به نام خدا

شبکه‌های کامپیوتری

تمرین کامپیوتری شماره سه

اعضای گروه:

مرضیه باقری‌نیا امیری / 810197682

آبتین هیدجی / 810197607

بهار 1400

➢ ماژول‌ها و متدها

- **ماژول LoadBalancer**: این ماژول برای مدیریت کل برنامه تدارک دیده شده و در واقع مدیریت ورودی‌ها و ارسال آن به Switch و یا Systemها را بر عهده دارد؛ متدهای این ماژول به شرح زیر هستند:

- **LoadBalancer (Constructor)**: قبل از شروع برنامه، به تعداد ثابتی، unnamed pipe می‌سازیم و آن‌ها را در آرایه pipefds ذخیره می‌کنیم تا در صورت ایجاد Connection آن را به Componentها assign کنیم. این کار برای آن است که ساختار شبکه به واقعیت نزدیک‌تر باشد و تعداد محدودی Component را پذیرا باشد. پس از ساخت pipeها با صدا زدن متد ()read_input فرایند دریافت ورودی از کاربر را آغاز می‌کنیم.
- **Read_input**: تا زمانی که کاربر دستور exit را وارد ننماید، برنامه از ورودی دستور می‌خواند و آن را به متد ()command_handler برای parse و اجرا پاس می‌دهد.
- **command_handler**: در این متد که در واقع متد اصلی این ماژول می‌باشد، با توجه به ورودی کاربر، تصمیم گرفته می‌شود که چه Action انجام گیرد؛ انواع دستورات (word اول آن‌ها)، فرمت قابل قبول آن‌ها و فرایندی که برای هر یک در این ماژول انجام می‌شود، به شرح زیر است:
  - **exit**: اگر کاربر این دستور را وارد کند، متد ()exit_all_components صدا زده می‌شود تا دستور اتمام را به تمامی Componentهای شبکه ارسال کند و سپس با true کردن flag که برای مداومت برنامه در خواند ورودی تعبیه شده، برنامه را از loop خارج می‌کند.
  - **Switch**: اگر کاربر این کلمه را به عنوان کلمه اول دستورش وارد نماید، به معنی این است که قصد ساختن یک Switch را دارد. فرمت کامل این دستور به شکل زیر است:

Switch number_of_ports switch_name

در ابتدا چک می‌کنیم که فرمت ورودی به شکل بالا باشد، اگر نبود با پیغام Bad request آن را به اطلاع کاربر می‌رسانیم؛ سپس بررسی می‌کنیم که switch_name وارد شده تکراری نباشد (تا به حال در سامانه ثبت نشده باشد) و اگر تکراری بود، با پیغام Duplicate name آن را به اطلاع کاربر می‌رسانیم؛ اگر هیچکدام از این اتفاق‌ها رخ نداد، نوبت به ساخت Switch می‌رسد. ابتدا برای آنکه بتوانیم با Switch ساخته شده ارتباط برقرار کنیم، یک unnamed pipe از pipefds به آن اختصاص می‌دهیم و اطلاعات آن pipe را در vector با نام switch_pipe پوش می‌کنیم. برای راحتی کار در ادامه، در map با نام switch_index ایندکس متناظر با این switch و name آن را به صورت <name: index> ذخیره می‌نماییم تا در ارجاعات بعدی راحت باشیم. در مرحله بعدی باید اطلاعات لازم برای Switch را بر روی pipe مخصوص به آن قرار دهیم؛ به این منظور از fill_pipe استفاده می‌کنیم. پس از آن، با استفاده از تابع fork_component یک fork انجام می‌دهیم؛ ادامه فرایند fork در متد مربوط توضیح داده خواهد شد. پس از آنکه یک فرایند از نوع Switch را بالا آوردیم، برای آنکه بتوانیم پاسخ‌هایی که Switch به دستورات ما می‌دهد را بشنویم، با استفاده از متد

create_namedPipe یک namedPipe با استفاده از pid فرایند مربوط به Switch تازه ساخته شده، می‌سازیم. سپس منتظر پاسخ Switch تازه ساخته شده می‌مانیم؛ اگر پیام ارسالی S بود، یعنی فرایند به درستی انجام شده و اگر F بود یعنی Switch ما به درست ساخته نشده است و باید تمامی آنچه که انجام دادیم را بی‌اثر نماییم.

نکته خیلی مهم: با توجه به اینکه از کدهای پروژه قبلی استفاده شد، تمامی Switchها، همان Routerهای مورد نظر در این پروژه هستند.

- **System:** اگر کاربر این کلمه را به عنوان کلمه اول دستورش وارد نماید، به معنی این است که قصد ساختن یک System را دارد. فرمت کامل این دستور به شکل زیر است:

System system _name

در ابتدا چک می‌کنیم که فرمت ورودی به شکل بالا باشد، اگر نبود با پیغام Bad request آن را به اطلاع کاربر می‌رسانیم؛ سپس بررسی می‌کنیم که system _name وارد شده تکراری نباشد (تا به حال در سامانه ثبت نشده باشد) و اگر تکراری بود، با پیغام Duplicate name آن را به اطلاع کاربر می‌رسانیم؛ اگر هیچکدام از این اتفاق‌ها رخ نداد، نوبت به ساخت System می‌رسد. ابتدا برای آنکه بتوانیم با System ساخته شده ارتباط برقرار کنیم، یک unnamed pipe از pipefds به آن اختصاص می‌دهیم و اطلاعات آن pipe را در vector با نام system_pipe پوش می‌کنیم. برای راحتی کار در ادامه، در map با نام system_index ایندکس متناظر با این System و name آن را به صورت <name: index> ذخیره می‌نماییم تا در ارجاعات بعدی راحت باشیم. در مرحله بعدی باید اطلاعات لازم برای System را بر روی pipe مخصوص به آن قرار دهیم؛ به این منظور از fill_pipe استفاده می‌کنیم. پس از آن، با استفاده از تابع fork_component یک fork انجام می‌دهیم؛ ادامه فرایند fork در متد مربوط توضیح داده خواهد شد. پس از آنکه یک فرایند از نوع Switch را بالا آوردیم، برای آنکه بتوانیم پاسخ‌هایی که System به دستورات ما می‌دهد را بشنویم، با استفاده از متد create_namedPipe یک namedPipe با استفاده از pid فرایند مربوط به System تازه ساخته شده، می‌سازیم. سپس منتظر پاسخ System تازه ساخته شده می‌مانیم؛ اگر پیام ارسالی S بود، یعنی فرایند به درستی انجام شده و اگر F بود یعنی System ما به درست ساخته نشده است و باید تمامی آنچه که انجام دادیم را بی‌اثر نماییم.

نکته خیلی مهم: با توجه به اینکه از کدهای پروژه قبلی استفاده شد، تمامی Systemها، همان Clientهای مورد نظر در این پروژه هستند.

- **Connect:** اگر کاربر این کلمه را به عنوان کلمه اول دستورش وارد نماید، به معنی این است که قصد دارد یک System را به یک Switch متصل نماید. فرمت کامل این دستور به شکل زیر است:
- Connect system_name switch_name

در ابتدا چک می‌کنیم که فرمت ورودی به شکل بالا باشد، اگر نبود با پیغام Bad request آن را به اطلاع کاربر می‌رسانیم؛ سپس بررسی می‌کنیم که System فعالی با system _name وارد شده و Switch با Switch_name وارد شده در شبکه موجود باشد و اگر هر یک از این دو نبود، با پیغام Bad request آن را به اطلاع کاربر می‌رسانیم؛ اگر هیچکدام از این اتفاق‌ها رخ نداد، نوبت به ایجاد Connection می‌رسد. به این منظور لازم است تا هر طرف ارتباط یک unnamed_pipe مربوط به خودش را برای ارسال پیام داشته باشد و طرف دیگر از آدرس آن برای دریافت پیام مطلع باشد. برا آماده‌سازی این pipeها از متد ()prepare_connect_message استفاده می‌کنیم؛ ابتدا pipe مربوط به switch را با دستور Connect برای Switch ارسال می‌کنیم. Switch اگر Port خالی داشته باشد، Connection را از طرف خودش ایجاد می‌نماید و پیام S را ارسال می‌کند و اگر هم پورت خالی نداشته باشد پیام F را ارسال می‌نماید. اگر پاسخ S بود، pipe مربوط به system را با دستور Connect برای System ارسال می‌کنیم؛ System اقدامات لازم برای اتصال را انجام می‌دهد و اگر تمامی آنها موفقیت‌آمیز بود، S ارسال می‌کند و در غیر اینصورت F ارسال می‌کند. در صورت S بودن، اتصال با موفقیت ایجاد شده است.

- **Connect_S**: اگر کاربر این کلمه را به عنوان کلمه اول دستورش وارد نماید، به معنی این است که قصد دارد یک Switch را به یک Switch دیگر متصل نماید. فرمت کامل این دستور به شکل زیر است:

Connect switch_name#01 switch_name#02

در ابتدا چک می‌کنیم که فرمت ورودی به شکل بالا باشد، اگر نبود با پیغام Bad request آن را به اطلاع کاربر می‌رسانیم؛ سپس بررسی می‌کنیم که Switchهای فعالی با switch_name#01 و switch_name#02 وارد شده در شبکه موجود باشد و اگر هر یک از این دو نبود، با پیغام Bad request آن را به اطلاع کاربر می‌رسانیم؛ سپس باید اتصال را برقرار نماییم؛ به این منظور لازم است تا هر طرف ارتباط یک unnamed_pipe مربوط به خودش را برای ارسال پیام داشته باشد و طرف دیگر از آدرس آن برای دریافت پیام مطلع باشد. برا آماده‌سازی این pipeها از متد ()prepare_connect_message استفاده می‌کنیم؛ ابتدا pipe مربوط به switch دوم که در واقع مقصد است را با دستور Connect برای Switch ارسال می‌کنیم. Switch اگر Port خالی داشته باشد، Connection را از طرف خودش ایجاد می‌نماید و پیام S را ارسال می‌کند و اگر هم پورت خالی نداشته باشد پیام F را ارسال می‌نماید. اگر پاسخ S بود، pipe مربوط به Switch اول که مبدا است را با دستور Connect برای Switch ارسال می‌کنیم؛ Switch اقدامات لازم برای اتصال را انجام می‌دهد و اگر تمامی آنها موفقیت‌آمیز بود، S ارسال می‌کند و در غیر اینصورت F ارسال می‌کند. در صورت S بودن، اتصال با موفقیت ایجاد شده است.

- **Send**: اگر کاربر این کلمه را به عنوان کلمه اول دستورش وارد نماید، به معنی این است که یک System قصد دارد یک فایل را برای یک گروه از Systemهای دیگر به صورت multicast ارسال نماید. فرمت کامل این دستور به شکل زیر است:

Send System_Name#src Group_System_Name#dest filename

در ابتدا چک می‌کنیم که فرمت ورودی به شکل بالا باشد، اگر نبود با پیغام Bad request آن را به اطلاع کاربر می‌رسانیم؛ سپس بررسی می‌کنیم که System فعالی با نام سیستم Src وارد شده در شبکه موجود باشد و سپس وجود گروهی با نام وارد شده را بررسی می‌کنیم؛ اگر هریک موجود نباشند، با پیغام Wrong source or destination! موضوع را به کاربر اعلام می‌کنیم. اگر هیچکدام از این اتفاق‌ها رخ نداد، نوبت به ارسال پیام می‌رسد. در این مرحله تنها وظیفه LoadBalancer آن است که دستور Send را با همان فرمتی که گرفته برای System فرستنده ارسال کند تا او فرایند ارسال را آغاز نماید.

- **Sync**: اگر کاربر این کلمه را به عنوان کلمه اول دستورش وارد نماید، به معنی این است که قرار است Distance Vector هر روتر را بسازیم. برای اینکار، لیست تمامی Systemهای موجود در سیستم را برای همه Systemها ارسال می‌کنیم تا فرایند ساخت جدول‌های مربوطه آغاز شود. جزئیات الگوریتم در توضیحات مربوط به فایل‌های System و Switch آمده است.
- **Group**: اگر کاربر این کلمه را به عنوان کلمه اول دستورش وارد نماید، به معنی این است که قرار است یک گروه multicast ایجاد کند. فرمت کامل این دستور به شکل زیر است:

Group #Group_name

ابتدا یکتایی نام گروه وارد شده را بررسی می‌نماییم؛ در صورتی که مشکلی وارد نبود، گروه را ایجاد کرده و آیدی آن را برای تمامی روترها (Switchها) ارسال می‌کنیم.

- **Join_group**: اگر کاربر این کلمه را به عنوان کلمه اول دستورش وارد نماید، به معنی این است که قرار است یک System به یک گروه multicast بپیوندد. فرمت کامل این دستور به شکل زیر است:

Join_group #System_name #Group_name

در ابتدا چک می‌کنیم که فرمت ورودی به شکل بالا باشد، اگر نبود با پیغام Bad request آن را به اطلاع کاربر می‌رسانیم؛ سپس بررسی می‌کنیم که System فعالی با نام سیستم Src وارد شده در شبکه موجود باشد و سپس وجود گروهی با نام وارد شده را بررسی می‌کنیم؛ اگر هریک موجود نباشند، با پیغام Wrong source or destination! موضوع را به کاربر اعلام می‌کنیم. اگر هیچکدام از این اتفاق‌ها رخ نداد، باید عملیات پیوستن را کامل کنیم؛ بدین منظور، ابتدا آیدی گروه را برای خود System ارسال می‌کنیم تا از عضویت در آن گروه آگاه شود و سپس آیدی عضو جدید را برای همه Switchها ارسال می‌کنیم تا آن را به عنوان عضوی از گروه مذکور بشناسند.

- **create_all_pipes()**: قبل از شروع کار شبکه، با استفاده از این متد، به تعداد NUM_OF_PIPES پایپ از نوع unnamed می‌سازیم.
- **prepare_connect_message()**: از این متد برای آماده کردن مقدمات دستور Connect استفاده می‌شود؛ در این متد، ابتدا از آرایه حاوی تمامی pipeهای موجود شبکه، دو pipe اخذ می‌نماییم. آدرس pipe اول +

آدرس خواندن از پایپ دوم را برای یک طرف از Connection آماده می‌کنیم و آدرس pipe دوم + آدرس خواندن از پایپ اول را برای ظرف دیگر Connection آماده می‌نماییم.

- **create_pipe()**: از این متد در هنگام ساخت Switch یا System و برای ساخت pipe ارتباطی برنامه اصلی با آن‌ها استفاده می‌شود. در این متد نیز از آرایه حاوی تمامی pipeهای موجود شبکه، یک pipe اخذ می‌کنیم؛ اگر در حال ساخت Switch بودیم، آن را به مجموعه Switch_pipeها و اگر درحال ساخت System بودیم، آن را به مجموعه System_pipeها می‌افزاییم.
- **fill_pipe()**: در هنگام ساخت Switch لازم است تا تعداد portهای آن به او ارسال نماییم؛ بدین منظور از این متد استفاده کرده و برروی pipe که برای Switch ساختیم، تعداد portها را که از دستور ورودی خواندیم می‌نویسیم.
- **fork_component()**: پس از آماده‌سازی تمامی مقدمات، حال باید برنامه مربوط به Componnet جدید را اجرا کنیم؛ بدین منظور fork انجام می‌دهیم؛ اگر که در فرایند فرزند بودیم، برای اجرای برنامه Component جدید متد run_component را فراخوانی می‌کنیم و اگر در فرایند پدر بودیم، pid فرایند ایجاد شده را به وکتور switches و یا systems (با توجه به نوع دستور) اضافه می‌نماییم.
- **run_component()**: در این متد، اگر در حال ساخت switch بودیم، برنامه Switch_Main را با argv که حاوی آدرس پایپ ساخته شده مربوط به آن switch است + name آن switch، exec می‌نماییم و اگر در حال ساخت system بودیم، برنامه System_Main را با argv که حاوی آدرس پایپ ساخته شده مربوط به آن system است + name آن system، exec می‌نماییم.
- **create_namedPipe()**: در این متد برای هر Component ساخته شده، با استفاده از pid آن یک namedPipe می‌سازیم.
- **get_message()**: در این متد، از روی namedPipe مربوط به component که می‌خواهیم از او پیام دریافت کنیم، پیام ارسالی او را می‌خوانیم.
- **send_message()**: در این متد، بر روی pipe آن component مقصد، پیام مورد نظر را می‌نویسیم.
- **exit_all_components()**: در این متد، به تمامی Componentهای درون شبکه، پیام EXIT را ارسال می‌نماییم.
- **~LoadBalancer() (Destructor)**: برای تمامی Componenهایی که exec کردیم، wait می‌کنیم تا kill شوند و سپس object این class را از بین می‌بریم.

- **ماژول Switch**: این ماژول برای مدیریت تمامی فعالیت‌های یک Switch تعبیه شده و هر زمان که Switch ساخته شود، یک نسخه از این برنامه exec می‌شود. متدهای این ماژول به شرح زیر هستند:

- **Switch (Constructor)**: همانطور که قبلا گفتیم، آدرس pipe برنامه اصلی برای خواندن پیام‌ها و نیز نام Switch در argv به برنامه داده می‌شود و با استفاده از این دو مقدار Object را می‌سازیم؛ بنابراین

attributeهای pipeFd و name را مقداردهی می‌کنیم؛ سپس برای ارسال پیام به برنامه اصلی، آدرس named_pipe را با استفاده از pid فرایند، مقداردهی می‌کنیم. سپس از روی pipe ارسالی برنامه اصلی تعداد پورت‌ها را می‌خوانیم و در number_of_ports ذخیره می‌کنیم. اگر تمامی این اتفاقات با موفقیت انجام شد، پیام S را به برنامه اصلی ارسال می‌کنیم و با فراخوانی متد wait_for_command() منتظر دستورات بعدی می‌مانیم و در غیر اینصورت F را ارسال می‌کنیم و فرایند را به پایان می‌رسانیم.

- **send_message_LB()**: ابتدا named_pipe را با fifopath آماده شده باز می‌کنیم و پیام مورد نظر را بر روی آن می‌نویسیم.
- **wait_for_command()**: تا زمانی که دستور exit دریافت نکردیم، ابتدا از روی pipe ارسالی برنامه اصلی می‌خوانیم؛ اگر چیزی بر روی آن نوشته شده بود، آن را خوانده و با استفاده از متد command_handler آن را handle می‌کنیم و سپس هر بار تمامی connectionهای خود را (چه switchهای دیگر و چه systemهای دیگر) چک می‌کنیم و اگر پیامی روی آن بود آن را دریافت و بررسی می‌کنیم.
- **command_handler()**: در این متد که در واقع متد اصلی این ماژول می‌باشد، با توجه به پیام ارسالی برنامه اصلی و یا Switch و Systemهایی که به آن متصل هستیم، تصمیم گرفته می‌شود که چه Action انجام گیرد؛ انواع دستورات (word اول آن‌ها)، فرمت قابل قبول آن‌ها و فرایندی که برای هر یک در این ماژول انجام می‌شود، به شرح زیر است:
  - **exit**: اگر این پیام ارسال شده باشد، اجرای این برنامه را به آخر می‌رسانیم.
  - **Connect**: فرمت پیام ارسالی باید به شکل زیر باشد:
    - Connect fds_self[Read] fds_self[Write] fds_other[Read]
  - اگر Switch ما پورت خالی داشته باشد و فرمت پیام به شکل بالا باشد، pipeهای switch را با استفاده از متد create_pipe تنظیم می‌کنیم؛ به این صورت که fds_self[Read] و fds_self[Write] به عنوان پایپ‌های طرف خود (که بر روی آن می‌نوسید) و fds_other[Read] را به عنوان پایپ طرف دیگر اتصال (که از روی آن می‌خواند) تنظیم کرده و بر روی اولین Port خالی آن را قرار می‌دهد و در پایان پیام S را به عنوان موفقیت عملیات برای برنامه اصلی ارسال می‌کند و در غیر اینصورت پیام F را ارسال می‌کند.
  - **Send**: این پیام از طریق برنامه اصلی ارسال نمی‌شود بلکه از طریق سایر Switchها و یا Systemها بر روی پورت‌های آن قرار گرفته است؛ فرمت پیام باید به شکل زیر باشد:
    - Send Sender Src Dest Filename message port_number
  - لازم به ذکر است که port_number پس از اینکه پیام از روی پایپ خوانده شد، به آن اضافه می‌شود؛ پس از دریافت این پیام، با استفاده از متد prepare_message_send پیام را برای ارسال آماده می‌کنیم و پس از آن پیام آماده شده را با استفاده از متد send_multicast_message ارسال می‌کنیم (جزئیات ارسال پیام Multicast در این پیام قرار دارد).

- **Send_sync**: همانطور که گفتیم، برای Sync کردن اتصالات و در واقع تشکیل جدول multicast، لازم است تا مسیر میان هر دو Node را بیابیم؛ بدین منظور یکبار باید از مبدا به مقصد حرکت کنیم و یکبار باید مسیر یافته شده را به مبدا ارسال کنیم. مرحله دوم به وسیله این حالت صورت میگیرد. بدین صورت که ابتدا، مسیر ارسالی را بررسی کرده و جدول خود را آپدیت می‌کند (اینکه قدم بعدی برای رسیدن به مقصد درون پیام چیست را نگه می‌دارد) . سپس آن را با همان روش broadcast برای مبدا درخواست‌کننده مسیر ارسال می‌کند.
- **Recv**: این پیام از طریق برنامه اصلی ارسال نمی‌شود بلکه از طریق سایر Switchها و یا Systemها بر روی پورت‌های آن قرار گرفته است و در واقع برای انجام طرف اول عملیات Sync از سمت Src درخواست‌دهنده مسیر به مقصد ارسال می‌شود؛ فرمت پیام باید به شکل زیر باشد:

Recv Sender Src Dest Filename message port_number

لازم به ذکر است که port_number پس از اینکه پیام از روی پایپ خوانده شد، به آن اضافه می‌شود؛ پس از دریافت این پیام، ابتدا lookup_table را با استفاده از متد update_lookup به روزرسانی می‌کنیم. سپس با استفاده از متد prepare_message_send پیام را برای ارسال آماده می‌کنیم و پس از آن پیام آماده شده را با استفاده از متد send_message ارسال می‌کنیم (اینکه باید broadcast انجام شود یا خیر در این متد مشخص می‌شود).

- **Delete**: این دستور وقتی صادر می‌شود که در اتصالات دور ایجاد شده و می‌خواهیم یکی از اتصالات switch را که آدرس pipe آن داده شده حذف کنیم؛ این کار را با استفاده از متد Delete_pipe انجام می‌دهیم.
- **Group**: با دریافت این پیامف گروه مذکور را به لیت گروه های درون شبکه می‌افزاید.
- **Join_group**: با دریافت این پیام client جدیدی را که به گروه مذکور پیوسته، به لیست کاربران آن گروه می‌افزاید.

- **Delete_pipe()**: در این متد pipe که مشخصات آن داده شده از system_pipes حذف می‌شود و اگر در lookup_table این pipe را داشتیم، از آنجا هم حذفش می‌کنیم.
- **create_pipe()**: در این متد، با استفاده از ورودی، پایپ لازم برای اتصال به یک Componnet دیگر را آماده کرده و آن را در system_pipes ذخیره می‌کنیم و از پورت‌های باقی‌مانده یک واحد کم می‌کنیم (چون با اینکار یک پورت را به یک اتصال جدید اختصاص دادیم).
- **check_pipes()**: هر بار تمامی portها را بررسی می‌کنیم؛ اگر پیامی بر روی آن بود، شماره port که از روی آن پیام را خواندیم به انتهای پیام اضافه می‌کنیم و آن را با استفاده از command_handler، پارس و handle می‌کنیم.

- **update_lookup()**: چک می‌کنیم اگر آیدی مورد نظر قبلا در lookup_table بود، که هیچ اگر نه id و port متناظر با آن را به lookup_table اضافه می‌کنیم.
- **send_message()**: اگر id مقصد در lookup_table بود، که مستقیما پیام را برای او ارسال می‌کنیم اگر نه باید پیام را برای همه portها (به جز فرستنده به منظور جلوگیری از تشکیل حلقه) ارسال کنیم.
- **send_multicast_message()**: اگر نام گروه مقصد معتبر بود، به لیست clientهای آن گروه مراجعه می‌کند و با توجه به جدول multicast که به وسیله sync ایجاد شده بود، در می‌یابد که برای ارسال پیام به هر یک از این clientها لازم است تا پیام را در این مرحله به کدام روتر (یا مستقیما client) ارسال نماید؛ سپس بپیام را به مقاصد یافته شده ارسال می‌کند.
- **prepare_message_send()**: در این متد، پیام را برای دستور send آماده می‌کنیم؛ به جای نام فرستند قبلی، نام switch را به عنوان فرستنده جدید قرار می‌دهیم و همچنین شماره پورتی که به انتهای پیام اضافه کرده بودیم را حذف می‌کنیم و به سایر ساختار پیام دست نمی‌زنیم.
- **prepare_message_recv()**: در این متد، پیام را برای دستور Recv آماده می‌کنیم؛ به جای نام فرستند قبلی، نام switch را به عنوان فرستنده جدید قرار می‌دهیم و همچنین شماره پورتی که به انتهای پیام اضافه کرده بودیم را حذف می‌کنیم و به سایر ساختار پیام دست نمی‌زنیم.
- **~Switch() (Destructor)**: فرایند تشکیل شده را با استفاده از فراخوانی سیستمی exit از بین می‌بریم.

- **ماژول System**: این ماژول برای مدیریت تمامی فعالیت‌های یک System تعبیه شده و هر زمان که System ساخته شود، یک نسخه از این برنامه exec می‌شود. متدهای این ماژول به شرح زیر هستند:

- **System (Constructor)**: همانطور که قبلا گفتیم، آدرس pipe برنامه اصلی برای خواندن پیام‌ها و نیز نام System در argv به برنامه داده می‌شود و با استفاده از این دو مقدار Object را می‌سازیم؛ بنابراین attributeهای pipeFd و name را مقداردهی می‌کنیم؛ سپس برای ارسال پیام به برنامه اصلی، آدرس named_pipe را با استفاده از pid فرایند، مقداردهی می‌کنیم. اگر تمامی این اتفاقات با موفقیت انجام شد، پیام S را به برنامه اصلی ارسال می‌کنیم و با فراخوانی متد wait_for_command() منتظر دستورات بعدی می‌مانیم و در غیر اینصورت F را ارسال می‌کنیم و فرایند را به پایان می‌رسانیم.
- **send_message_LB()**: ابتدا named_pipe را با fifopath آماده شده باز می‌کنیم و پیام مورد نظر را بر روی آن می‌نویسیم.
- **wait_for_command()**: تا زمانی که دستور exit دریافت نکردیم، ابتدا از روی pipe ارسالی برنامه اصلی می‌خوانیم؛ اگر چیزی بر روی آن نوشته شده بود، آن را خوانده و با استفاده از متد command_handler آن را handle می‌کنیم و سپس هر بار connection خود به switch را (در صورت برقراری connection) چک می‌کنیم و اگر پیامی روی آن بود آن را دریافت و بررسی می‌کنیم.

- **command_handler()**: در این متد که در واقع متد اصلی این ماژول می‌باشد، با توجه به پیام ارسالی برنامه اصلی، تصمیم گرفته می‌شود که چه Action انجام گیرد؛ انواع دستورات (word اول آن‌ها)، فرمت قابل قبول آن‌ها و فرایندی که برای هر یک در این ماژول انجام می‌شود، به شرح زیر است:
  - **exit**: اگر این پیام ارسال شده باشد، اجرای این برنامه را به آخر می‌رسانیم.
  - **Connect**: فرمت پیام ارسالی باید به شکل زیر باشد:

Connect fds_self[Read] fds_self[Write] fds_other[Read]

pipeهای System را با استفاده از متد create_pipe تنظیم می‌کنیم؛ به این صورت که fds_self[Read] و fds_self[Write] به عنوان پایپ‌های طرف خود (که بر روی آن می‌نوسید) و fds_other[Read] را به عنوان پایپ طرف دیگر اتصال (که از روی آن می‌خواند) تنظیم کرده و در پایان پیام S را به عنوان موفقیت عملیات برای برنامه اصلی ارسال می‌کند و در غیر اینصورت پیام F را ارسال می‌کند.

  - **Send**: این پیام از طریق برنامه اصلی ارسال می‌شود و به این منظور است که دستور Send دریافت شده که شما فرستنده آن هستید (و باید پیام با آن محتویات را ارسال کنید)؛ فرمت پیام باید به شکل زیر باشد:

Send Src Dest Filename

پس از دریافت این پیام، ابتدا محتوای فایل با نام Filename را با استفاده از متد read_file را به صورت یک string دریافت می‌کنیم. سپس تا زمانی که ارسال محتوای فایل به پایان نرسیده است (برای Handle کردن این موضوع که اگر سایز فایل از یک حدی بیشتر بود باید قطعه قطعه شود) محتوای پیام ارسالی را با استفاده از متد prepare_message_send آماده می‌کنیم و پس از آن پیام آماده شده را با استفاده از متد send_message به switch که به آن متصل هستیم، ارسال می‌کنیم.

  - **Recv**: این پیام از طریق برنامه اصلی ارسال می‌شود و به این منظور است که دستور Recv دریافت شده که شما فرستنده آن هستید (و باید پیام Recv خود را ارسال کنید)؛ فرمت پیام باید به شکل زیر باشد:

Recv Src Dest Filename

پس از دریافت این پیام، پیام را با استفاده از متد prepare_message_recv آماده می‌کنیم و پس از آن پیام آماده شده را با استفاده از متد send_message به switch که به آن متصل هستیم، ارسال می‌کنیم.

  - **Sync**: با دریافت این پیام، پیام درخواست یافتن مسیر به **همه** clientهای موجود در شبکه را به روتری که به آن متصل است، ارسال می‌کند.

- **Join_group**: با دریافت این پیام، گروه مذکور را به لیست گروه‌هایی که client در آن عضو است، اضافه می‌کند.

- **create_pipe()**: در این متد، با استفاده از ورودی، پایپ لازم برای اتصال به یک Switch را آماده کرده و آن را در fds ذخیره می‌کنیم و وضعیت اتصال System را با استفاده از متغیر connected در حالت برقرار تنظیم می‌کنیم.
- **send_message()**: پیام مورد نظر را با استفاده از pipe که در متد قبلی تنظیم کردیم، برای switch ارسال می‌کنیم.
- **prepare_message_send()**: در این متد، پیام را برای دستور send آماده می‌کنیم؛ پیام باید به فرمت زیر باشد:

Send my_name src dest filename file_content

در هنگام تنظیم file_content بررسی می‌کنیم که اگر سایز پیام از max_size قابل انتشار کمتر شد، که همه محتوا را ارسال می‌کنیم اگر نه به اندازه جای خالی پیام ارسال کرده و باقی را برای دفعات بعد می‌گذاریم.

- **prepare_message_recv()**: در این متد، پیام را برای دستور Recv آماده می‌کنیم؛ پیام باید به فرمت زیر باشد:

Recv my_name src dest filename

- **check_pipes()**: در این متد، اگر اتصال برقرار بود، هر بار pipe مخصوص خواندن از switch را چک می‌کنیم، اگر پیامی بر روی آن قرار داده شده بود، بررسی می‌کنیم که از id مربوط به dest آن پیام با نام ما یکسان بود (یعنی پیام برای ما ارسال شده بود) آن را گرفته و با استفاده از متد handle_message آن را handle می‌کنیم.
- **handle_message()**: این متد برای رسیدگی به پیام‌هایی که از طریق switch برای system ارسال می‌شود تعبیه شده است؛ انواع دستورات (word اول آن‌ها)، فرمت قابل قبول آن‌ها و فرایندی که برای هر یک در این ماژول انجام می‌شود، به شرح زیر است:
  - **Send**: اگر این پیام ارسال شده باشد، یعنی یک System دیگری یک فایل برای ما ارسال کرده است. در این حالت، در لیست فایل‌های خود جست‌وجو می‌کنیم، اگر این فایل موجود نبود، این فایل و محتویاتش را به لیست خود اضافه می‌کنیم و اگر در لیست فایل‌ها موجود بود، یعنی این فایل قبلا ارسال شده و حالا ادامه محتویات آن ارسال شده است که در این صورت محتویات جدید را به انتهای فایل اضافه می‌کنیم.
  - **Send_sync**: اگر این پیام ارسال شده باشد، یعنی یک پاسخ درخواست یافتن مسیری که قبلا داده بودیمف آمده است و مسیر میان client با client که src پیام دریافتی است، ارسال شده است. ابتدا پیام را دیکود کرده و مسیر را می‌یابیم. در جدول، قدم بعدی برای رسیدن به آن مقصد را set می‌کنیم.

- **Recv**: اگر این پیام ارسال شده باشد، یعنی یک System دیگری مسیر خود به مقصد ما را درخواست کرده است. مسیری که تا الان یافته شده (و در متن پیام ارسال موجود است) را به اضافه ادرس خود کرده و به مقصدِ مبدا پیام (کسی که درخواست مسیر را داشته) ارسال می‌کنیم.

- **~System() (Destructor):** فرایند تشکیل شده را با استفاده از فراخوانی سیستمی exit از بین می‌بریم.

➢ راستی‌آزمایی

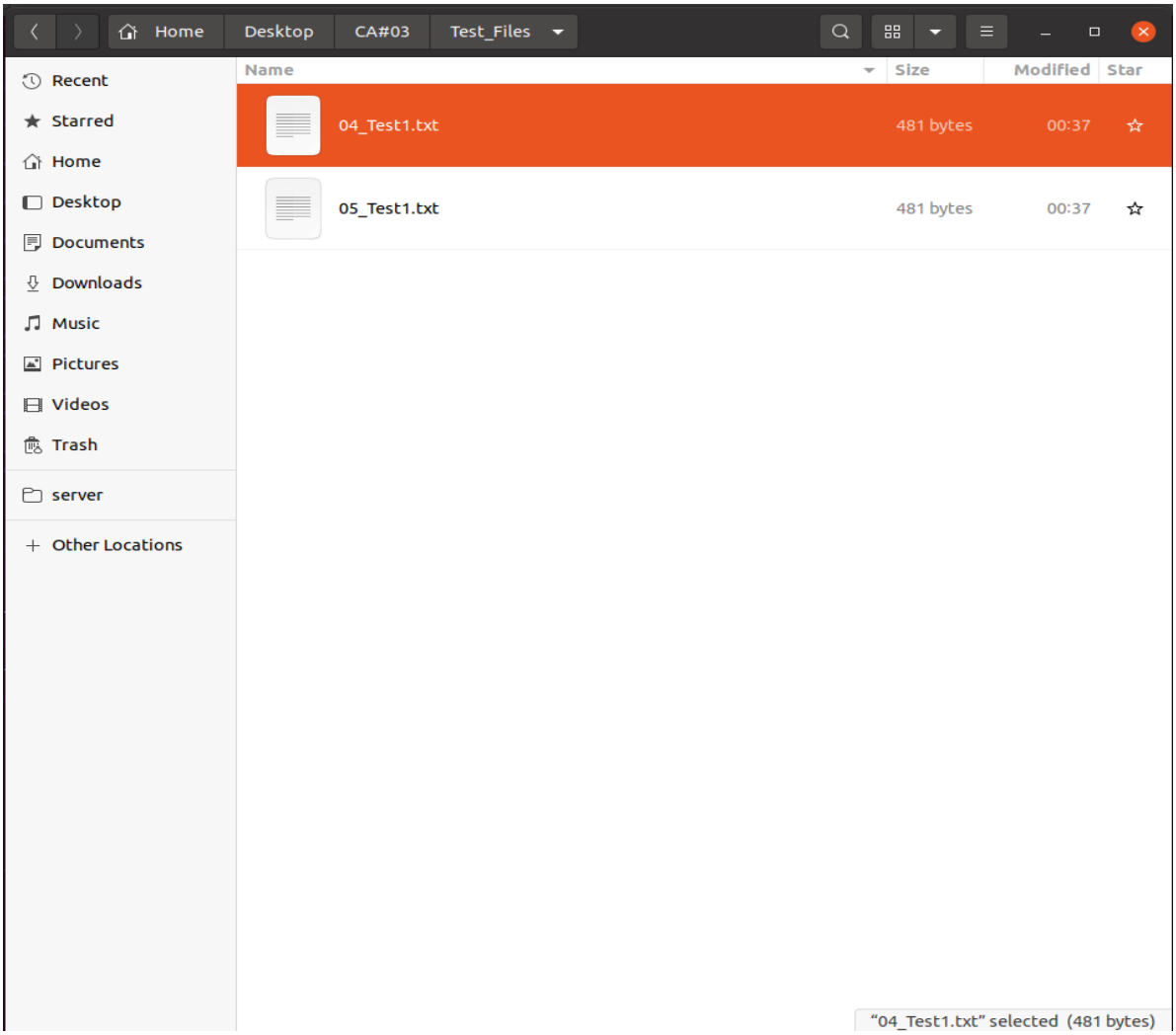
- ساخت دو switch و سه system و تشکیل یک گروه با دو system و ارسال پیام از system تنها به گروه تشکیل شده!

```
marzieh@marzieh:~/Desktop/CA#03$ ./Network.out
Welcome!
Enter your command:
Switch 5 01
Enter your command:
Switch 5 02
Enter your command:
System 03
Enter your command:
System 04
Enter your command:
System 05
Enter your command:
Connect_S 01 02
Connected!
Enter your command:
Connect 03 01
Connected!
Enter your command:
Connect 04 01
Connected!
Enter your command:
Connect 05 02
Connected!
Enter your command:
Sync
Enter your command:
Group G1
Group G1 created successfully!
Enter your command:
Join_group 04 G1
Enter your command:
Client joins G1 Group successfully!
Join_group 05 G1
Enter your command:
Client joins G1 Group successfully!
Send 03 G1 Test1.txt
Enter your command:
The system 03 sent the <Send> message.
The switch 01 send the <Send> message.
The system 04 from Group G1 recieved the <Send> message.
The switch 02 send the <Send> message.
The system 05 from Group G1 recieved the <Send> message.
exit
marzieh@marzieh:~/Desktop/CA#03$
```

در این تست، ابتدا دو سوئیچ با ۵ پورت و آیدی ۰۱ و ۰۲ می‌سازیم. سپس ۳ عدد system با آیدی‌های ۰۳ تا ۰۵ می‌سازیم. دو سوئیچ را بهم متصل می‌کنیم؛ Systemهای ۰۳ و ۰۴ را به سوئیچ ۰۱ و System ۰۵ را به سوئیچ ۰۲ وصل می‌کنیم.

سپس شبکه را Sync می‌کنیم تا جداول تشکیل شوند.

در گام بعدی گروه G1 را ساخته و Clientهای 04 و 05 را به آن متصل می‌کنیم. سپس از client ۰۳ فایل Test1.txt را برای گروه G1 ارسال می‌کنیم. لاگ‌های چاپ شده، نشان از انتقال درست پیام (همراه با هرس شدن مسیرهای اضافی) دارد. برای آنکه از ارسال درست فایل‌ها مطمعن شویم، فایل ارسالی را با پیشوند نام کلاینت گیرنده، در فولدر Test_files ذخیره می‌کنیم. (این فولدر در ابتدای اجرای برنامه به کمک make clean خالی خواهد شد)



با توجه به اینکه اعضای گروه G1، کلاینت‌های 04 و 05 بودند، ارسال پیام‌ها به درستی انجام شده است

➢ نکته‌های تکمیلی

- برای اطمینان از آنکه فایل‌ها به درستی انتقال می‌یابند، هر فایل ارسالی پس از اینکه در مقصد دریافت شدف در پوشه Test_files ذخیره می‌شود.
- برای اجرای برنامه پس از استفاده از make، با وارد کردن دستور Network.out/. اجرای برنامه آغاز می‌شود. نمونه دستورات در تست‌های بالا آمده است.
- در زمانی که ارسال یا دریافت فایل صورت می‌گیرد، به علت تقدم یا تاخر فرایندها، دستورات جا به جا بر روی ترمینال نوشته می‌شوند.