۱۰ RDT: پیاده‌سازی یک پروتکل انتقال قابل اعتماد

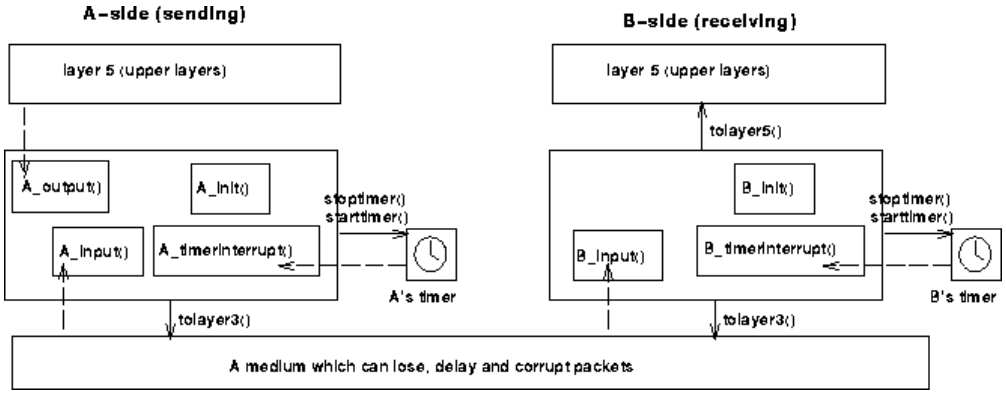
نمای کلی

در این تکلیف برنامه‌نویسی آزمایشگاهی، شما کد سمت فرستنده و گیرنده لایه انتقال را برای پیاده‌سازی یک پروتکل انتقال داده قابل اعتماد ساده خواهید نوشت. این آزمایشگاه باید جالب باشد زیرا پیاده‌سازی شما تفاوت بسیار کمی با آنچه در یک وضعیت دنیای واقعی مورد نیاز است، خواهد داشت. شما کافی است نسخه پروتکل بیت متناوب (Alternating-Bit-Protocol) این آزمایشگاه را با زبان C یا پایتون پیاده سازی کنید. starter code مورد نیاز هر کدام در اختیار شما قرار میگیرد.

از آنجایی که شما احتمالاً ماشین‌های مستقلی (با سیستم‌عاملی که بتوانید آن را تغییر دهید) در اختیار ندارید، کد شما باید در یک محیط سخت‌افزاری/نرم‌افزاری شبیه‌سازی شده اجرا شود. با این حال، رابط برنامه‌نویسی ارائه‌شده به روتین‌های شما، یعنی کدی که موجودیت‌های شما را از بالا و از پایین فراخوانی می‌کند، بسیار نزدیک به آن چیزی است که در یک محیط واقعی UNIX انجام می‌شود. (در واقع، رابط‌های نرم‌افزاری توصیف‌شده در این تکلیف برنامه‌نویسی بسیار واقعی‌تر از فرستنده‌ها و گیرنده‌های حلقه بی‌نهایتی هستند که بسیاری از متون توصیف می‌کنند). توقف/شروع تایمرها نیز شبیه‌سازی شده است، و وقفه‌های تایمر باعث فعال شدن روتین مدیریت تایمر شما می‌شوند.

روتین‌هایی که شما خواهید نوشت

پروسه‌هایی که شما خواهید نوشت برای موجودیت فرستنده (A) و موجودیت گیرنده (B) هستند. فقط انتقال یک‌طرفه داده (از A به B) مورد نیاز است. البته، سمت B مجبور خواهد بود برای تایید (مثبت یا منفی) دریافت داده، بسته‌هایی را به A ارسال کند. روتین‌های شما باید به شکل پروسه‌هایی که در زیر توضیح داده شده‌اند، پیاده‌سازی شوند. این پروسه‌ها توسط پروسه‌هایی که ما نوشته‌ام و یک محیط شبکه را شبیه‌سازی می‌کنند، فراخوانی خواهند شد (و آن‌ها را فراخوانی خواهند کرد). ساختار کلی محیط در شکل ۱ (ساختار محیط شبیه‌سازی شده) نشان داده شده است.



شکل ۱: ساختار کلی محیط

واحد داده‌ای که بین لایه‌های بالایی و پروتکل‌های شما پاس داده می‌شود، یک message (پیام) است که به این صورت تعریف شده است:

```
struct msg {
char data[20];
  };
```

این تعریف، و تمام ساختارهای داده‌ای دیگر و روتین‌های شبیه‌ساز، و همچنین روتین‌های استاب (stub) (یعنی آنهایی که شما باید تکمیل کنید) در فایل prog2.c قرار دارند که بعداً توضیح داده می‌شود. بنابراین، موجودیت فرستنده شما داده‌ها را در تکه‌های (chunk) ۲۰ بایتی از layer5 دریافت خواهد کرد؛ موجودیت گیرنده شما باید تکه‌های ۲۰ بایتی از داده‌های به درستی دریافت شده را به layer5 در سمت گیرنده تحویل دهد.
واحد داده‌ای که بین روتین‌های شما و لایه شبکه پاس داده می‌شود، packet (بسته) است که به این صورت تعریف شده است:

```
 struct pkt {
int seqnum;
int acknum;
```

```
int checksum;
char payload[20];
 };
```

روتین‌های شما فیلد payload را از داده‌های پیامی که از layer5 پایین آمده است، پر خواهند کرد. فیلدهای دیگر بسته توسط پروتکل‌های شما برای اطمینان از تحویل قابل اعتماد، همانطور که در کلاس دیده‌ایم، استفاده خواهند شد.

روتین‌هایی که شما خواهید نوشت در زیر به تفصیل آمده‌اند. همانطور که در بالا ذکر شد، چنین پروسه‌هایی در زندگی واقعی بخشی از سیستم‌عامل خواهند بود و توسط پروسه‌های دیگر در سیستم‌عامل فراخوانی می‌شوند.

- A-output(message)، که در آن message ساختاری از نوع msg است و حاوی داده‌ای برای ارسال به سمت B می‌باشد. این روتین هر زمان که لایه بالایی در سمت فرستنده (A) پیامی برای ارسال داشته باشد، فراخوانی خواهد شد. این وظیفه پروتکل شماست که اطمینان حاصل کند داده‌های موجود در چنین پیامی به ترتیب و به درستی به لایه بالایی سمت گیرنده تحویل داده می‌شوند.

- A-input(packet)، که در آن packet ساختاری از نوع pkt است. این روتین هر زمان که بسته‌ای ارسالی از سمت B (یعنی در نتیجه اجرای tolayer3() توسط یک پروسه در سمت B) به سمت A برسد، فراخوانی خواهد شد. packet بسته (احتمالاً خراب شده) ارسالی از سمت B است.

- A-timerinterrupt() این روتین زمانی که تایمر A منقضی شود (و در نتیجه یک وقفه تایمر ایجاد کند) فراخوانی خواهد شد. شما احتمالاً می‌خواهید از این روتین برای کنترل ارسال مجدد بسته‌ها استفاده کنید. برای نحوه شروع و توقف تایمر، به starttimer() و stoptimer() در زیر مراجعه کنید.

- A-init() این روتین یک بار، قبل از اینکه هر یک از روتین‌های دیگر سمت A شما فراخوانی شوند، فراخوانی خواهد شد. می‌توان از آن برای انجام هرگونه مقداردهی اولیه مورد نیاز استفاده کرد.

- B-input(packet)، که در آن packet ساختاری از نوع pkt است. این روتین هر زمان که بسته‌ای ارسالی از سمت A (یعنی در نتیجه اجرای tolayer3() توسط یک پروسه در سمت A) به سمت B برسد، فراخوانی خواهد شد. packet بسته (احتمالاً خراب شده) ارسالی از سمت A است.

- B-init() این روتین یک بار، قبل از اینکه هر یک از روتین‌های دیگر سمت B شما فراخوانی شوند، فراخوانی خواهد شد. می‌توان از آن برای انجام هرگونه مقداردهی اولیه مورد نیاز استفاده کرد.

رابط‌های نرم‌افزاری

پروسه‌هایی که در بالا توضیح داده شدند، آنهایی هستند که شما خواهید نوشت. ما روتین‌های زیر را نوشته‌ایم که می‌توانند توسط روتین‌های شما فراخوانی شوند:

- starttimer(calling-entity,increment)، که در آن calling-entity یا 0 (برای شروع تایمر سمت A) یا 1 (برای شروع تایمر سمت B) است، و increment یک مقدار float است که نشان‌دهنده مقدار زمانی است که قبل از وقفه تایمر سپری خواهد شد. تایمر A فقط باید توسط روتین‌های سمت A شروع (یا متوقف) شود، و به طور مشابه برای تایمر سمت B. برای اینکه ایده‌ای از مقدار increment مناسب برای استفاده داشته باشید: یک بسته ارسال شده به شبکه به طور متوسط 5 واحد زمانی طول می‌کشد تا به طرف دیگر برسد، زمانی که هیچ پیام دیگری در رسانه وجود ندارد.
- stoptimer(calling-entity)، که در آن calling-entity یا 0 (برای توقف تایمر سمت A) یا 1 (برای توقف تایمر سمت B) است.
- tolayer3(calling-entity,packet)، که در آن calling-entity یا 0 (برای ارسال از سمت A) یا 1 (برای ارسال از سمت B) است، و packet ساختاری از نوع pkt است. فراخوانی این روتین باعث می‌شود بسته به شبکه ارسال شود و مقصد آن موجودیت دیگر باشد.
- tolayer5(calling-entity,message)، که در آن calling-entity یا 0 (برای تحویل به layer 5 در سمت A) یا 1 (برای تحویل به layer 5 در سمت B) است، و message ساختاری از نوع msg است. با انتقال داده یک‌طرفه، شما فقط این روتین را با calling-entity برابر با 1 (تحویل به سمت B) فراخوانی خواهید کرد. فراخوانی این روتین باعث می‌شود داده‌ها به layer 5 پاس داده شوند.

محیط شبکه شبیه‌سازی شده

فراخوانی پروسه tolayer3() بسته‌ها را به رسانه (یعنی به لایه شبکه) ارسال می‌کند. پروسه‌های شما A-input() و B-input() زمانی فراخوانی می‌شوند که بسته‌ای قرار است از رسانه به لایه پروتکل شما تحویل داده شود.
رسانه قادر به خراب کردن و گم کردن بسته‌ها است. ترتیب بسته‌ها را تغییر نخواهد داد. هنگامی که شما پروسه‌های خود و پروسه‌های ما را با هم کامپایل کرده و برنامه حاصل را اجرا می‌کنید، از شما خواسته می‌شود تا مقادیری را در مورد محیط شبکه شبیه‌سازی شده مشخص کنید:

- تعداد پیام‌ها برای شبیه‌سازی. شبیه‌ساز ما (و روتین‌های شما) به محض اینکه این تعداد پیام از layer 5 پایین فرستاده شد، متوقف خواهد شد، صرف نظر از اینکه آیا تمام پیام‌ها به درستی تحویل داده شده‌اند یا خیر.

بنابراین، نیازی نیست نگران پیام‌های تحویل‌داده‌نشده یا ACK نشده‌ای باشید که هنگام توقف شبیه‌ساز هنوز در فرستنده شما هستند. توجه داشته باشید که اگر این مقدار را روی 1 تنظیم کنید، برنامه شما بلافاصله، قبل از تحویل پیام به طرف دیگر، خاتمه می‌یابد. بنابراین، این مقدار باید همیشه بزرگتر از 1 باشد.

- گم شدن (**Loss**). از شما خواسته می‌شود تا احتمال گم شدن بسته را مشخص کنید. مقدار 0.1 به این معنی است که (به طور متوسط) از هر ده بسته یکی گم می‌شود.
- خراب شدن (**Corruption**). از شما خواسته می‌شود تا احتمال خراب شدن بسته را مشخص کنید. مقدار 0.2 به این معنی است که (به طور متوسط) از هر پنج بسته یکی خراب می‌شود. توجه داشته باشید که محتویات فیلدهای payload، sequence، ack یا checksum می‌توانند خراب شوند. بنابراین، checksum شما باید شامل داده‌ها، فیلدهای sequence و ack باشد.
- ردیابی (**Tracing**). تنظیم مقدار ردیابی روی 1 یا 2 اطلاعات مفیدی در مورد آنچه در داخل شبیه‌سازی در حال رخ دادن است چاپ می‌کند (مثلاً چه اتفاقی برای بسته‌ها و تایمرها می‌افتد). مقدار ردیابی 0 این قابلیت را خاموش می‌کند. مقدار ردیابی بزرگتر از 2 انواع پیام‌های عجیبی را نمایش می‌دهد که برای اهداف دیباگ کردن خود شبیه‌ساز ما هستند. مقدار ردیابی 2 ممکن است برای شما در دیباگ کردن کدتان مفید باشد. باید به خاطر داشته باشید که پیاده‌سازان واقعی شبکه‌های زیربنایی ندارند که چنین اطلاعات خوبی در مورد آنچه قرار است برای بسته‌هایشان اتفاق بیفتد، ارائه دهند!
- متوسط زمان بین پیام‌ها از **layer5** فرستنده. شما می‌توانید این مقدار را روی هر مقدار مثبت و غیر صفر تنظیم کنید. توجه داشته باشید که هرچه مقدار کمتری انتخاب کنید، بسته‌ها سریع‌تر به فرستنده شما می‌رسند.

نسخه پروتکل بیت متناوب (**Alternating-Bit-Protocol**) این آزمایشگاه

شما باید پروسه‌های A-output()، A-input()، A-timerinterrupt()، A-init()، B-input() و B-init() را بنویسید که با هم یک انتقال داده یک‌طرفه توقف-و-انتظار (stop-and-wait) (یعنی پروتکل بیت متناوب، که ما در متن کتاب به آن rdt3.0 می‌گوییم) را از سمت A به سمت B پیاده‌سازی کنند. پروتکل شما باید از هر دو پیام ACK و NACK استفاده کند.

شما باید مقدار بسیار بزرگی را برای متوسط زمان بین پیام‌ها از layer5 فرستنده انتخاب کنید، تا فرستنده شما هرگز زمانی که هنوز یک پیام برجسته و تایید نشده دارد که در تلاش برای ارسال آن به گیرنده است، فراخوانی نشود. ما پیشنهاد می‌کنیم مقداری حدود 1000 را انتخاب کنید. همچنین باید در فرستنده خود بررسی کنید تا مطمئن شوید زمانی که A-output() فراخوانی می‌شود، هیچ پیامی در حال حاضر در حال انتقال نیست. اگر وجود دارد، می‌توانید به سادگی داده‌های پاس داده شده به روتین A-output() را نادیده بگیرید (drop کنید).

شما باید پروسه‌های خود را در فایلی به نام prog2.c قرار دهید. شما به نسخه اولیه این فایل، حاوی روتین‌های شبیه‌سازی که ما برای شما نوشته‌ایم، و استاب‌های (stubs) پروسه‌های شما، نیاز خواهید داشت. می‌توانید این برنامه را از صفحه درس دریافت کنید.

این آزمایشگاه را می‌توان بر روی هر ماشینی که از C پشتیبانی می‌کند، تکمیل کرد. این برنامه از هیچ ویژگی UNIX استفاده نمی‌کند. (شما می‌توانید به سادگی فایل prog2.c را به هر ماشین و سیستم‌عاملی که انتخاب می‌کنید کپی کنید).

ما توصیه می‌کنیم که یک لیست کد، یک سند طراحی، و خروجی نمونه را تحویل دهید. برای خروجی نمونه شما، پروسه‌های شما می‌توانند هر زمان که رویدادی در فرستنده یا گیرنده شما رخ می‌دهد (ورود یک پیام/بسته، یا یک وقفه تایمر) و همچنین هر اقدامی که در پاسخ انجام می‌شود، پیامی را چاپ کنند. شاید بخواهید خروجی را برای یک اجرا تا نقطه‌ای (تقریباً) که 10 پیام به درستی در گیرنده ACK شده‌اند، با احتمال گم شدن 0.1 و احتمال خرابی 0.3 و سطح ردیابی 2 تحویل دهید. شاید بخواهید پرینت خروجی خود را با یک خودکار رنگی حاشیه‌نویسی کنید که نشان می‌دهد چگونه پروتکل شما به درستی از گم شدن و خرابی بسته بازیابی شده است.

اطمینان حاصل کنید که "نکات مفید" (helpful hints) این آزمایشگاه را می‌خوانید.

نسخه **Go-Back-N** این آزمایشگاه (امتیازی)

شما باید رویه‌های A_output()، A_input()، A_timerinterrupt()، A_init()، B_input() و B_init() را بنویسید که روی هم رفته یک انتقال داده یک‌طرفه Go-Back-N را از فرستنده A به گیرنده B با اندازه پنجره ۸ پیاده‌سازی کنند. پروتکل شما باید از پیام‌های ACK و NACK استفاده کند. برای اطلاعات در مورد نحوه دستیابی به شبیه‌ساز شبکه، به نسخه پروتکل بیت متناوب (alternating-bit-protocol) این آزمایشگاه در بالا مراجعه کنید. ما اکیداً توصیه می‌کنیم که ابتدا آزمایشگاه آسان‌تر (بیت متناوب - Alternating Bit) را پیاده‌سازی کنید و سپس کد خود را برای پیاده‌سازی آزمایشگاه سخت‌تر (Go-Back-N) گسترش دهید. باور کنید - این کار اتلاف وقت نخواهد بود! با این حال، چند ملاحظه جدید برای کد Go-Back-N شما (که در مورد پروتکل بیت متناوب صدق نمی‌کنند) عبارتند از:

- A_output(message)، که در آن message ساختاری از نوع msg است و حاوی داده‌هایی برای ارسال به گیرنده B می‌باشد. روال A_output() شما اکنون گاهی اوقات زمانی فراخوانی می‌شود که پیام‌های در حال انتظار و تأیید نشده‌ای در رسانه وجود دارد - این بدان معناست که شما باید چندین پیام را در فرستنده خود بافر کنید. همچنین، به دلیل ماهیت Go-Back-N، به بافرینگ در فرستنده خود نیاز خواهید داشت: گاهی اوقات فرستنده شما فراخوانی می‌شود اما نمی‌تواند پیام جدید را ارسال کند زیرا پیام جدید خارج از پنجره قرار می‌گیرد.

- به جای اینکه نگران بافر کردن تعداد نامحدودی از پیام‌ها باشید، اشکالی ندارد که تعداد محدود و مشخصی بافر در فرستنده خود داشته باشید (مثلاً برای ۵۰ پیام) و اگر هر ۵۰ بافر در یک لحظه در حال استفاده بودند، فرستنده شما به سادگی abort کند (تسلیم شود و خارج شود) (توجه: با استفاده از مقادیر داده شده در ادامه، این اتفاق هرگز نباید رخ دهد!) البته در «دنیای واقعی»، باید راه‌حل ظریف‌تری برای مشکل بافر محدود پیدا کرد!

- A_timerinterrupt() این روال زمانی فراخوانی می‌شود که تایمر A منقضی شود (و در نتیجه یک وقفه تایمر ایجاد کند). به یاد داشته باشید که شما فقط یک تایمر دارید و ممکن است بسته‌های در حال انتظار و تأیید نشده زیادی در رسانه داشته باشید، بنابراین باید کمی در مورد نحوه استفاده از این تایمر واحد فکر کنید.

برای توضیحات کلی در مورد آنچه ممکن است بخواهید تحویل دهید، به نسخه پروتکل بیت متناوب (Alternating-bit-protocol) این آزمایشگاه در بالا مراجعه کنید. شاید بهتر باشد خروجی اجرایی را تحویل دهید که به اندازه کافی طولانی بوده تا حداقل ۲۰ پیام با موفقیت از فرستنده به گیرنده منتقل شده باشند (یعنی فرستنده برای این پیام‌ها ACK دریافت کند)، با احتمال گم شدن 0.2، احتمال خرابی 0.2، سطح ردیابی (trace level) 2، و میانگین زمان بین ورودها 10. همچنین می‌توانید بخش‌هایی از خروجی چاپی خود را با یک خودکار رنگی حاشیه‌نویسی کنید که نشان دهد چگونه پروتکل شما به درستی از گم شدن و خرابی بسته‌ها بازیابی شده است.

نکته:

می‌توانید انتقال دوطرفه پیام‌ها را پیاده‌سازی کنید. در این حالت، موجودیت‌های A و B هم به عنوان فرستنده و هم به عنوان گیرنده عمل می‌کنند. همچنین می‌توانید تأییدیه‌ها را روی بسته‌های داده piggyback کنید (یا می‌توانید این کار را نکنید). برای اینکه شبیه‌ساز من پیام‌ها را از لایه ۵ به روال B_output() شما تحویل دهد، باید مقدار تعریف شده BIDIRECTIONAL را از 0 به 1 تغییر دهید.

نکات مفید و موارد مشابه

- چکسام کردن (**Checksumming**). شما می‌توانید از هر رویکردی که برای checksumming می‌خواهید استفاده کنید. به یاد داشته باشید که شماره توالی و فیلد ack نیز می‌توانند خراب شوند. ما یک checksum شبیه به TCP را پیشنهاد می‌کنیم، که شامل مجموع مقادیر (صحیح) فیلدهای sequence و ack است، که به مجموع کاراکتر به کاراکتر فیلد payload بسته اضافه شده است (یعنی با هر کاراکتر طوری رفتار کنید که گویی یک عدد صحیح 8 بیتی است و فقط آنها را با هم جمع کنید).

- توجه داشته باشید که هر «وضعیت» مشترک بین پروسه‌های شما باید به شکل متغیرهای سراسری باشد. همچنین توجه داشته باشید که هر اطلاعاتی که پروسه‌های شما باید از یک فراخوانی به فراخوانی بعدی ذخیره کنند، باید یک متغیر سراسری (یا ایستا) نیز باشد. به عنوان مثال، پروسه‌های شما باید یک کپی از یک بسته را برای ارسال مجدد احتمالی نگه دارند. احتمالاً ایده خوبی است که چنین ساختار داده‌ای یک متغیر سراسری در کد شما باشد. با این حال، توجه داشته باشید که اگر یکی از متغیرهای سراسری شما توسط سمت فرستنده استفاده می‌شود، آن متغیر نباید توسط موجودیت سمت گیرنده قابل دسترسی باشد، زیرا در زندگی واقعی، موجودیت‌های ارتباطی که فقط از طریق یک کانال ارتباطی به هم متصل هستند، نمی‌توانند متغیرهای سراسری را به اشتراک بگذارند.

- یک متغیر سراسری float به نام time وجود دارد که می‌توانید از داخل کد خود به آن دسترسی داشته باشید تا در پیام‌های تشخیصی به شما کمک کند.

- ساده شروع کنید (**START SIMPLE**). احتمالات گم شدن و خرابی را صفر تنظیم کنید و روتین‌های خود را آزمایش کنید. بهتر از آن، پروسه‌های خود را برای حالت بدون گم شدن و بدون خرابی طراحی و پیاده‌سازی کنید و ابتدا آنها را به کار بیندازید. سپس حالت یکی از این احتمالات غیر صفر را مدیریت کنید، و در نهایت هر دو را غیر صفر در نظر بگیرید.

- دیباگ کردن (**Debugging**). ما توصیه می‌کنیم که سطح ردیابی را روی 2 تنظیم کنید و در حین دیباگ کردن پروسه‌های خود، printfهای زیادی در کد خود قرار دهید.

- اعداد تصادفی (**Random Numbers**). شبیه‌ساز با استفاده از یک تولیدکننده اعداد تصادفی، گم شدن و خطاهای بسته را ایجاد می‌کند. تجربه گذشته ما این است که تولیدکننده‌های اعداد تصادفی می‌توانند از یک ماشین به ماشین دیگر بسیار متفاوت باشند. ممکن است نیاز باشد کد تولید اعداد تصادفی را در شبیه‌سازی که ما به شما عرضه کرده‌ایم، تغییر دهید. روتین‌های شبیه‌سازی ما یک تست دارند تا ببینند آیا تولیدکننده اعداد تصادفی روی ماشین شما با کد ما کار می‌کند یا خیر. اگر پیغام خطایی دریافت کردید:

```
It is likely that random number generation on your machine is different
from what this emulator expects. Please take a look at the routine
jimsrand() in the emulator code. Sorry.
```

در این صورت خواهید دانست که باید به نحوه تولید اعداد تصادفی در روتین jimsrand() نگاهی بیندازید؛ به توضیحات (comments) در آن روتین مراجعه کنید.