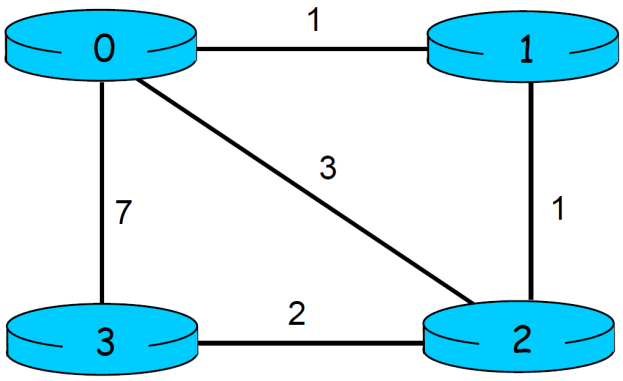
سوال ۱۱

مرور کلی

در این تکلیف برنامه‌نویسی، شما مجموعه‌ای از پروسه‌های «توزیع‌شده» را خواهید نوشت که یک مسیریابی بردار فاصله (distance vector) ناهمگام توزیع‌شده را برای شبکه نشان داده شده در زیر پیاده‌سازی می‌کنند.



تکلیف اصلی

پروسه‌هایی که خواهید نوشت

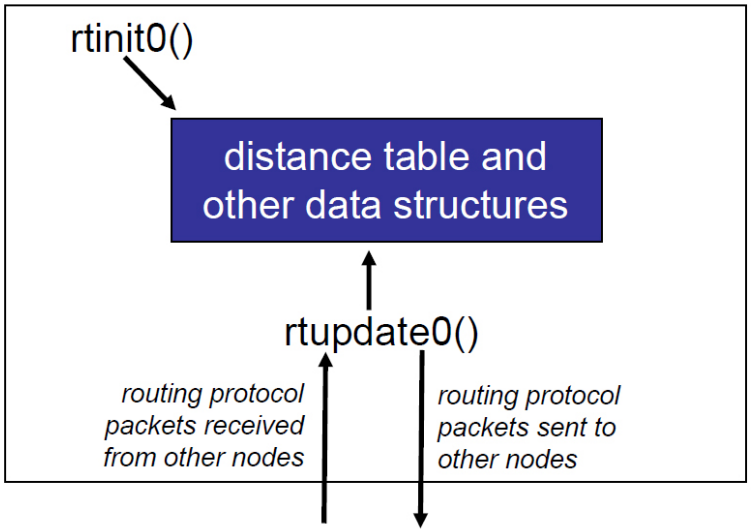
برای بخش اصلی تکلیف، شما باید پروسه‌های زیر را بنویسید که به صورت ناهمگام در محیط شبیه‌سازی شده‌ای که ما برای این تکلیف نوشته‌ایم، «اجرا» خواهند شد.
برای گره 0، شما پروسه‌های زیر را خواهید نوشت:

- rtinit0() این پروسه یک بار در ابتدای شبیه‌سازی فراخوانی خواهد شد. rtinit0() هیچ آرگومانی ندارد. این پروسه باید جدول فاصله (distance table) (شکل زیر را ببینید) در گره 0 را مقداردهی اولیه کند تا هزینه‌های مستقیم 1، 3 و 7 به ترتیب به گره‌های 1، 2 و 3 را منعکس کند. در شکل بالا، تمام پیوندها دوطرفه هستند و هزینه‌ها در هر دو جهت یکسان می‌باشند. پس از مقداردهی اولیه جدول فاصله و هر ساختار داده دیگری که مورد نیاز پروسه‌های گره 0 شما باشد، باید هزینه‌های مسیرهای با کمترین هزینه خود، یعنی بردار فاصله (distance vector) خود را، به تمام گره‌های دیگر شبکه، برای همسایگان مستقیماً متصل خود (در این مورد، 1، 2 و 3) ارسال کند. این اطلاعات کمترین هزینه در یک بسته مسیریابی (routing packet) با فراخوانی پروسه tolayer2()، همانطور که در زیر توضیح داده شده، به گره‌های همسایه ارسال می‌شود. قالب بسته مسیریابی نیز در زیر شرح داده شده است.

- rtupdate0(struct rtpkt *rcvdpkt) این پروسه زمانی فراخوانی می‌شود که گره 0 یک بسته مسیریابی دریافت کند که توسط یکی از همسایگان مستقیماً متصل به آن ارسال شده است. پارامتر rcvdpkt* یک اشاره‌گر به بسته دریافت شده است. rtupdate0() «قلب» الگوریتم بردار فاصله است. مقادیری که در یک بسته مسیریابی از گره دیگری مانند j دریافت می‌کند، حاوی هزینه‌های کوتاه‌ترین مسیر فعلی j (بردار فاصله j) به تمام گره‌های دیگر شبکه است. rtupdate0() از این مقادیر دریافت شده برای به‌روزرسانی «جدول فاصله» خود استفاده می‌کند. جدول فاصله درون هر گره، ساختار داده اصلی مورد استفاده توسط الگوریتم بردار فاصله است. برای شما راحت خواهد بود که جدول فاصله را به عنوان یک آرایه 4 در 4 از int تعریف کنید، که در آن ورودی [i,j] در جدول فاصله در گره 0، هزینه محاسبه شده فعلی گره 0 به گره i از طریق همسایه مستقیم j است؛ گره 0 ورودی‌های جدول فاصله خود را برای ستون j پس از دریافت بردار فاصله j (دوباره) محاسبه می‌کند. اگر 0 مستقیماً به j متصل نباشد، می‌توانید این ورودی را نادیده بگیرید. ما از این قرارداد استفاده خواهیم کرد که مقدار صحیح 999 «بی‌نهایت» است.

همانطور که توسط الگوریتم بردار فاصله که در کلاس و در متن درس مطالعه کردیم مشخص شده است، اگر کمترین هزینه خود گره 0 به گره دیگری در نتیجه این به‌روزرسانی تغییر کند، گره 0 همسایگان مستقیماً متصل خود را از این تغییر در کمترین هزینه با ارسال یک بسته مسیریابی به آنها مطلع می‌سازد. به یاد بیاورید که در الگوریتم بردار فاصله، فقط گره‌های مستقیماً متصل بسته‌های مسیریابی را مبادله می‌کنند. بنابراین گره‌های 1 و 2 با یکدیگر ارتباط برقرار می‌کنند، اما گره‌های 1 و 3 با یکدیگر ارتباط برقرار نخواهند کرد.

پروسه‌های مشابهی برای گره‌های 1، 2 و 3 تعریف شده‌اند. بنابراین، شما در مجموع 8 پروسه خواهید نوشت: rtinit0()، rtinit1()، rtinit2()، rtinit3()، rtupdate0()، rtupdate1()، rtupdate2()، rtupdate3()



واسط‌های نرم‌افزاری

پروسه‌های توصیف شده در بالا آنهایی هستند که شما خواهید نوشت. ما پروسه‌های زیر را نوشته‌ایم که می‌توانند توسط پروسه‌های شما فراخوانی شوند:

- tolayer2(struct rtpkt pkt2send) که در آن rtpkt ساختار زیر است، که قبلاً برای شما تعریف شده است. پروسه tolayer2() در فایل distance_vector.c تعریف شده است.

```
extern struct rtpkt {
  int sourceid;   /* id of node sending this pkt, 0, 1, 2, or 3 */
  int destid;     /* id of router to which pkt being sent
                     (must be an immediate neighbor) */
  int mincost[4];   /* min cost to node 0 ... 3 */
  };
```

توجه داشته باشید که به tolayer2() یک ساختار پاس داده می‌شود، نه یک اشاره‌گر به ساختار.

- printdt0() جدول فاصله را برای گره 0 به صورت زیبا چاپ (pretty print) می‌کند. به آن یک اشاره‌گر به ساختاری از نوع distance_table پاس داده می‌شود. printdt0() و اعلان ساختار برای جدول فاصله گره 0 در فایل node0.c تعریف شده‌اند. پروسه‌های مشابه چاپ زیبا برای شما در فایل‌های node1.c، node2.c و node3.c تعریف شده‌اند.

محیط شبکه شبیه‌سازی شده

پروسه‌های شما rtinit0()، rtinit1()، rtinit2()، rtinit3() و rtupdate0()، rtupdate1()، rtupdate2()، rtupdate3() بسته‌های مسیریابی (که قالب آنها در بالا توضیح داده شد) را به محیط ارسال می‌کنند. محیط، بسته‌ها را به ترتیب و بدون از دست دادن به مقصد مشخص شده تحویل می‌دهد. فقط گره‌های مستقیماً متصل می‌توانند ارتباط برقرار کنند. تأخیر بین فرستنده و گیرنده متغیر (و ناشناخته) است.
وقتی شما پروسه‌های خود و پروسه‌های من را با هم کامپایل می‌کنید و برنامه حاصل را اجرا می‌کنید، از شما خواسته می‌شود که فقط یک مقدار را در مورد محیط شبکه شبیه‌سازی شده مشخص کنید:

- Tracing. تنظیم مقدار tracing روی 1 یا 2 اطلاعات مفیدی در مورد آنچه در داخل شبیه‌سازی در حال رخ دادن است (مثلاً، چه اتفاقی برای بسته‌ها و تایمرها می‌افتد) چاپ می‌کند. مقدار tracing برابر 0 این قابلیت را

خاموش می‌کند. مقدار tracing بزرگتر از 2 انواع پیام‌های عجیب و غریبی را نمایش می‌دهد که برای اهداف اشکال‌زدایی شبیه‌ساز خود من هستند.

یک مقدار tracing برابر 2 ممکن است برای شما در اشکال‌زدایی 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!

تکلیف اصلی

شما باید پروسه‌های ()rtinit0، ()rtinit1، ()rtinit2، ()rtinit3 و ()rtupdate0، ()rtupdate1، ()rtupdate2، ()rtupdate3 را بنویسید که با هم یک محاسبه توزیع‌شده و ناهمگام از جداول فاصله را برای توپولوژی و هزینه‌های نشان داده شده در شکل اول پیاده‌سازی کنند.

شما باید پروسه‌های خود را برای گره‌های 0 تا 3 در فایل‌هایی به نام node0.c، .... node3.c قرار دهید. شما مجاز به تعریف هیچ متغیر سراسری (global variables) که خارج از یک فایل C معین قابل مشاهده باشد، نیستید (به عنوان مثال، هر متغیر سراسری که در node0.c تعریف می‌کنید، فقط باید در داخل node0.c قابل دسترسی باشد). این به منظور وادار کردن شما به رعایت قراردادهای کدنویسی است که اگر واقعاً پروسه‌ها را در چهار گره مجزا اجرا می‌کردید، مجبور به اتخاذ آن‌ها بودید. برای کامپایل پروسه‌های خود: cc distance_vector.c node0.c node1.c node2.c node3.c. نسخه‌های اولیه این فایل‌ها در اینجا هستند: node0.c، node1.c، node2.c، node3.c. شما می‌توانید یک کپی از فایل distance_vector.c را از پوشه starter_code در صفحه ایلرن درس بردارید. شما می‌توانید پروسه‌های node*.c و distance_vector.c را که به طور خاص برای linux اصلاح شده‌اند، از همان پوشه بردارید.

این تکلیف می‌تواند بر روی هر ماشینی که از C، C++، یا JAVA پشتیبانی می‌کند، تکمیل شود. این تکلیف از هیچ ویژگی UNIX استفاده نمی‌کند.

برای خروجی نمونه شما، پروسه‌های شما باید هر زمان که پروسه‌های ()rtinit0، ()rtinit1، ()rtinit2، ()rtinit3 یا ()rtupdate0، ()rtupdate1، ()rtupdate2، ()rtupdate3 شما فراخوانی می‌شوند، پیامی را چاپ کنند که زمان (از طریق متغیر سراسری من clocktime در دسترس است) را نشان دهد. برای ()rtupdate0، ()rtupdate1، ()rtupdate2، ()rtupdate3 شما باید هویت فرستنده بسته مسیریابی که به پروسه شما پاس داده می‌شود، اینکه آیا جدول فاصله به‌روزرسانی شده است یا نه، محتویات جدول فاصله، و توصیفی از هر پیامی که در نتیجه هرگونه به‌روزرسانی جدول فاصله به گره‌های همسایه ارسال می‌شود را چاپ کنید.

خروجی نمونه باید یک لیست خروجی با مقدار TRACE برابر 2 باشد. جدول فاصله نهایی تولید شده در هر گره را هایلایت کنید. برنامه شما تا زمانی که هیچ بسته مسیریابی دیگری در حال انتقال در شبکه نباشد، اجرا خواهد شد،

در آن نقطه شبیه‌ساز ما خاتمه می‌یابد.
توجه داشته باشید که یک سند طراحی کوتاه مورد نیاز است. شما باید هر فرآیند خود را طوری برنامه‌ریزی کنید که هرگاه اقدامی انجام می‌دهد (مثلاً، ارسال یا دریافت پیام، تشخیص پایان ورودی، و غیره)، یک عبارت آموزنده چاپ کند تا بتوانید ببینید که فرآیندهای شما به درستی کار می‌کنند (یا نه!). این همچنین به TA اجازه می‌دهد تا از این خروجی تشخیص دهد که آیا فرآیندهای شما به درستی کار می‌کنند یا خیر. شما باید اسکرین‌شات‌ها (یا محتوای فایل، اگر فرآیند شما در حال نوشتن در یک فایل است) از این پیام‌های آموزنده را تحویل دهید.

امتیاز اضافی

شما باید دو پروسه، rtlinkhandler0(int linkid, int newcost) و rtlinkhandler1(int linkid, int newcost) را بنویسید، که اگر (و زمانی که) هزینه پیوند بین 0 و 1 تغییر کند، فراخوانی خواهند شد. این پروسه‌ها باید به ترتیب در فایل‌های node0.c و node1.c تعریف شوند. به این پروسه‌ها نام (id) گره همسایه در طرف دیگر پیوندی که هزینه‌اش تغییر کرده، و هزینه جدید پیوند پاس داده خواهد شد. توجه داشته باشید که وقتی هزینه یک پیوند تغییر می‌کند، این پروسه‌ها باید جدول فاصله را به‌روزرسانی کنند و ممکن است (یا نباشد) مجبور شوند بسته‌های مسیریابی به‌روزرسانی شده را به گره‌های همسایه ارسال کنند.
برای تکمیل بخش پیشرفته تکلیف، شما باید مقدار ثابت LINKCHANGES (خط 3 در distance_vector.c) را به 1 تغییر دهید. جهت اطلاع، هزینه پیوند از 1 به 20 در زمان 10000 تغییر خواهد کرد و سپس در زمان 20000 به 1 باز خواهد گشت. پروسه‌های شما در این زمان‌ها فراخوانی خواهند شد.
ما مجدداً اکیداً توصیه می‌کنیم که ابتدا تکلیف اصلی را پیاده‌سازی کنید و سپس کد خود را برای پیاده‌سازی تکلیف امتیاز اضافی گسترش دهید. این کار اتلاف وقت نخواهد بود.

نسخه JAVA این تکلیف

مستندات بالا پروژه را با جزئیات توصیف می‌کند. در اینجا ما پیوندی به کد مورد نیاز برای انجام تکلیف در JAVA ارائه می‌دهیم. مطمئن شوید که مطالب بالا را درک کرده‌اید.
کد JAVA که نیاز دارید را می‌توانید در اینجا پیدا کنید. در اینجا فایل‌های جداگانه‌ای که نیاز خواهید داشت، آمده‌اند:

- Entity.java, Entity0.java, Entity1.java, Entity2.java, Entity3.java, NetworkSimulator.java, Event.java, Packet.java, EventList.java, EventListImpl.java, Project.java

شما سازنده‌های (constructors) Entity0.java، Entity1.java، Entity2.java، و Entity3.java را خواهید نوشت که مشابه rtinit0()، rtinit1()، rtinit2() و rtinit3() در نسخه C هستند. شما همچنین باید متدهای