تمرین‌های نهایی

در ادامه تمرین‌های نهایی آزمایشگاه سیستم عامل شرح داده می‌شوند. این تمرین‌ها باید با زبان C و در محیطی که از استاندارد POSIX پشتیبانی می‌کند، نوشته شوند.

پیاده‌سازی یک Thread Pool

در این تمرین یک Thread Pool در زبان C پیاده‌سازی می‌شود. در این پیاده‌سازی باید توابع معرفی شده در فایل taskpool.h تعریف شوند؛ این سرآیند[1] چهار تابع را برای ساختن، آزاد کردن منابع، اضافه کردن یک وظیفه برای اجرا و انتظار برای پایان یک وظیفه را معرفی می‌کند (برای جزئیات بیشتر به فایل نام برده شده رجوع شود). همچنین، برنامه‌هایی برای استفاده از این پیاده‌سازی و آزمایش جنبه‌های مختلفی از آن باید نوشته شود. در انجام این تمرین، مفاهیم و مسائل مهمی که در همروندی مطرح هستند، به کار گرفته می‌شوند.

یک Thread Pool شامل تعدادی ریسمان آماده‌ی پردازش است که وظیفه‌های پردازشی درخواست شده توسط کاربر را انجام می‌دهند. سه مزیت مهم استفاده از Thread Pool به جای استفاده‌ی مستقیم از ریسمان این موارد هستند: حداکثر تعداد ریسمان‌های همزمانی که می‌توانند اجرا شوند محدود می‌شود؛ سربار ساختن و از بین بردن ریسمان‌ها کاهش می‌یابد؛ چون کاربر Thread Pool به صورت مستقیم ریسمان‌ها را مدیریت نمی‌کند، مدیریت ریسمان‌ها ساده‌تر می‌شود.

---

۱ Header

انتظار برای اجرای موفق

در تمرین شماره‌ی پنج، یک دستور می‌بایست تا زمانی تکرار شود که یک بار به صورت موفقیت آمیز اجرا گردد. در این تمرین، تمرین شماره‌ی پنج گسترش داده می‌شود. فهرستی از دستورات (از ورودی استاندارد؛ هر دستور در یک خط) دریافت می‌شوند که باید به صورت موازی اجرا می‌شوند. برنامه در دو حالت کار می‌کند که با یک ورودی تعیین می‌گردد: در حالت اشتراک، هر یک از دستورات مشخص شده باید تکرار شوند تا اینکه یک بار اجرای موفقیت آمیز داشته باشند. در حالت اجتماع، دستورات ورودی به صورت موازی اجرا می‌شوند ولی وقتی یکی از آنها موفقیت آمیز بود، همه‌ی دستورات خاتمه می‌یابند.

پیاده‌سازی یک Process Queue

در این تمرین برنامه‌ای نوشته می‌شود که با خواندن فهرستی از دستورها، آنها را به صورت موازی اجرا نماید با این شرط که در هر زمان، فقط تعداد محدودی از آنها در حال اجرا باشند. حداکثر تعداد دستورهای در حال اجرا توسط یک پارامتر به برنامه داده می‌شود و دستورات ورودی از ورودی استاندارد (هر دستور در یک خط مجزا) خوانده می‌شوند. در مثال زیر، برنامه با چهار دستور همزمان و ده برنامه‌ی ورودی اجرا می‌شود.

```
$ cat input
sleep 1 && echo 0
sleep 2 && echo 1
sleep 3 && echo 2
sleep 4 && echo 3
sleep 2 && echo 4
sleep 2 && echo 5
sleep 2 && echo 6
sleep 2 && echo 7
sleep 2 && echo 8
sleep 2 && echo 9
$ ./processqueue 4 <input
```

مرتب‌سازی موازی

الگوریتم‌های مرتب‌سازی[۱] که از تقسیم و حل[۲] استفاده می‌کنند ورودی‌ها را به دسته‌های کوچک‌تری می‌شکنند و پس از مرتب کردن این دسته‌ها، آنها را با هم ترکیب می‌کنند. در این تمرین برنامه‌ای نوشته می‌شود که تعدادی رشته‌ی ورودی را مرتب سازد (هر رشته در یک خط ورودی مشخص می‌شود): این برنامه با گرفتن پارامتر n، ورودی‌ها را به ۲^n قسمت تقسیم می‌کند و هر یک از این قسمت‌ها را با استفاده از یک ریسمان مجزا به صورت موازی مرتب می‌کند. در نهایت دسته‌های مرتب شده با هم ترکیب می‌شوند.

۱ Sorting
۲ Divide and Conquer

پیاده‌سازی یک نسخه‌ی ساده‌ی Make

در تمرین شماره‌ی چهار، قابلیت‌های ابتدایی ابزار Make معرفی شدند. در این تمرین برنامه‌ای باید نوشته شود که با خواندن فایل Makefile، مشابه ابزار Make، فایل مشخص شده را بسازد. در این تمرین فقط قابلیت‌های معرفی شده در تمرین چهار مورد نظر هستند. در این تمرین باید پس از خواندن قواعد مشخص شده در فایل Makefile و تشکیل درخت وابستگی، با توجه به وجود (یا زمان تغییر فایل‌ها)، وابستگی‌های فایل مشخص شده و خود آن فایل ساخته شود.