linux the guarantee To fewer, or more with shore the leave won't it people, four exactly three with boat the in hacker one put to permissible not is it passengers, the of safety safe, is combination other Any hackers, three with serf one put to or serfs. You .board called function a invoke should it boat the boards thread each As of any before board invoke boatload each from threads four all that guarantee must

do. boatload next the from threads the
a call should them of one exactly .board invoked have threads four all After
doesn't It oars. the take will thread that that indicating .rowBoat named function
does. one as long as function. the calls thread which matter
in interested only are we Assume travel. of direction the about worry Don't
directions. the of one in going traffic

hint crossing River 1.V.5

solution: my in used I variables the are Here

River crossing hint

```

1 barrier = Barrier(4)
2 mutex = Semaphore(1)
3 hackers = 0
4 serfs = 0
5 hackerQueue = Semaphore(0)
6 serfQueue = Semaphore(0)
7 local isCaptain = False

```

board. to waiting serfs and hackers of number the count `serfs` and `hackers`
 vari- both of condition the check can we `mutex` by protected both are they Since
 a of example another is This update. untimely an about worrying without ables
 scoreboard.

and hackers of number the control to us allow `serfQueue` and `hackerQueue`
 board invoked have threads four all that sure makes barrier The pass. that serfs
`.rowBoat` invokes captain the before
`.row` invoke should thread which indicates that variable local a is `isCaptain`

solution crossing River V.V.5

and counters the of one updates arrival each that is solution this of idea basic The
 its of fourth the being by either complement, full a makes it whether checks then
 pairs, of pair mixed a completing by or kind
 course, of (except, symmetric is code serf the hackers; for code the present I'll
 browser): web embedded an contains it and bugs, of full bigger, times \dots is it that

River crossing solution

```

1 mutex.wait()
2     hackers += 1
3     if hackers == 4:
4         hackerQueue.signal(4)
5         hackers = 0
6         isCaptain = True
7     elif hackers == 2 and serfs >= 2:
8         hackerQueue.signal(2)
9         serfQueue.signal(2)
10        serfs -= 2
11        hackers = 0
12        isCaptain = True
13    else:
14        mutex.signal()      # captain keeps the mutex
15
16 hackerQueue.wait()
17
18 board()
19 barrier.wait()
20
21 if isCaptain:
22     rowBoat()
23     mutex.signal()      # captain releases the mutex

```

a whether checks it section, exclusion mutual the through files thread each As
 declares threads, appropriate the signals it so, If board, to ready is crew complete
 boat the until threads additional bar to order in mutex the holds and captain, itself
 sailed, has
 last the When boarded, have threads many how of track keeps barrier The
 re- (finally) then and row invoked captain The proceed, threads all arrives, thread
 mutex, the leases

problem coaster roller The 8.5

it attributes he but, [1] *Programming Concurrent* Andrews's from is problem This
thesis. Master's Herman's S. J. to

pas- The thread. car a and threads passenger n are there Suppose
pas- C hold can which car. the in rides take to wait repeatedly sengers
it when only tracks the around go can car The $C < n$ where sengers.
full. is

details: additional some are Here

- .unboard and board invoke should Passengers
- .unload and run .load invoke should car The
- load invoked has car the until board cannot Passengers
- boarded. have passengers C until depart cannot car The
- .unload invoked has car the until unboard cannot Passengers

constraints. these enforces that car and passengers the for code Write Puzzle:

hint Coaster Roller ١.٨.٥

Roller Coaster hint

```

1 mutex = Semaphore(1)
2 mutex2 = Semaphore(1)
3 boarders = 0
4 unboarders = 0
5 boardQueue = Semaphore(0)
6 unboardQueue = Semaphore(0)
7 allAboard = Semaphore(0)
8 allAshore = Semaphore(0)

```

have that passengers of number the counts which ,passengers protects mutex

.boardCar invoked

un- before unboardQueue and boarding before boardQueue on wait Passengers

full. is car the that indicates allAboard boarding.

solution Coaster Roller 7.8.5

thread: car the for code my is Here

Roller Coaster solution (car)

```

1 load()
2 boardQueue.signal(C)
3 allAboard.wait()
4
5 run()
6
7 unload()
8 unboardQueue.signal(C)
9 allAshore.wait()

```

signal to one last the for waits then passengers. C signals it arrives. car the When
for waits then disembark. to passengers C allows it departs. it After .allAboard
.allAshore

Roller Coaster solution (passenger)

```

1 boardQueue.wait()
2 board()
3
4 mutex.wait()
5     boarders += 1
6     if boarders == C:
7         allAboard.signal()
8         boarders = 0
9 mutex.signal()
10
11 unboardQueue.wait()
12 unboard()
13
14 mutex2.wait()
15     unboarders += 1
16     if unboarders == C:
17         allAshore.signal()
18         unboarders = 0
19 mutex2.signal()

```

stop to car the for wait and naturally. boarding. before car the for wait Passengers
passenger the resets and car the signals board to passenger last The leaving. before
counter.

problem Coaster Roller Multi-car 3.8.5

In car, one than more is there where case the to generalize not does solution This
constraints: additional some satisfy to have we that, do to order

time, a at boarding be can car one Only •

concurrently, track the on be can cars Multiple •

they order same the in unload to have they other, each pass can't cars Since •
boarded.

threads the of any before disembark must carload one from threads the All •
carloads, subsequent from

You constraints, additional the handle to solution previous the modify Puzzle:

that i named variable local a has car each that and cars, m are there that assume can
. $m - \backslash$ and \cdot between identifier an contains

hint Coaster Roller Multi-car 4.8.5

loading the represents One order. in cars the keep to semaphores of lists two used I
 for semaphore one contains list Each area. unloading the represents one and area
 enforces that so unlocked. is list each in semaphore one only time. any At car. each
 • Car for semaphores the only Initially. unload. and load can threads order the
 own its on waits it unloading) (or loading the enters car each As unlocked. are
 line. in car next the signals it leaves it as semaphore:

Multi-car Roller Coaster hint

```

1 loadingArea = [Semaphore(0) for i in range(m)]
2 loadingArea[1].signal()
3 unloadingArea = [Semaphore(0) for i in range(m)]
4 unloadingArea[1].signal()

```

sequence the in car next the of identifier the computes next function The
 :(\bullet to $m - 1$ from around (wrapping

Implementation of next

```

1 def next(i):
2     return (i + 1) % m

```


solution Coaster Roller Multi-car ٥.٨.٥

cars: the for code modified the is Here

Multi-car Roller Coaster solution (car)

```
1 loadingArea[i].wait()
2 load()
3 boardQueue.signal(C)
4 allAboard.wait()
5 loadingArea[next(i)].signal()
6
7 run()
8
9 unloadingArea[i].wait()
10 unload()
11 unboardQueue.signal(C)
12 allAshore.wait()
13 unloadingArea[next(i)].signal()
```

unchanged. is passengers the for code The

problems Not-so-classical

problem search-insert-delete The 1.6

. [1] *Programming Concurrent* Andrews's from is one This

searchers. list: singly-linked a to access share threads of kinds Three
can they hence list: the examine merely Searchers deleters. and inserters
the to items new add Inserters other. each with concurrently execute
two preclude to exclusive mutually be must insertions list: the of end
However, time. same the about at items new inserting from inserters
Finally, searches. of number any with parallel in proceed can insert one
deleter one most At list. the in anywhere from items remove deleters
mutually be also must deletion and time. a at list the access can process
insertions. and searches with exclusive

of kind this enforces that deleters and inserters searchers, for code write Puzzle:

exclusion. mutual categorical three-way

hint Search-Insert-Delete 1.1.6

Search-Insert-Delete hint

```

1 insertMutex = Semaphore(1)
2 noSearcher = Semaphore(1)
3 noInserter = Semaphore(1)
4 searchSwitch = Lightswitch()
5 insertSwitch = Lightswitch()

```

time. a at section critical its in is inserter one only that ensures insertMutex
no and searchers no are there that (surprise) indicate noInserter and noSearcher
enter. to these of both hold to needs deleter a sections: critical their in inserters
exclude to inserters and searchers by used are insertSwitch and searchSwitch
deleters.

solution Search-Insert-Delete 2.1.6

solution: my is Here

Search-Insert-Delete solution (searcher)

```

1 searchSwitch.wait(noSearcher)
2 # critical section
3 searchSwitch.signal(noSearcher)

```

searcher first The deleter. a is about worry to needs searcher a thing only The
it. releases out one last the 'noSearcher takes in

Search-Insert-Delete solution (inserter)

```

1 insertSwitch.wait(noInserter)
2 insertMutex.wait()
3 # critical section
4 insertMutex.signal()
5 insertSwitch.signal(noInserter)

```

it. releases out one last the and noInserter takes inserter first the Similarly.
their in be can they semaphores. different for compete inserters and searchers Since
in is inserter one only that ensures insertMutex But concurrently. section critical
time. a at room the

Search-Insert-Delete solution (deleter)

```

1 noSearcher.wait()
2 noInserter.wait()
3 # critical section
4 noInserter.signal()
5 noSearcher.signal()

```

guaranteed is it noInserter and noSearcher both holds deleter the Since
one than more holding thread a see we time any course. Of access. exclusive
you scenarios. few a out trying By deadlocks. for check to need we semaphore.
free. deadlock is solution this that yourself convince to able be should
prone is one this problems. exclusion categorical many like hand. other the On
sometimes can we problem. Readers-Writers the in saw we As starvation. to
to according threads of category one to priority giving by problem this mitigate

solu-efficient an write to difficult is it general in But criteria, application-specific starvation, avoids that concurrency) of degree maximum the allows that (one tion

problem bathroom unisex The 2.6

at physics teaching position her left mine of friend a when \problem this wrote I
 Xerox, at job a took and College Colby
 the and monolith, concrete a of basement the in cubicle a in working was She
 that Uberboss the to proposed She up, floors two was bathroom women's nearest
 on like of sort bathroom, unisex a to floor her on bathroom men's the convert they
 McBeal, Ally
 constraints synchronization following the that provided agreed, Uberboss The
 maintained: be can

time, same the at bathroom the in women and men be cannot There •
 company squandering employees three than more be never should There •
 bathroom, the in time

worry don't though, now, For deadlock, avoid should solution the course Of
 the all with equipped is bathroom the that assume may You starvation, about
 need, you semaphores

[1] *Programming Concurrent* Andrews's in appears problem identical nearly a that learned I Later

hint bathroom Unisex 1.2.6

solution: my in used I variables the are Here

Unisex bathroom hint

```

1 empty = Semaphore(1)
2 maleSwitch = Lightswitch()
3 femaleSwitch = Lightswitch()
4 maleMultiplex = Semaphore(3)
5 femaleMultiplex = Semaphore(3)

```

otherwise. • and empty is room the if \ is empty
 en- male first the When room. the from women bar to men allows maleSwitch
 un- it exits. male last the When women: barring ,empty locks lightswitch the ters.
 .femaleSwitch using likewise do Women enter. to women allowing ,empty locks
 than more no are there that ensure femaleMultiplex and maleMultiplex
 time. a at system the in women three and men three

solution bathroom Unisex ۲.۲.۶

code: female the is Here

Unisex bathroom solution (female)

```
1 femaleSwitch.lock(empty)
2     femaleMultiplex.wait()
3     # bathroom code here
4     femaleMultiplex.signal()
5 female Switch.unlock(empty)
```

similar. is code male The

solution? this with problems any there Are

problem bathroom unisex No-starve ۳.۲.۶

of line long A starvation. allows it that is solution previous the with problem The
 versa. vice and waiting. man a is there while enter and arrive can women
 problem. the fix Puzzle:

solution bathroom unisex No-starve 4.2.6

stop to thread of kind one allow to turnstile a use can we before, seen have we As
code: male the at look we'll time This thread. of kind other the of flow the

No-starve unisex bathroom solution (male)

```

1 turnstile.wait()
2   maleSwitch.lock(empty)
3 turnstile.signal()
4
5   maleMultiplex.wait()
6   # bathroom code here
7   maleMultiplex.signal()
8
9 maleSwitch.unlock (empty)
```

turnstile the through pass will arrivals new room. the in men are there as long As
block will male the arrives, male a when room the in women are there If enter. and
entering from female) and (male arrivals later all bar will which turnstile. the inside
enters, turnstile the in male the point that At leave. occupants current the until
enter. to males additional allowing possibly
female arriving an room the in men are there if so similar. is code female The

men. additional barring turnstile. the in stuck get will
often will there then busy. is system the If efficient. be not may solution This
empty time Each turnstile. the on queued female. and male threads. several be
new the If enter. will another and turnstile the leave will thread one signaled. is
threads. additional barring block. promptly will it gender. opposite the is thread
the and time. a at bathroom the in threads 2-1 only be usually will there Thus.
concurrency. available the of advantage full take not will system

problem crossing Baboon 3.6

Im- and Design Systems: Operating Tanenbaum's from adapted is problem This
Park. National Kruger in somewhere canyon deep a is There .[12] plementation
canyon the cross can Baboons canyon. the spans that rope single a and Africa. South

opposite in going baboons two if but rope, the on hand-over-hand swinging by
Further- deaths, their to drop and fight will they middle, the in meet directions
baboons more are there If baboons, Δ hold to enough strong only is rope the more,

break, will it time, same the at rope the on
to like would we semaphores, use to baboons the teach can we that Assuming
properties: following the with scheme synchronization a design

side other the to get to guaranteed is it cross, to begun has baboon a Once •
way, other the going baboon a into running without

rope, the on baboons Δ than more never are There •

ba- bar not should direction one in crossing baboons of stream continuing A •
starvation). (no indefinitely way other the going boons

clear, be should that reasons for problem this to solution a include not will I

Problem Hall Modus The 4.6

in living students Olin the of one Karst, Nathan by written was problem This
.2005 of winter the during ^YHall Modus

Modus of denizens the winter, this snowfall heavy particularly a After
shantytown cardboard their between path trench-like a created Hall
and to walk residents the of some day Every campus, of rest the and
indo- the ignore will we path, the via civilization and food class, from
the ignore also will We .3 Tier to drive to daily chose who students lent
rea- unknown some For traveling, are pedestrians which in direction
necessary it find occasionally would Hall West in living students son,

Mods, the to venture to

students some that Mods, aka buildings, modular the for nicknames several of one is Hall Modus^Y
built, being was hall residence second the while in lived

to people two allow to enough wide not is path the Unfortunately,
 the on point some at meet persons Mods two If side-by-side, walk
 accommodate to drift high neck the into aside step gladly will one path,
 cross inhabitants ResHall two if occur will situation similar A other, the
 violent a however, meet, prude ResHall a and heathen Mods a If paths,
 of strength by solely determined victors the with ensue will skirmish
 the force will population larger the with faction the is, that numbers:
 wait, to other

with one), than ways more (in problem Crossing Baboon the to similar is This
 rule, majority by determined is section critical the of control that twist added the
 categor- the to solution starvation-free and efficient an be to potential the has This
 problem, exclusion ical
 section, critical the controls faction one while because avoided is Starvation
 majority, a achieve they until queue in accumulate faction other the of members
 critical the for wait they while entering from opponents new bar can they Then
 move to tend will it because efficient be to solution this expect I clear, to section
 section, critical the in concurrency maximum allowing batches, in through threads
 rule, majority with exclusion categorical implements that code write Puzzle:

hint problem Hall Modus ۱.۴.۶

solution. my in used I variables the are Here

Modus problem hint

```

1 heathens = 0
2 prudes = 0
3 status = 'neutral'
4 mutex = Semaphore(1)
5 heathenTurn = Semaphore(1)
6 prudeTurn = Semaphore(1)
7 heathenQueue = Semaphore(0)
8 prudeQueue = Semaphore(0)

```

field. the of status the records status and counters. are prudes and heathens
or heathens' to 'transition rule', 'prudes rule', 'heathens 'neutral', be can which
pat- scoreboard usual the in mutex by protected are three All prudes'. to 'transition
tern.

one bar can we that so field the to access control prudeTurn and heathenTurn

transition. a during other the or side

and in checking after wait threads where are prudeQueue and heathenQueue

field. the taking before

solution problem Hall Modus ۲.۴.۶

heathens: for code the is Here

Modus problem solution

```
1 heathenTurn.wait()
2 heathenTurn.signal()
3
4 mutex.wait()
5 heathens++
6
7 if status == 'neutral':
8     status = 'heathens rule'
9     mutex.signal()
10 elif status == 'prudes rule':
11     if heathens > prudes:
12         status = 'transition to heathens'
13         prudeTurn.wait()
14     mutex.signal()
15     heathenQueue.wait()
16 elif status == 'transition to heathens':
17     mutex.signal()
18     heathenQueue.wait()
19 else
20     mutex.signal()
21
22 # cross the field
23
24 mutex.wait()
25 heathens--
26
27 if heathens == 0:
28     if status == 'transition to prudes':
29         prudeTurn.signal()
30     if prudes:
31         prudeQueue.signal(prudes)
32         status = 'prudes rule'
33     else:
34         status = 'neutral'
35
36 if status == 'heathens rule':
37     if prudes > heathens:
38         status = 'transition to prudes'
39         heathenTurn.wait()
40
41 mutex.signal()
```

cases: following the consider to has he in. checks student each As

heathens. the for claim lays student the empty. is field the If ●

balance. the tipped has arrival new the but charge. in currently heathens the If ●

mode. transition to switches system the and turnstile prude the locks he

joins he balance. the tip doesn't arrival new the but charge. in prudes the If ●

queue. the

the joins arrival new the control. heathen to transitioning is system the If ●

queue.

system the or charge. in are heathens the either that conclude we Otherwise ●

proceed. can thread this case. either In control. prude to transitioning is

cases. several consider to has she out. checks student each as Similarly.

following: the consider to has she out. check to heathen last the is she If ●

locked. is turnstile prude the that means that transition. in is system the If —

it. open to has she so

the so status updates and them signals she waiting. prudes are there If —

'neutral'. is status new the not. If charge. in are prudes

possibility the check to has still she out. check to heathen last the not is she If ●

heathen the closes she case. that In balance. the tip will departure her that

transition. the starts and turnstile

could threads of number any that is solution this of difficulty potential One

yet not but turnstile the passed have would they where ,۳ Line at interrupted be

power of balance the so counted. not are they in. check they Until in. checked

transi- a Also. turnstile. the passed have that threads of number the reflect not may

that At out. checked also have in checked have that threads the all when ends tion

turnstile. the passed have that types) both (of threads be may there point.

max- guarantee not does solution efficiency—this affect may behaviors These
“majority that accept you if correctness. affect don’t they concurrency—but inum
vote. to registered have that threads to applies only rule”

فصل ٧

classical remotely Not problems

problem bar sushi The ١.٧

Imagine .[٩] Reek Kenneth by proposed problem a by inspired was problem This
seat a take can you seat, empty an is there while arrive you If seats. Δ with bar sushi a
them of all that means that full, are seats Δ all when arrive you if But immediately.
before leave to party entire the for wait to have will you and together, dining are
down, sit you
enforces that bar sushi the leaving and entering customers for code write Puzzle:
requirements, these

hint bar Sushi ۱.۱.۷

used: I variables the are Here

Sushi bar hint

```

1 eating = waiting = 0
2 mutex = Semaphore(1)
3 block = Semaphore(0)
4 must_wait = False

```

and bar the at sitting threads of number the of track keep waiting and eating
 has (or is bar the that indicates must_wait counters. both protects mutex waiting.
 .block on block to have customers incoming som full, been)

non-solution bar Sushi 2.1.7

this of difficulties the of one illustrate to uses Reek solution incorrect an is Here
problem.

Sushi bar non-solution

```

1 mutex.wait()
2 if must_wait:
3     waiting += 1
4     mutex.signal()
5     block.wait()
6
7     mutex.wait()      # reacquire mutex
8     waiting -= 1
9
10 eating += 1
11 must_wait = (eating == 5)
12 mutex.signal()
13
14 # eat sushi
15
16 mutex.wait()
17 eating -= 1
18 if eating == 0:
19     n = min(5, waiting)
20     block.signal(n)
21     must_wait = False
22 mutex.signal()

```

solution? this with wrong what's Puzzle:

non-solution bar Sushi ३.१.१

up give to has he full. is bar the while arrives customer a If . १ Line at is problem The
 customer last the When leave. can customers other that so waits he while mutex the
 customers. waiting the of some least at up wakes which ,block signals she leaves.
 .must_wait clears and

that and back. mutex the get to have they up. wake customers the when But
 and arrive threads new If threads. new incoming with compete to have they means
 is This threads. waiting the before seats the all take could they first. mutex the get
 the in be to threads १ than more for possible is it injustice. of question a just not
 constraints. synchronization the violates which concurrently. section critical
 two next the in appear which problem. this to solutions two provides Reek
 sections.

solutions! correct different two with up come can you if see Puzzle:

variables. additional any uses solution neither Hint:

#1 solution bar Sushi 4.1.7

state the update to is mutex the reacquire to has customer waiting a reason only The
departing the make to is problem the solve to way one so .waiting and eating of
updating. the do mutex. the has already who customer.

Sushi bar solution #1

```

1 mutex.wait()
2 if must_wait:
3     waiting += 1
4     mutex.signal()
5     block.wait()
6 else:
7     eating += 1
8     must_wait = (eating == 5)
9     mutex.signal()
10
11 # eat sushi
12
13 mutex.wait()
14 eating -= 1
15 if eating == 0:
16     n = min(5, waiting)
17     waiting -= n
18     eating += n
19     must_wait = (eating == 5)
20     block.signal(n)
21 mutex.signal()

```

been already has eating mutex. the releases customer departing last the When
necessary. if block and state right the see customers arriving newly so updated.
doing is thread departing the because you.” for it do “I’ll pattern this calls Reek
threads. waiting the to belong to logically. seems. that work
the that confirm to difficult more little a it is that is approach this of drawback A
correctly. updated being is state

#2 solution bar Sushi 0.1.7

can we that notion counterintuitive the on based is solution alternative Reek's acquire can thread one words. other In another! to thread one from mutex a transfer understand threads both as long As it. release can thread another then and lock a this. with wrong nothing is there transferred. been has lock the that

Sushi bar solution #2

```

1 mutex.wait()
2 if must_wait:
3     waiting += 1
4     mutex.signal()
5     block.wait()      # when we resume, we have the mutex
6     waiting -= 1
7
8 eating += 1
9 must_wait = (eating == 5)
10 if waiting and not must_wait:
11     block.signal()    # and pass the mutex
12 else:
13     mutex.signal()
14
15 # eat sushi
16
17 mutex.wait()
18 eating -= 1
19 if eating == 0: must_wait = False
20
21 if waiting and not must_wait:
22     block.signal()    # and pass the mutex
23 else:
24     mutex.signal()

```

entering an waiting. one no and bar the at customers 0 than fewer are there If sets customer fifth The mutex. the releases and eating increments just customer .must_wait

bar the at customer last the until block customers entering set. is must_wait If gives thread signaling the that understood is It .block signals and must_wait clears this that though. mind. in Keep it. receives thread waiting the and mutex the up comments. the in documented and programmer. the by understood invariant an is

right. it get to us to up is It semaphores. of semantics the by enforced not but there If mutex. the has it that understand we resumes. thread waiting the When wait- a to mutex the passes again. which. block signals it waiting. threads other are next the to mutex the passing thread each with continues. process This thread. ing last the case. either In threads. waiting more no or chairs more no are there until down. sit to goes and mutex the releases thread one from passed being is mutex the since baton.” the “Pass pattern this calls Reek solution this about thing nice One race. relay a in baton a like next the to thread A consistent. are waiting and eating to updates that confirm to easy is it that is correctly. used being is mutex the that confirm to harder is it that is drawback

problem care child The 2.7

Mid- and Systems Operating textbook his for problem this wrote Hailperin Max one always is there that require regulations state center. care child a At .[5] dleware children. three every for present adult con- this enforces that threads adult and threads child for code Write Puzzle: section. critical a in straint

hint care Child 7.7.7

semaphore. one with problem this solve *almost* can you that suggests Hailperin

Child care hint

```
1 multiplex = Semaphore(0)
```

a allows token each where available. tokens of number the counts multiplex
 they as times: three multiplex signal they enter. adults As enter. to thread child
 solution. this with problem a is there But times. three wait they leave.
 problem? the is what Puzzle:

non-solution care Child ۲.۲.۷

non-solution: Hailperin's in like looks code adult the what is Here

Child care non-solution (adult)

```

1 multiplex.signal(3)
2
3 # critical section
4
5 multiplex.wait()
6 multiplex.wait()
7 multiplex.wait()

```

and children three are there that Imagine deadlock. potential a is problem The
 adult either so ,۳ is multiplex of value The center. care child the in adults two
 they time. same the at leave to start adults both if But leave. to able be should
 block. both and them. between tokens available the divide might
 change. minimal a with problem this solve Puzzle:

solution care Child ۳.۲.۷

problem: the solves mutex a Adding

Child care solution (adult)

```

1 multiplex.signal(3)
2
3 # critical section
4
5 mutex.wait()
6     multiplex.wait()
7     multiplex.wait()
8     multiplex.wait()
9 mutex.signal()
```

available. tokens three are there If atomic. are operations wait three the Now
fewer are there If exit. and tokens three all get will mutex the gets that thread the
will threads subsequent and mutex the in block will thread first the available. tokens
mutex. the on queue

problem care child Extended ۴.۲.۷

child prevent can leave to waiting thread adult an that is solution this of feature One
entering. from threads
is multiplex the of value the so adults. two and children ۴ are there that Imagine
waiting block then and tokens two take will she leave. to tries adults the of one If . ۲
to legal be would it though even wait will it arrives. thread child a If third. the for
fine. just be might that leave. to trying adult the of view of point the From enter.
not. it's center. care child the of utilization the maximize to trying are you if but
waiting. unnecessary avoids that problem this to solution a write Puzzle:

.۸.۳ Section in dancers the about think Hint:

hint care child Extended ۵.۲.۷

solution: my in used I variables the are Here

Extended child care hint

```

1 children = adults = waiting = leaving = 0
2 mutex = Semaphore(1)
3 childQueue = Semaphore(0)
4 adultQueue = Semaphore(0)

```

chil- of number the of track keep leaving and waiting ,adults ,children
 pro- are they leave: to waiting adults and enter: to waiting children adults, dren,
 .mutex by tected
 on wait Adults necessary. if enter: to childQueue on wait Children
 leave: to adultQueue

solution care child Extended ۶.۲.۷

mostly is it but solution. elegant Hailperin's than complicated more is solution This
 “I'll and queues, two scoreboard, a before: seen have we patterns of combination a
 you”. for it do
 code: child the is Here

Extended child care solution (child)

```

1 mutex.wait()
2     if children < 3 * adults:
3         children++
4         mutex.signal()
5     else:
6         waiting++
7         mutex.signal()
8         childQueue.wait()
9
10 # critical section
11
12 mutex.wait()
13     children--
14     if leaving and children <= 3 * (adults-1):
15         leaving--
16         adults--
17         adultQueue.signal()
18 mutex.signal()

```

(۱) either and adults enough are there whether check they enter. children As
 they When block. and waiting increment (۲) or enter and children increment
 possible. if it signal and leave to waiting thread adult an for check they exit.

adults: for code the is Here

Extended child care solution (adult)

```

1 mutex.wait()
2     adults++
3     if waiting:
4         n = min(3, waiting)
5         childQueue.signal(n)
6         waiting -= n
7         children += n
8 mutex.signal()
9
10 # critical section
11
12 mutex.wait()
13     if children <= 3 * (adults-1):
14         adults--
15         mutex.signal()
16     else:
17         leaving++
18         mutex.signal()
19         adultQueue.wait()

```

they leave, they Before any. if children, waiting signal they enter, adults As
 exit, and adults decrement they so. If left, adults enough are there whether check
 to waiting is thread adult an While block, and leaving increment they Otherwise
 can children additional so section, critical the in adults the of one as counts it leave,
 enter.

problem party room The ۳.۷

con- a was there semester One College. Colby at was I while problem this wrote I
 Students of Dean the from someone that student a by allegation an over troversy
 public, was allegation the Although absence. his in room his searched had Office
 out found never we so case, the on comment to able wasn't Students of Dean the
 the was who mine, of friend a tease to problem this wrote I happened. really what
 Housing, Student of Dean
 of Dean the and students to apply constraints synchronization following The
 Students:

time. same the at room a in be can students of number Any .۱

the in students no are there if room a enter only can Students of Dean The .۲
 room the in students ۵۰ than more are there if or search) a conduct (to room
 party). the up break (to

enter, may students additional no room, the in is Students of Dean the While .۳
 leave. may students but

left. have students all until room the leave not may Students of Dean The .۴
 exclusion enforce to have not do you so Students. of Dean one only is There .۵
 deans. multiple among

Students of Dean the for and students for code synchronization write Puzzle:
 constraints. these of all enforces that

hint party Room ۱.۳.۷

Room party hint

```

1 students = 0
2 dean = 'not here'
3 mutex = Semaphore(1)
4 turn = Semaphore(1)
5 clear = Semaphore(0)
6 lieIn = Semaphore(0)

```

of state the is dean and room, the in students of number the counts students
 students protects mutex room". the "in or "waiting" be also can which Dean, the
 scoreboard. a of example another yet is this so ,dean and
 the in is Dean the while entering from students keeps that turnstile a is turn
 room.

Dean the and student a between rendezvouses as used are lieIn and clear
 scandal!). of kind other whole a is (which

solution party Room ۲.۳.۷

one. this to got I before versions of lot a through worked I hard. is problem This occasionally but correct. mostly was edition first the in appeared that version The break nor search neither could he that find then and room the enter would Dean the silence. embarrassed in off skulk to have would he so party. the up was result the but humiliation. this spared that solution a wrote Tesch Matt cor- was it that ourselves convincing time hard a had we that enough complicated readable. more bit a is which one. this to me led solution that But rect.

Room party solution (dean)

```

1 mutex.wait()
2     if students > 0 and students < 50:
3         dean = 'waiting'
4         mutex.signal()
5         lieIn.wait()      # and get mutex from the student.
6
7         # students must be 0 or >= 50
8
9         if students >= 50:
10            dean = 'in the room'
11            breakup()
12            turn.wait()    # lock the turnstile
13            mutex.signal()
14            clear.wait()   # and get mutex from the student.
15            turn.signal()  # unlock the turnstile
16
17        else:              # students must be 0
18            search()
19
20 dean = 'not here'
21 mutex.signal()

```

room. the in students are there if cases: three are there arrives. Dean the When breaks Dean the more. or ۵۰ are there If wait. to has Dean the more. or ۵۰ not but Dean the students. no are there If leave. to students the for waits and party the up leaves. and searches so student. a with rendezvous a for wait to has Dean the cases. two first the In to has he up. wakes Dean the When deadlock. a avoid to mutex up give to has he

the to similar is This back. mutex the get to needs he so scoreboard. the modify
the “Pass the is chose I solution The problem. Bar Sushi the in saw we situation
pattern. baton”

Room party solution (student)

```

1 mutex.wait()
2     if dean == 'in the room':
3         mutex.signal()
4         turn.wait()
5         turn.signal()
6         mutex.wait()
7
8     students += 1
9
10    if students == 50 and dean == 'waiting':
11        lieIn.signal()           # and pass mutex to the dean
12    else:
13        mutex.signal()
14
15 party()
16
17 mutex.wait()
18     students -= 1
19
20    if students == 0 and dean == 'waiting':
21        lieIn.signal()           # and pass mutex to the dean
22    elif students == 0 and dean == 'in the room':
23        clear.signal()          # and pass mutex to the dean
24    else:
25        mutex.signal()

```

the If Dean, the signal to have might student a where cases three are There
 .lieIn signal to has out one last the or in student ۵۰th the then waiting, is Dean
 student last the leave), to students the all for (waiting room the in is Dean the If
 the from passes mutex the that understood is it cases, three all In .clear signals out

Dean, the to student

of ۷ Line at know we how is obvious be not may that solution this of part One
 realize to is key The .۵۰ than less not or • be must students that code Dean's the
 was conditional first the either point: this to get to ways two only are there that
 was Dean the or :۵۰ than less not or • either is students that means which false,
 is students that again, means, which signaled, student a when lieIn on waiting
 .۵۰ than less not or • either

problem Bus Senate The ٩.٧

Riders College. Wellesley at bus Senate the on based originally was problem This
riders waiting the all arrives, bus the When bus. a for wait and stop bus a to come
for wait to has boarding is bus the while arrives who anyone but ,boardBus invoke
people Δ than more are there if people: Δ is bus the of capacity The bus. next the
bus. next the for wait to have will some waiting,
the If .depart invoke can bus the boarded, have riders waiting the all When
immediately. depart should it riders, no are there when arrives bus
constraints. these of all enforces that code synchronization Write Puzzle:

hint problem Bus ۱.۴.۷

solution: my in used I variables the are Here

Bus problem hint

```

1 riders = 0
2 mutex = Semaphore(1)
3 multiplex = Semaphore(50)
4 bus = Semaphore(0)
5 allAboard = Semaphore(0)

```

waiting: are riders many how of track keeps which ,riders protects mutex

area. boarding the in riders ۵۰ than more no are there sure makes multiplex

waits bus The arrives. bus the when signaled gets which ,bus on wait Riders

board. to student last the by signaled gets which ,allAboard on

#۱ solution problem Bus ۲.۴.۷

pattern. baton” the “Pass the using are we Again. bus. the for code the is Here

Bus problem solution (bus)

```

1 mutex.wait()
2 if riders > 0:
3     bus.signal()          # and pass the mutex
4     allAboard.wait()      # and get the mutex back
5 mutex.signal()
6
7 depart()
```

entering from arrivals late prevents which .mutex gets it arrives. bus the When
 it Otherwise. immediately. departs it riders. no are there If area. boarding the
 board. to riders the for waits and bus signals
 riders: the for code the is Here

Bus problem solution (riders)

```

1 multiplex.wait()
2     mutex.wait()
3     riders += 1
4     mutex.signal()
5
6     bus.wait()             # and get the mutex
7 multiplex.signal()
8
9 boardBus()
10
11 riders -= 1
12 if riders == 0:
13     allAboard.signal()
14 else:
15     bus.signal()          # and pass the mutex
```

although area. waiting the in riders of number the controls multiplex The
 .riders increments she until area waiting the enter doesn't rider a speaking. strictly
 understood is it up. wakes rider a When arrives. bus the until bus on wait Riders
 are there If .riders decrements rider each boarding. After mutex. the has she that
 next the to mutex the pass and bus signals rider boarding the waiting. riders more
 bus. the to back mutex the passes and allAboard signals rider last The rider.

departs. and mutex the releases bus the Finally.
you” for it do “I’ll the using problem this to solution a find you can Puzzle:
pattern?

#४ solution problem Bus ३.१.४

the than variables fewer uses which solution. this with up came Hutchins Grant
vari- the are Here mutexes. any around passing involve doesn't and one. previous
ables:

Bus problem solution #2 (initialization)

```
1 waiting = 0
2 mutex = new Semaphore(1)
3 bus = new Semaphore(0)
4 boarded = new Semaphore(0)
```

by protected is which area. boarding the in riders of number the is waiting
has rider a that signals boarded arrived: has bus the when signals bus .mutex
boarded.

bus. the for code the is Here

Bus problem solution (bus)

```
1 mutex.wait()
2 n = min(waiting, 50)
3 for i in range(n):
4     bus.signal()
5     boarded.wait()
6
7 waiting = max(waiting-50, 0)
8 mutex.signal()
9
10 depart()
```

loop The process. boarding the throughout it holds and mutex the gets bus The
of number the controlling By board. to her for waits and turn in rider each signals
boarding. from riders 50 than more prevents bus the signals.
example an is which .waiting updates bus the boarded. have riders the all When
pattern. you” for it do “I’ll the of
rendezvous. a and mutex a patterns: simple two uses riders the for code The

Bus problem solution (riders)

```
1 mutex.wait()
2     waiting += 1
3 mutex.signal()
```

```
4  
5 bus.wait()  
6 board()  
7 boarded.signal()
```

annoyed be might they boarding, is bus the while arrive riders if Challenge:
late allows that solution a find you Can one. next the for wait them make you if
constraints? other the violating without board to arrivals

problem Hall Faneuil The ۵.۷

who friend a by inspired was who Hutchins. Grant by written was problem This
 Boston. in Hall Faneuil at Citizenship of Oath her took
 Im- judge. one a and spectators. immigrants. threads: of kinds three are “There
 judge the point. some At down. sit then and in. check line. in wait must migrants
 the and enter. may one no building. the in is judge the When building. the enters
 in. check immigrants all Once leave. may Spectators leave. not may immigrants
 immigrants the confirmation. the After naturalization. the confirm can judge the
 after point some at leaves judge The Citizenship. U.S. of certificates their up pick
 their get immigrants After before. as enter now may Spectators confirmation. the
 leave.” may they certificates.

functions some threads the give let’s specific. more requirements these make To
 functions. those on constraints put and execute. to

- `.swear` `.sitDown` `.checkIn` `.enter` `invoke` `must` `Immigrants`
`.leave` and `getCertificate`
- `.leave` and `confirm` `.enter` `invokes` `judge` `The`
- `.leave` and `spectate` `.enter` `invoke` `Spectators`
- `may` `immigrants` and `enter` `may` `one` `no` `building.` `the` `in` `is` `judge` `the` `While`
`.leave` `not`
- `enter` `invoked` `have` `who` `immigrants` `all` `until` `confirm` `not` `can` `judge` `The`
`.checkIn` `invoked` `also` `have`
- `.confirm` `executed` `has` `judge` `the` `until` `getCertificate` `not` `can` `Immigrants`

Hint Problem Hall Faneuil ۱.۵.۷

Faneuil Hall problem hint

```

1 noJudge = Semaphore(1)
2 entered = 0
3 checked = 0
4 mutex = Semaphore(1)
5 confirmed = Semaphore(0)

```

pro- also it spectators: and immigrants incoming for turnstile a as acts noJudge
 checked room. the in immigrants of number the counts which entered tects
 .mutex by protected is it in: checked have who immigrants of number the counts
 .confirm executed has judge the that signals confirmed

solution problem Hall Faneuil ۲.۵.۷

immigrants: for code the is Here

Faneuil Hall problem solution (immigrant)

```

1 noJudge.wait()
2 enter()
3 entered++
4 noJudge.signal()
5
6 mutex.wait()
7 checkIn()
8 checked++
9
10 if judge == 1 and entered == checked:
11     allSignedIn.signal()
12     # and pass the mutex
13 else:
14     mutex.signal()
15
16 sitDown()
17 confirmed.wait()
18
19 swear()
20 getCertificate()
21
22 noJudge.wait()
23 leave()
24 noJudge.signal()

```

the in is judge the while enter: they when turnstile a through pass Immigrants

locked. is turnstile the room.

.checked update and in check to mutex get to have immigrants entering. After

and allSignedIn signals in check to immigrant last the waiting. judge a is there If

judge. the to mutex the passes

judge: the for code the is Here

Faneuil Hall problem solution (judge)

```

1 noJudge.wait()
2 mutex.wait()
3
4 enter()
5 judge = 1

```

```

6
7 if entered > checked:
8     mutex.signal()
9     allSignedIn.wait()
# and get the mutex back.
10
11 confirm()
12 confirmed.signal(checkered)
13 entered = checked = 0
14
15 leave()
16 judge = 0
17
18 mutex.signal()
19 noJudge.signal()

```

and entering, from spectators and immigrants bar to noJudge holds judge The
 .checked and entered access can he so mutex
 also has entered has who everyone when instant an at arrives judge the If
 mu- the up give to has she Otherwise, immediately, proceed can she in, checked
 is it .allSignedIn signals and in checks immigrant last the When wait, and tex
 back, mutex the get will judge the that understood
 immigrant every for once confirmed signals judge the ,confirm invoking After
 you”), for it do “I’ll of example (an counters the resets then and in, checked has who
 .noJudge and mutex releases and leaves judge the Then
 and swear invoke immigrants ,confirmed signals judge the After
 to turnstile noJudge the for wait then and concurrently, getCertificate
 leaving, before open
 the is obey to have they constraint only the easy: is spectators for code The
 turnstile, noJudge

Faneuil Hall problem solution (spectator)

```

1 noJudge.wait()
2 enter()
3 noJudge.signal()
4
5 spectate()
6
7 leave()

```

get they after stuck. get to immigrants for possible is it solution this in Note:
immigrants. of batch next the in swear to coming judge another by certificate. their
in-ceremony. swearing another through wait to have might they happens. that If
the after that constraint additional the handle to solution this modify Puzzle:
judge the before leave must in sworn been have who immigrants all leaves. judge
again. enter can

Hint Problem Hall Faneuil Extended ۳.۵.۷

variables: additional following the uses solution My

Faneuil Hall problem hint

```
1 exit = Semaphore(0)
2 allGone = Semaphore(0)
```

it solve can we rendezvous. additional an involves problem extended the Since

semaphores. two with

again. pattern baton” the “pass the use to useful it found I hint: other One

solution problem Hall Faneuil Extended ۴.۵.۷

.۲۱ Line at starts difference The before. as same the is solution this of half top The
leave. to judge the for here wait Immigrants

Faneuil Hall problem solution (immigrant)

```

1 noJudge.wait()
2 enter()
3 entered++
4 noJudge.signal()
5
6 mutex.wait()
7 checkIn()
8 checked++
9
10 if judge = 1 and entered == checked:
11     allSignedIn.signal()
12 # and pass the mutex
13 else:
14     mutex.signal()
15
16 sitDown()
17 confirmed.wait()
18
19 swear()
20 getCertificate()
21
22 exit.wait() # and get the mutex
23 leave()
24 checked--
25 if checked == 0:
26     allGone.signal() # and pass the mutex
27 else:
28     exit.signal() # and pass the mutex

```

leave. to ready is judge the When .۲۱ Line at starts difference the judge. the For
possibly and immigrants. more allow would that because .noJudge release can't she
to immigrant one allows which .exit signals she Instead. enter. to judge. another
.mutex passes and leave.

ba- the passes then and checked decrements signal the gets that immigrant The
passes and allGone signals leave to immigrant last The immigrant. next the to ton
has it but necessary. strictly not is pass-back This judge. the to back mutex the

phase the end to noJudge and mutex both releases judge the that feature nice the
cleanly.

Faneuil Hall problem solution (judge)

```
1 noJudge.wait()
2 mutex.wait()
3
4 enter()
5 judge = 1
6
7 if entered > checked:
8     mutex.signal()
9     allSignedIn.wait()
10 # and get the mutex back.
11
12 confirm()
13 confirmed.signal(checked)
14 entered = 0
15
16 leave()
17 judge = 0
18
19 exit.signal() # and pass the mutex
20 allGone.wait() # and get it back
21 mutex.signal()
22 noJudge.signal()
```

unchanged. is problem extended the for code spectator The

problem Hall Dining 9.7

Olin at class Synchronization my during Pollack Jon by written was problem This
College.

dine invoking After .leave then and dine invoke hall dining the in Students
leave”. to “ready considered is student a leave invoking before and
main- to order in that. is students to applies that constraint synchronization The
is student A alone. table a at sit never may student a suave, social of illusion the tain
leave invokes dine invoked has who else everyone if alone sitting be to considered
.dine finished has she before
constraint. this enforces that code write Puzzle:

hint problem Hall Dining ۱.۶.۷

Dining Hall problem hint

```

1 eating = 0
2 readyToLeave = 0
3 mutex = Semaphore(1)
4 okToLeave = Semaphore(0)

```

usual the is this so «mutex by protected counters are readyToLeave and eating

pattern. scoreboard

table. the at alone left be would student another but leave. to ready is student a If

signals. and situation the changes student another until okToLeave on waits she

solution problem Hall Dining ۲.۶.۷

where situation one only is there that realize will you constraints. the analyze you If to wants who student one and eating student one is there if wait. to has student a might student another situation: this of out get to ways two are there But leave.

finish. might student dining the or eat. to arrive counters. the updates student waiting the signals who student the case. either In example another is This back. mutex the get to have doesn't student waiting the so pattern. you" for it do "I'll the the of

Dining Hall problem solution

```

1  getFood()
2
3  mutex.wait()
4  eating++
5  if eating == 2 and readyToLeave == 1:
6      okToLeave.signal()
7      readyToLeave--
8  mutex.signal()
9
10 dine()
11
12 mutex.wait()
13 eating--
14 readyToLeave++
15
16 if eating == 1 and readyToLeave == 1:
17     mutex.signal()
18     okToLeave.wait()
19 elif eating == 0 and readyToLeave == 2:
20     okToLeave.signal()
21     readyToLeave -= 2
22     mutex.signal()
23 else:
24     readyToLeave--
25     mutex.signal()
26
27 leave()

```

waiting one and eating student one sees she if in. checking is student is When him. for readyToLeave decrements and hook the off waiter the lets she leave. to

cases: three checks student the dining. After
 up give to has student departing the eating, left student one only is there If •
 wait, and mutex the
 him signals she her, for waiting is someone that finds student departing the If •
 them, of both for counter the updates and
 leaves, and readyToLeave decrements just she Otherwise, •

problem Hall Dining Extended ३.९.१

As step, another add we if challenging more little a gets problem Hall Dining The
 invoking After .leave then and dine .getFood invoke they lunch to come students
 Similarly, eat”. to “ready considered is student a ,dine invoking before and getFood
 leave”. to “ready considered is student a dine invoking after
 table a at sit never may student a applies: constraint synchronization same The
 either if alone sitting be to considered is student A alone.
 to ready one no and table the at else one no is there while dine invokes She •
 or eat.
 finished has she before leave invokes dine invoked has who else everyone •
 .dine

constraints, these enforces that code write Puzzle:

hint problem Hall Dining Extended ۶.۶.۷

solution: my in used I variables the are Here

Extended Dining Hall problem hint

```

1 readyToEat = 0
2 eating = 0
3 readyToLeave = 0
4 mutex = Semaphore(1)
5 okToSit = Semaphore(0)
6 okToLeave = Semaphore(0)

```

.mutex by protected all counters. are readyToLeave and eating ,readyToEat
 or okToSit on waits she proceed. cannot she where situation a in is student a If
 signals. and situation the changes student another until okToLeave
 whether of track keep help to hasMutex named variable per-thread a used also I
 mutex. the holds thread a not or

solution problem Hall Dining Extended ۵.۶.۷

where situation one only is there that realize we constraints. the analyze we if Again.
 else one no and eating one no is there if wait. to has eat to ready is who student a
 eat. to ready is who arrives else someone if is out way only the And eat. to ready

Extended Dining Hall problem solution

```

1  getFood()
2
3  mutex.wait()
4  readyToEat++
5  if eating == 0 and readyToEat == 1:
6      mutex.signal()
7      okToSit.wait()
8  elif eating == 0 and readyToEat == 2:
9      okToSit.signal()
10     readyToEat -= 2
11     eating += 2
12     mutex.signal()
13 else:
14     readyToEat--
15     eating++
16     if eating == 2 and readyToLeave == 1:
17         okToLeave.signal()
18         readyToLeave--
19     mutex.signal()
20
21 dine()
22
23 mutex.wait()
24 eating--
25 readyToLeave++
26 if eating == 1 and readyToLeave == 1:
27     mutex.signal()
28     okToLeave.wait()
29 elif eating == 0 and readyToLeave == 2:
30     okToLeave.signal()
31     readyToLeave -= 2
32     mutex.signal()
33 else:
34     readyToLeave--
35     mutex.signal()
36
37 leave()

```

waiting a that so pattern you” for it do “I’ll the used I solution, previous the in As
back. mutex the get to have doesn’t student
the that is one previous the and solution this between difference primary The
allows student second the and wait. to has table empty an at arrives who student first
waiting students for check to have don’t we case. either It proceed. to students both
table! empty an leave can one no since leave. to

Python in Synchronization

synchronization of details ugly the of some avoided have we pseudocode. using By
 Python: in code synchronization real at look we'll chapter this In world. real the in
 C. at look we'll chapter next the in
 complete environment. multithreading pleasant reasonably a provides Python
 in code cleanup some is there but foibles. few a has It objects. Semaphore with
 better. little a things makes that Appendix
 example: simple a is Here

Listing:

```

1 from threading_cleanup import *
2
3 class Shared:
4     def __init__(self):
5         self.counter = 0
6
7     def child_code(shared):
8         while True:
9             shared.counter += 1
10            print shared.counter
11            time.sleep(0.5)
12
13 shared = Shared()
14 children = [Thread(child_code, shared) for i in range(2)]
15 for child in children: child.join()
```

out line this leave will I : Appendix from code cleanup the runs line first The
 examples. other the of
 variables Global variables. shared contain will that type object an defines Shared
 Threads examples. these in any use won't we but threads. between shared also are
 the in local also are function a inside declared are they that sense the in local are that
 thread-specific. are they that sense
 new the prints .counter increments that loop infinite an is code child The
 seconds. 0.5 for sleeps then and value.
 children the for waits then children. two and shared creates thread parent The
 won't). they case. this in (which exit to

problem checker Mutex 1.8

unsynchro- make children the that notice will synchronization of students Diligent
 might you program. this run you If safe! not is which .counter to updates nized
 synchronization about thing nasty The won't. probably you but errors. some see
 may testing extensive even that means which unpredictable. are they that is errors
 them. reveal not
 we case. this In search. the automate to necessary often is it errors. detect To
 .counter of values the of track keeping by errors detect can

Listing:

```

1 class Shared:
2     def __init__(self, end=10):
3         self.counter = 0
4         self.end = end
5         self.array = [0]* self.end
6
7     def child_code(shared):
8         while True:
9             if shared.counter >= shared.end: break
10            shared.array[shared.counter] += 1
11            shared.counter += 1
12
13 shared = Shared(10)
14 children = [Thread(child_code, shared) for i in range(2)]

```

15

16

1] ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,[1

might we array. the of size the increase we If correct. disappointingly is which
result. the check to harder gets also it but errors. more expect

array: the in results the of histogram a making by checker the automate can We

Listing:

```

1 class Histogram(dict):
2     def __init__(self, seq=[]):
3         for item in seq:
4             self[item] = self.get(item, 0) + 1
5
6 print Histogram(shared.array)
```

get I program, the run I when Now

10} {1:

but far, so errors No expected, as times, \ appeared \ value the that means which

interesting: more get things bigger, end make we if

100} {1: ,100 = end

1000} {1: ,1000 = end

10000} {1: ,10000 = end

72439} 2: ,27561 {1: ,100000 = end

between switches context of lot a are there that enough big is end When Oops!

of *lot* a get we case, this In errors, synchronization get to start we children, the

threads where pattern recurring a into falls program the that suggests which errors,

section, critical the in interrupted consistently are

which errors, synchronization of dangers the of one demonstrates example This

a in time one occurs error an If random, not are they but rare, be may they that is

row, a in times million a happen won't it mean doesn't that million,

ac- exclusive enforce to program this to code synchronization add Puzzle:

from section this in code the download can You variables, shared the to cess

greenteapress.com/semaphores/counter.py

hint checker Mutex \.A.A

used: I Shared of version the is Here

Listing:

```
1 class Shared:
2     def __init__(self, end=10):
3         self.counter = 0
4         self.end = end
5         self.array = [0]* self.end
6         self.mutex = Semaphore(1)
```

no as come should which .mutex named Semaphore the is change only The
surprise.

solution checker Mutex 1.1.1

solution: my is Here

Listing:

```

1 def child_code(shared):
2     while True:
3         shared.mutex.wait()
4         if shared.counter < shared.end:
5             shared.array[shared.counter] += 1
6             shared.counter += 1
7             shared.mutex.signal()
8         else:
9             shared.mutex.signal()
10            break

```

book, this in problem synchronization difficult most the not is this Although
to easy is it particular. In right, details the get to tricky it found have might you
a cause would which loop, the of out breaking before mutex the signal to forget
deadlock.

result: following the got and ,1000000 = end with solution this ran I

1000000} {1:

start, good a to off is it but correct, is solution my mean doesn't that course. Of

problem machine coke The 2.8

removing and adding consumers and producers simulates program following The
machine: coke a from cokes

Listing:

```

1 import random
2
3 class Shared:
4     def __init__(self, start=5):
5         self.cokes = start
6
7     def consume(shared):
8         shared.cokes -= 1
9         print shared.cokes
10
11    def produce(shared):
12        shared.cokes += 1
13        print shared.cokes
14
15    def loop(shared, f, mu=1):
16        while True:
17            t = random.expovariate(1.0/mu)
18            time.sleep(t)
19            f(shared)
20
21    shared = Shared()
22    fs = [consume]*2 + [produce]*2
23    threads = [Thread(loop, shared, f) for f in fs]
24    for thread in threads: thread.join()

```

shared the So full. half initially is machine the that and cokes. 10 is capacity The
.5 is cokes variable
both They consumers. two and producers two threads. 4 creates program The
These .consume invoke consumers and produce invoke producers but .loop run
no-no. a is which variable. shared a to access unsynchronized make functions
cho- duration a for sleep consumers and producers loop. the through time Each
producers two are there Since .mu mean with distribution exponential an from sen
average. on second. per machine the to added get cokes two consumers. two and
removed. get two and

vary can in run short the in but constant. is cokes of number the average on So
 value the see probably will you while. a for program the run you If widely. quite
 should these of neither course. Of .\ • above climb or zero. below dip cokes of
 happen.

synchronization following the enforce to program this to code add Puzzle:

constraints:

- exclusive. mutually be should cokes to Access •
- added. is coke a until block should consumers zero. is cokes of number the If •
- removed. is coke a until block should producers .\ • is cokes of number the If •

greenteapress.com/semaphores/ from program the download can You

[coke.py](#)

hint machine Coke 1.2.8

solution: my in used I variables shared the are Here

Listing:

```

1 class Shared:
2     def __init__(self, start=5, capacity=10):
3         self.cokes = Semaphore(start)
4         self.slots = Semaphore(capacity-start)
5         self.mutex = Semaphore(1)

```

tricky it makes which integer): simple a than (rather now Semaphore a is cokes
 Semaphore. a of value the access never should you course. Of value. its print to
 methods cheater the of any provide doesn't way do-gooder usual its in Python and
 implementations. some in see you
 stored is Semaphore a of value the that know to interesting it find might you But
 know. don't you case in Also. ._Semaphore__value named attribute private a in
 You attributes. private to access on restriction any enforce actually doesn't Python
 interested. be might you thought I but course. of it. access never should
 Ahem.

solution machine Coke 2.2.1

with up coming trouble no had have should you book. this of rest the read you've If
this: as good as least at something

Listing:

```
1 def consume(shared):  
2     shared.cokes.wait()  
3     shared.mutex.wait()  
4     print shared.cokes.value()  
5     shared.mutex.signal()  
6     shared.slots.signal()  
7  
8 def produce(shared):  
9     shared.slots.wait()  
10    shared.mutex.wait()  
11    print shared.cokes._Semaphore__value  
12    shared.mutex.signal()  
13    shared.cokes.signal()
```

num- the that confirm to able be should you while. a for program this run you If
solution this So .\• than greater or negative never is machine the in cokes of ber
correct. be to seems
far. So

C in Synchronization

Ap- C. in program synchronized multithreaded. a write will we section this In more little a code C the make to use I code utility the of some provides ب pendix code. that on depend section this in examples The palatable.

exclusion Mutual ۱.۹

variables: shared contains that structure a defining by start We'll

Listing:

```

1 typedef struct {
2     int counter;
3     int end;
4     int *array;
5 } Shared;
6
7 Shared *make_shared (int end)
8 {
9     int i;
10    Shared *shared = check_malloc (sizeof (Shared));
11
12    shared->counter = 0;
13    shared->end = end;
14
15    shared->array = check_malloc (shared->end * sizeof(int));
16    for (i=0; i<shared->end; i++) {

```

```

17     shared->array[i] = 0;
18 }
19 return shared;
20 }

```

until threads concurrent by incremented be will that variable shared a is counter keeping by errors synchronization for check to array use will We .end reaches it increment. each after counter of value the of track

code Parent ۱.۱.۹

com- to them for wait and threads create to uses thread parent the code the is Here plete:

Listing:

```

1 int main ()
2 {
3     int i;
4     pthread_t child[NUM_CHILDREN];
5
6     Shared *shared = make_shared (100000);
7
8     for (i=0; i<NUM_CHILDREN; i++) {
9         child[i] = make_thread (entry, shared);
10    }
11
12    for (i=0; i<NUM_CHILDREN; i++) {
13        join_thread (child[i]);
14    }
15
16    check_array (shared);
17    return 0;
18 }

```

com- to them for waits loop second the threads: child the creates loop first The check to check_array invokes parent the finished. has child last the When plete. Appendix in defined are join_thread and make_thread errors. for

code Child ۲.۱.۹

children: the of each by executed is that code the is Here

Listing:

```

1 void child_code (Shared *shared)
2 {
3     while (1) {
4         if (shared->counter >= shared->end) {
5             return;
6         }
7         shared->array[shared->counter]++;
8         shared->counter++;
9     }
10 }

```

into index an as counter use threads child the loop, the through time Each
counter increment they Then element. corresponding the increment and array
done. they're if see to check and

errors Synchronization ۳.۱.۹

incremented be should array the of element each correctly. works everything If
not are that elements of number the count just can we errors. for check to So once.

: \

Listing:

```

1 void check_array (Shared *shared)
2 {
3     int i, errors=0;
4
5     for (i=0; i<shared->end; i++) {
6         if (shared->array[i] != 1) errors++;
7     }
8     printf ("%d errors.\n", errors);
9 }

```

from code) cleanup the (including program this download can You

greenteapress.com/semaphores/counter.c

this: like output see should you program. the run and compile you If

0 counter at child Starting

10000

```

20000
30000
40000
50000
60000
70000
80000
90000
done. Child
100000 counter at child Starting
done. Child
Checking...
errors. 0

```

operating your of details on depends children the of interaction the course. Of shown example the In computer. your on running programs other also and system started. got thread other the before end to • from way the all ran thread one here.

errors. synchronization no were there that surprising not is it so children. the between switches context more are there bigger. gets end as But

.\•••••••• is end when errors see to start I system my On and variables shared the to access exclusive enforce to semaphores use Puzzle:

errors. no are there that confirm to again program the run

hint exclusion Mutual ۴.۱.۹

solution: my in used I Shared of version the is Here

Listing:

```

1 typedef struct {
2     int counter;
3     int end;
4     int *array;
5     Semaphore *mutex;
6 } Shared;
7
8 Shared *make_shared (int end)
9 {
10     int i;
11     Shared *shared = check_malloc (sizeof (Shared));
12
13     shared->counter = 0;
14     shared->end = end;
15
16     shared->array = check_malloc (shared->end * sizeof(int));
17     for (i=0; i<shared->end; i++) {
18         shared->array[i] = 0;
19     }
20     shared->mutex = make_semaphore(1);
21     return shared;
22 }
```

the with mutex the initializes ۲۰ Line Semaphore: a as mutex declares ۵ Line

.\ value

solution exclusion Mutual ۵.۱.۹

code: child the of version synchronized the is Here

Listing:

```

1 void child_code (Shared *shared)
2 {
3     while (1) {
4         sem_wait(shared->mutex);
5         if (shared->counter >= shared->end) {
6             sem_signal(shared->mutex);
7             return;
8         }
9
10        shared->array[shared->counter]++;
11        shared->counter++;
12        sem_signal(shared->mutex);
13    }
14 }
```

to remember to is thing tricky only the here: surprising too nothing is There

statement. return the before mutex the release

greenteapress.com/semaphores/ from solution this download can You

[counter_mutex.c](#)

semaphores own your Make ٢.٩

Threads use that programs for tools synchronization used commonly most The these of explanation an For semaphores. not variables. condition and mutexes are

.[٧] *Threads POSIX with Programming* Butenhof's recommend I tools. write to them use then and variables. condition and mutexes about read Puzzle:

semaphores. of implementation an my is Here solutions. your in code utility following the use to want might You

mutexes: Pthreads for wrapper

Listing:

```

1 typedef pthread_mutex_t Mutex;
2
3 Mutex *make_mutex ()
4 {
5     Mutex *mutex = check_malloc (sizeof(Mutex));
6     int n = pthread_mutex_init (mutex, NULL);
7     if (n != 0) perror_exit ("make_lock failed");
8     return mutex;
9 }
10
11 void mutex_lock (Mutex *mutex)
12 {
13     int n = pthread_mutex_lock (mutex);
14     if (n != 0) perror_exit ("lock failed");
15 }
16
17 void mutex_unlock (Mutex *mutex)
18 {
19     int n = pthread_mutex_unlock (mutex);
20     if (n != 0) perror_exit ("unlock failed");
21 }

```

variables: condition Pthread for wrapper my And

Listing:

```
1 typedef pthread_cond_t Cond;
2
3 Cond *make_cond ()
4 {
5     Cond *cond = check_malloc (sizeof(Cond));
6     int n = pthread_cond_init (cond, NULL);
7     if (n != 0) perror_exit ("make_cond failed");
8     return cond;
9 }
10
11 void cond_wait (Cond *cond, Mutex *mutex)
12 {
13     int n = pthread_cond_wait (cond, mutex);
14     if (n != 0) perror_exit ("cond_wait failed");
15 }
16
17 void cond_signal (Cond *cond)
18 {
19     int n = pthread_cond_signal (cond);
20     if (n != 0) perror_exit ("cond_signal failed");
21 }
```

hint implementation Semaphore ١.٢.٩

semaphores: my for used I definition structure the is Here

Listing:

```

1 typedef struct {
2     int value, wakeups;
3     Mutex *mutex;
4     Cond *cond;
5 } Semaphore;
```

pend- of number the counts wakeups semaphore. the of value the is value
yet not have but woken been have that threads of number the is, that signals: ing
semaphores our that sure make to is wakeups for reason The execution. resumed

.٣.٢ Section in described ,٣ Property have

condition the is cond ;wakeups and value to access exclusive provides mutex

semaphore. the on wait they if on wait threads variable

structure: this for code initialization the is Here

Listing:

```

1 Semaphore *make_semaphore (int value)
2 {
3     Semaphore *semaphore = check_malloc (sizeof(Semaphore));
4     semaphore->value = value;
5     semaphore->wakeups = 0;
6     semaphore->mutex = make_mutex ();
7     semaphore->cond = make_cond ();
8     return semaphore;
9 }
```


implementation Semaphore 2.2.9

condition and mutexes Pthread's using semaphores of implementation my is Here
variables:

Listing:

```

1 void sem_wait (Semaphore *semaphore)
2 {
3     mutex_lock (semaphore->mutex);
4     semaphore->value--;
5
6     if (semaphore->value < 0) {
7         do {
8             cond_wait (semaphore->cond, semaphore->mutex);
9             } while (semaphore->wakeups < 1);
10        semaphore->wakeups--;
11    }
12    mutex_unlock (semaphore->mutex);
13 }
14
15 void sem_signal (Semaphore *semaphore)
16 {
17     mutex_lock (semaphore->mutex);
18     semaphore->value++;
19
20     if (semaphore->value <= 0) {
21         semaphore->wakeups++;
22         cond_signal (semaphore->cond);
23     }
24     mutex_unlock (semaphore->mutex);
25 }
```

the is tricky be might that thing only the straightforward: is this of Most
variable, condition a use to way unusual an is This .V Line at loop do...while
necessary. is it case this in but
loop? while a with loop do...while this replace we can't why Puzzle:

detail implementation Semaphore 3.2.9

be would It .3 Property have not would implementation this loop, while a With
 signal. own its catch and around run then and signal to thread a for possible
 one signals, thread a when that 'guaranteed is it loop, do...while the With
 at mutex the gets thread another if even signal, the get will threads waiting the of
 resumes, threads waiting the of one before 3 Line

<http://en.wikipedia.org/> (see wakeup spurious well-timed a that out turns It almost. Well, 'guarantee, this violate can (wiki/Spurious_wakeup

کتاب نامه

- [۱] *Practice and Principles Programming: Concurrent* Andrews. R. Gregory [۱]
Addison-Wesley. ۱۹۹۱.
- [۲] Addison-Wesley. *Threads POSIX with Programming* Butenhof. R. David [۲]
۱۹۹۷.
- [۳] *Pro* – in Reprinted ۱۹۶۵ processes. sequential Cooperating Dijkstra. Edsger [۳]
۱۹۶۸ York New Press. Academic ed. Genuys. F. *Languages gramming*
- [۴] *Bulletin SIGCSE ACM* revisited. philosophers Dining Gingras. R. Armando [۴]
۱۹۹۰ September ۲۸، ۲۴–۲۱: (۳) ۲۲
- [۵] *Controlled Supporting Middleware: and Systems Operating* Hailperin. Max [۵]
۲۰۰۶ Technology. Course Thompson *Interaction*
- [۶] prob– exclusion mutual the to solution starvation-free A Morris. M. Joseph [۶]
۱۹۷۹ February ۸۰–۸: ۷۶، *Letters Processing Information* lem.
- [۷] without problem smokers' cigarette the to solution a On Parnas. L. David [۷]
March ۱۸۳–۱۸: ۱۸۱، *ACM the of Communications* statements. conditional
۱۹۷۵.
- [۸] for primitives semaphore Dijkstra's of capabilities and Limitations Patil. Suhas [۸]
۱۹۷۱ MIT. report. Technical processes. among coordination
- [۹] ۲۰۰۴، *SIGCSE ACM* In semaphores. for patterns Design Reek. A. Kenneth [۹]

[۱۰] *Concepts Systems Operating* Galvin. Baer Peter and Silberschatz Abraham

.۱۹۹۷ edition, fifth Wesley, Addison

Pren- *Principles Design and Internals Systems: Operating* Stallings. William [۱۱]

.۲۰۰۰ edition, fourth Hall, tice

second Hall, Prentice *Systems Operating Modern* Tanenbaum. S. Andrew [۱۲]

.۲۰۰۱ edition,

threads Python up Cleaning

pretty are threads Python environments. threading other of lot a to Compared
fix can you Fortunately. me. annoy that features of couple a are there but good.
code. clean-up little a with them

۱.۵ Semaphore methods

which ,release and acquire called are semaphores Python for methods the First.
years. of couple a for book this on working after but choice. reasonable perfectly a is
subclassing by way my it have can I Fortunately. .wait and signal to used am I
module: threading the in Semaphore of version the

Semaphore name change

```
1 import threading
2
3 class Semaphore(threading._Semaphore):
4     wait = threading._Semaphore.acquire
5     signal = threading._Semaphore.release
```

the using Semaphores manipulate and create can you defined. is class this Once
book. this in syntax

Semaphore example

```
1 mutex = Semaphore()
```

```

2 mutex.wait()
3 mutex.signal()

```

threads Creating २.३

for interface the is me annoys that module threading the of feature other The
two and arguments keyword requires way usual The threads. starting and creating
steps:

Thread example (standard way)

```

1 import threading
2
3 def function(x, y, z):
4     print x, y, z
5
6 thread = threading.Thread(target=function, args=[1, 2, 3])
7 thread.start()

```

you when But effect. immediate no has thread the creating example. this In
argu- given the with function target the executes thread new the .start invoke
starts. it before thread the with something do to need you if great is This ments.
are args and target arguments keyword the think I Also. do. never almost I but
awkward.

code. of lines four with problems these of both solve can we Fortunately.

Cleaned-up Thread class

```

1 class Thread(threading.Thread):
2     def __init__(self, t, *args):
3         threading.Thread.__init__(self, target=t, args=args)
4         self.start()

```

automatically: start they and interface. nicer a with threads create can we Now

Thread example (my way)

```

1 thread = Thread(function, 1, 2, 3)

```

with Threads multiple create to is which like. I idiom an to itself lends also This
comprehension: list a

Multiple thread example

```
1 threads = [Thread(function, i, i, i) for i in range(10)]
```

interrupts keyboard Handling ३.१

inter- be can't Thread.join that is class threading the with problem other One
into translates Python which .SIGINT signal the generates which Ctrl-C. by rupted
KeyboardInterrupt. a

program: following the write you if So.

Unstoppable program

```

1 import threading, time
2
3 class Thread(threading.Thread):
4     def __init__(self, t, *args):
5         threading.Thread.__init__(self, target=t, args=args)
6         self.start()
7
8 def parent_code():
9     child = Thread(child_code, 10)
10    child.join()
11
12 def child_code(n=10):
13     for i in range(n):
14         print i
15         time.sleep(1)
16
17 parent_code()

```

. \SIGINT a or Ctrl-C with interrupted be cannot it that find will You
works only it so ,os.wait and os.fork uses problem this for workaround My
the threads. new creating before works: it how Here's Macintosh. and UNIX on
and returns process new The process. new a forks which ,watcher invokes program
to process child the for waits process original The program. the of rest the executes
:watcher name the hence complete.

The watcher

```

1 import threading, time, os, signal, sys
2
3 class Thread(threading.Thread):
4     def __init__(self, t, *args):
5         threading.Thread.__init__(self, target=t, args=args)
6         self.start()
7
8 def parent_code():
9     child = Thread(child_code, 10)
10    child.join()
11

```

was it but ,\16V93 number assigned and reported been had bug this writing. this of time the At¹

.(<https://sourceforge.net/projects/python/>) unassigned and open

```

12 def child_code(n=10):
13     for i in range(n):
14         print i
15         time.sleep(1)
16
17 def watcher():
18     child = os.fork()
19     if child == 0: return
20     try:
21         os.wait()
22     except KeyboardInterrupt:
23         print 'KeyboardInterrupt '
24         os.kill(child, signal.SIGKILL)
25         sys.exit()
26
27 watcher()
28 parent_code()

```

with it interrupt to able be should you program. the of version this run you If
to delivered is SIGINT the that guaranteed is it think I but sure. not am I Ctrl-C.
deal to have threads child and parent the thing less one that's so process. watcher the
with.

can you which ,threading_cleanup.py named file a in code this all keep I
greenteapress.com/semaphores/threading_cleanup.py from download
code this that understanding the with presented are ^ Chapter in examples The
code. example the to prior executes

پیوست ب

threads POSIX up Cleaning

little a C in multithreading make to use I code utility some present I section. this In
code. this on based are ۹ Section in examples The pleasant. more
Threads. POSIX is C with used standard threading popular most the Probably
interface an and model thread a defines standard POSIX The short. for Pthreads or
imple– an provide UNIX of versions Most threads. controlling and creating for
Pthreads. of mentation

code Pthread Compiling ب.۱

libraries: C most using like is Pthreads Using

program. your of beginning the at files headers include You •

Pthreads. by defined functions calls that code write You •

library. Pthread the with it link you program. the compile you When •

headers: following the include I examples. my For

Headers

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
```

```
#include <semaphore.h>
```

for is fourth the and Pthreads for is third the standard: are two first The
option -l the use can you .gcc in library Pthread the with compile To semaphores.

line: command the on

Listing:

```
gcc -g -O2 -o array array.c -lpthread
```

the with links optimization, and info debugging with array.c compiles This
.array named executable an generates and library. Pthread
you handling, exception provides that Python like language a to used are you If
error for check to you require that C like languages with annoyed be probably will
calls function library wrapping by hassle this mitigate often I explicitly. conditions
example. For functions. own my inside code error-checking their with together
value. return the checks that malloc of version a is here

Listing:

```
void *check_malloc(int size)
{
    void *p = malloc (size);
    if (p == NULL) {
        perror ("malloc failed");
        exit (-1);
    }
    return p;
}
```

threads Creating ب.٢

my here's use: to going I'm functions Pthread the with thing same the done I've
.pthread_create for wrapper

Listing:

```
pthread_t make_thread(void *entry)(void , Shared *shared)
{
    int n;
    pthread_t thread;
```

```

5
6     n = pthread_create (&thread, NULL, entry, (void *)shared);
7     if (n != 0) {
8         perror ("pthread_create failed");
9         exit (-1);
10    }
11    return thread;
12 }

```

of think can you which ,pthread_t is pthread_create from type return The
imple- the about worry to have shouldn't You thread. new the for handle a as
a of semantics the has it that know to have do you but ,pthread_t of mentation
immutable an as handle thread a of think can you that means That . ' type primitive
this point I problems. causing without value by it pass or it copy can you so value.

minute. a in to get will I which semaphores, for true not is it because now out
of handle the returns function my and • returns it succeeds, pthread_create If
my and code error an returns pthread_create occurs. error an If thread. new the
exits. and message error an prints function
the with Starting explaining. some take pthread_create of parameters The
fol- The variables. shared contains that structure user-defined a is Shared second.
type: new the creates statement typedef lowing

Listing:

```

1 typedef struct {
2     int counter;
3 } Shared;

```

space allocates make_shared .counter is variable shared only the case, this In
contents: the initializes and structure Shared a for

Listing:

```

1 Shared *make_shared ()
2 {
3     int i;
4     Shared *shared = check_malloc (sizeof (Shared));
5     shared->counter = 0;
6     return shared;

```

know. I implementations the all in is pthread_t a what is which example, for integer. an Like\

```
7 }
```

.pthread_create to back get let's structure. data shared a have we that Now
returns and pointer void a takes that function a to pointer a is parameter first The
are you bleed, eyes your makes type this declaring for syntax the If pointer. void a
where function the specify to is parameter this of purpose the Anyway, alone. not
named is function this convention. By begin. will thread new the of execution the
:entry

Listing:

```
1 void *entry (void *arg)
2 {
3     Shared *shared = (Shared *) arg;
4     child_code (shared);
5     pthread_exit (NULL);
6 }
```

pro- this in but pointer. void a as declared be to has entry of parameter The
it typecast can we so structure. Shared a to pointer a really is it that know we gram
work. real the does which ,child_code to along it pass then and accordingly
to used be can which pthread_exit invoke we returns. child_code When
this In thread. this with joins that parent) the (usually thread any to value a pass
.NULL pass we so say. to nothing has child the case.

threads Joining ۳.ب

invokes it complete. to thread another for wait to want thread one When
:pthread_join for wrapper my is Here .pthread_join

Listing:

```
1 void join_thread (pthread_t thread)
2 {
3     int ret = pthread_join (thread, NULL);
4     if (ret == -1) {
5         perror ("pthread_join failed");
6         exit (-1);
7     }
8 }
```

function my All for. wait to want you thread the of handle the is parameter The
result. the check and pthread_join call is does

Semaphores ۴.ب

part not is interface This semaphores. for interface an specifies standard POSIX The
semaphores. provide also Pthreads implement that UNIXes most but Pthreads. of
your make can you semaphores. without and Pthreads with yourself find you If
Section see own: ۲.۹.

the about know to have shouldn't You .sem_t type have semaphores POSIX
seman- structure has it that know to have do you but type. this of implementation
the of copy a making are you variable a to it assign you if that means which tics.
In idea. bad a certainly almost is semaphore a Copying structure. a of contents
undefined. is copy the of behavior the POSIX.
semantics. structure with types denote to letters capital use I programs. my In
wrapper a put to easy is it Fortunately. pointers. with them manipulate always I and
the and typedef the is Here object. proper a like behave it make to sem_t around
semaphores: initializes and creates that wrapper

Listing:

```

1 typedef sem_t Semaphore;
2
3 Semaphore *make_semaphore (int n)
4 {
5     Semaphore *sem = check_malloc (sizeof(Semaphore));
6     int ret = sem_init(sem, 0, n);
7     if (ret == -1) {
8         perror ("sem_init failed");
9         exit (-1);
10    }
11    return sem;
12 }
```

al- It parameter. a as semaphore the of value initial the takes make_semaphore
.Semaphore to pointer a returns and it. initializes Semaphore. a for space locates

_ returns it that means which reporting. error UNIX old-style uses `sem_init` that is functions wrapper these about thing nice Once wrong. went something if style. reporting which use functions which remember to have don't we pro- real a like looks almost that code C write can we definitions. these With language: gramming

Listing:

```
1 Semaphore *mutex = make_semaphore(1);
2 sem_wait(mutex);
3 sem_post(mutex);
```

fix can we but .signal of instead post use semaphores POSIX Annoyingly, that:

Listing:

```
1 int sem_signal(Semaphore *sem)
2 {
3     return sem_post(sem);
4 }
```

now. for cleanup enough That's