دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

عنوان:

پیاده‌سازی شل سفارشی شده

اعضای گروه

مهتا فطرت (۹۸۱۰۲۰۹۲)

فاطمه عسگری (۹۸۱۰۵۹۲۱)

مریم‌سادات رضوی‌طاهری (۹۸۱۰۱۶۳۹)

نام درس

سیستم‌های عامل

نیم‌سال اول ۱۴۰۱-۱۴۰۲

نام استاد درس

دکتر حسین اسدی


بسم الله الرحمن الرحیم

۱ مقدمه

در این پروژه یک پوسته[۱] سفارشی پیاده‌سازی شده است که مشابه پوسته Bash بوده و به کاربر واسطی جهت ارتباط با سرویس‌های سیستم‌عامل شامل کار با فایل‌ها و پردازه‌ها ارائه می‌دهد. پیاده‌سازی این شل به زبان C و با استفاده از برخی فراخوانی‌های سیستمی[۲] سیستم‌عامل انجام شده است.

۲ نحوه راه‌اندازی

برای راه‌اندازی اولیه پروژه کافی است فایل اصلی (main.c) به همراه فایل‌های parser.c و ex-ecutor.c با استفاده از کامپایلر دلخواه کامپایل شود و سپس با اجرای فایل خروجی main، پوسته سفارشی قابل استفاده خواهد بود. جهت تسهیل در فرایند راه‌اندازی پروژه می‌توان با دستور make پروژه را بلافاصله اجرا کرد. لازم به ذکر است این Makefile از کامپایلر gcc استفاده می‌کند و لازم است gcc برروی سیستم مورد نظر نصب شده باشد.

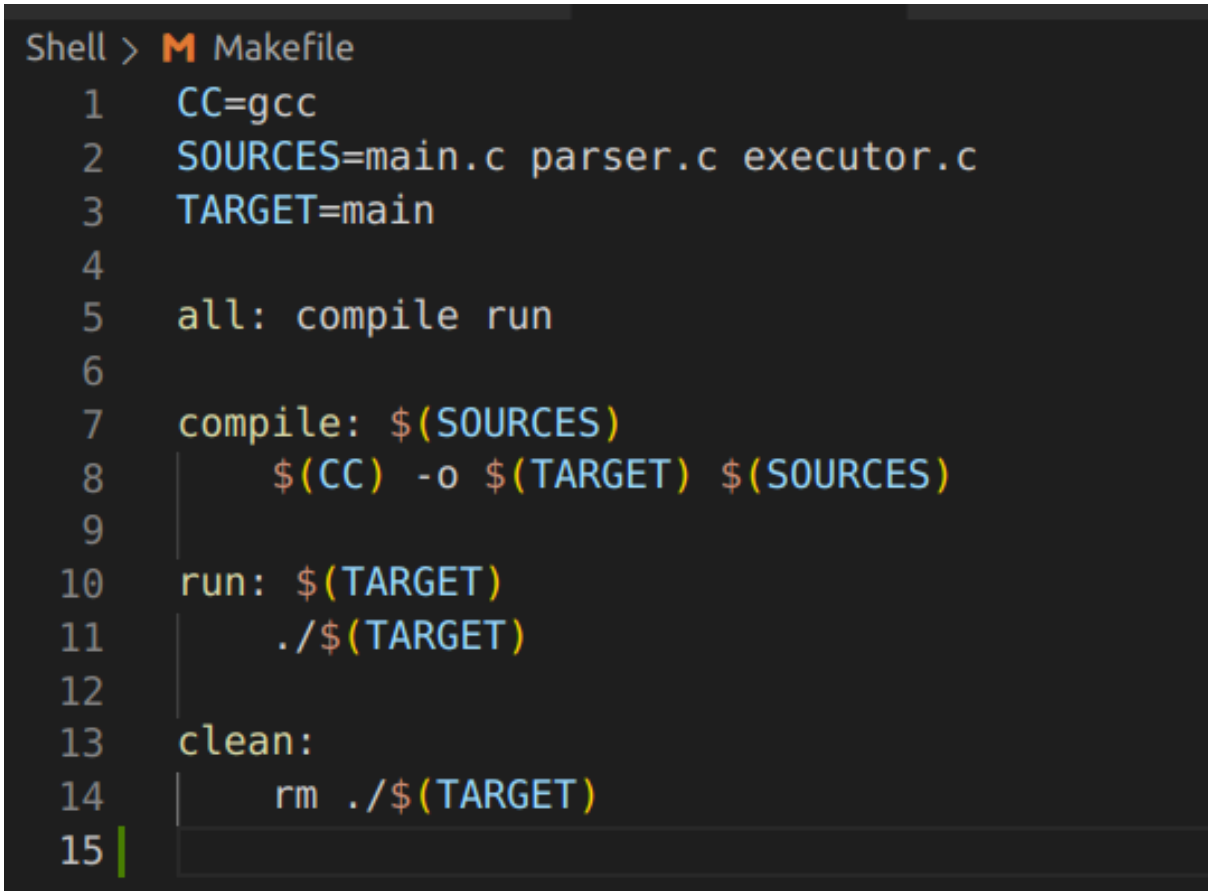
۱-۲ Makefile

به منظور تسهیل عملیات‌های compile و مدیریت فایل‌های اجرایی پروژه، یک Makefile در root پروژه قرار گرفته است که از چند دستور پشتیبانی می‌کند. محتوای این فایل در شکل ۱ قابل مشاهده است.

در ابتدای این فایل، یک‌سری متغیر تعریف شده‌اند. متغیر CC کامپایلر مورد استفاده را نگه‌داری می‌کند و متغیرهای SOURCES و TARGET به ترتیب نام فایل‌هایی که باید کامپایل شوند و نام فایل اجرایی خروجی را نگه می‌دارند. در ادامه سه دستور در این فایل پیاده‌سازی شده‌اند. این موارد عبارت‌اند از:

- **compile**: این دستور، فایل‌های c. یا همان Source ها را به کمک gcc کامپایل می‌کند و یک فایل اجرایی با نام main خروجی می‌دهد. فایل‌های منبع به‌جز main به کمک فایل‌های header شان در main لینک شده و مورد استفاده قرار می‌گیرند.

---
[۱] Shell
[۲] Systemcall



شکل ۱: Makefile پروژه

- **run**: این دستور، فایل اجرایی خروجی (main) را اجرا می‌کند.
- **clean**: این دستور، فایل‌های اجرایی خروجی کامپایلر را از root پروژه پاک می‌کند.

در پایان این دستورات در قالب دستور all آورده شده‌اند تا با اجرای دستور make در root پروژه، فایل‌های مورد نیاز compile شده و خروجی نیز اجرا شود.

۳ نحوه کار با پوسته

با وارد کردن دستور help می‌توان توضیح مختصری از تمامی دستورات قابل اجرا در پوسته را مشاهده کرد. Shell پیاده‌سازی‌شده در این پروژه شامل یک‌سری دستورات built-in می‌باشد که در هسته‌ی Shell پیاده‌سازی و اجرا می‌شوند. علاوه بر این دستورات که عناوین آن در ادامه آورده شده‌است، دستورات مورد پشتیبانی پوسته‌ی سیستم‌عامل و برنامه‌های custom کاربر نیز در این Shell قابل اجرا هستند.

۳-۱ دستورات مورد پشتیبانی

- **quit**: اجرای Shell را متوقف می‌کند.
- **help**: خلاصه‌ای از دستورات مورد پشتیبانی Shell را نمایش می‌دهد.
- **cd [dir]**: مسیر اجرایی جاری را به آرگومان dir تغییر می‌دهد.
- **cwd**: directory جاری را نمایش می‌دهد.
- **history**: این دستور تاریخچه‌ای از تمام دستوراتی که تا کنون وارد شده را نمایش می‌دهد.
- **[program] [args ...]**: یک برنامه به همراه آرگومان‌های ورودی آن را دریافت کرده و برنامه را اجرا می‌کند. این دستور از ۳ نوع آدرس‌دهی برای برنامه پشتیبانی می‌کند.
  - مطلق: در این حالت آدرس برنامه به صورت کامل داده می‌شود.
  - نسبی: در این حالت آدرس برنامه نسبت به آدرس جاری داده می‌شود. به عنوان مثال myfile/...، برنامه‌ای در یک پوشه بالاتر را مشخص می‌کند.
  - نسبت به PATH: در این حالت آدرس برنامه مجددا نسبی داده می‌شود اما این بار نسبت به directory های تعریف شده در متغیر محلی .محلیPATH به عنوان مثال زمانی که تنها نام برنامه‌ی ls به عنوان ورودی داده شود، برنامه با جستجو در مسیر usr/bin/، این برنامه را در آدرس usr/bin/ls/ پیدا می‌کند و آن را اجرا می‌کند.

۴ توضیحات پیاده‌سازی

در پیاده‌سازی این پروژه از زبان C و برخی فراخوانی‌های سیستمی سیستم‌عامل استفاده شده است. پروژه به صورت ماژولار پیاده‌سازی شده‌است به نحوی که اجرای اصلی برنامه از تابع main آغاز شده و دستورات وارد شده توسط parser تجزیه و بررسی شده و توسط executor اجرا می‌شوند. هر کدام از ماژول‌های پروژه شامل یک فایل c. و یک فایل header هستند که فانکشنالیتی و برخی جزئیات پیاده‌سازی توابع، متغیرها و ماژول‌ها را در قالب مستندات inline در بر دارند.

۱-۴ ماژول main

فایل main.c، ماژول اصلی پروژه و starting point پروژه می‌باشد. در این ماژول، prompt پوسته که فرمت cwd$ را دارد، به کاربر نمایش داده می‌شود. همچنین در یک loop بی‌نهایت، دائما از کاربر ورودی دریافت و به parser فرستاده می‌شود تا یک دستور معنادار به همراه پارامتر های آن دریافت شود. سپس این دستور به ماژول executor پاس داده می‌شود تا به اجرا در بیاید. نمونه‌ای از prompt پوسته در تصویر ۲ آمده‌است.

```
~/Downloads/Semester7/os/project/Shell$ cd ..
~/Downloads/Semester7/os/project$
```

شکل ۲: نمونه‌ی prompt پوسته

۲-۴ ماژول parser

این ماژول همانگونه که از نامش پیداست، برای تشخیص و تجزیه‌ی دستورات مورد استفاده قرار می‌گیرد. این ماژول شامل دو بخش اصلی است.

۱-۲-۴ regexes

الگوی هر کدام از انواع دستورات در آرایه‌ای از regex ها در این ماژول تعریف شده‌اند. هر دستور ورودی، یک به یک با این الگوها مقایسه می‌شوند و اولین الگویی که با دستور match شد، نوع دستور را تعیین می‌کند. عملیات‌های تعریف و execute کردن regex ها در زبان c به کمک کتاب‌خانه‌ی regex.h انجام شده‌اند. برخی از regex های تعریف شده در این ماژول در تصویر ۳ قابل مشاهده هستند.

```
10  static const char *COMMANDS[] = {
11      "^[[:space:]]*quit[[:space:]]*$",
12      "^[[:space:]]*help[[:space:]]*$",
13      "^[[:space:]]*cd[[:space:]]+[^[:space:]]+[[:space:]]*$",
14      "^[[:space:]]*cwd[[:space:]]*$",
15      "^[[:space:]]*[^[:space:]]+([[:space:]]+[^[:space:]]+)*[[:space:]]*$"};
```

شکل ۳: regex دستورات

۴-۲-۲ arguments

هر دستور از بخش‌هایی تشکیل شده‌است که شامل keyword ها، symbol ها و آرگومان‌ها هستند. ماژول parser، دستور ورودی را به این زیربخش‌ها تقسیم می‌کند و آن‌ها را در قالب آرایه‌ای به نام argv به همراه طول این آرایه (argc)، به ماژول بالاتر ارائه می‌کند.

۴-۳ ماژول executor

این ماژول دستورات تجزیه و شناخته شده را به اجرا در می‌آورد. بخشی از این ماژول به پیاده‌سازی دستورات built-in پوسته همچون cd و cwd اختصاص دارد و بخش دیگری از آن به اجرای دستورات و برنامه‌های خارجی می‌پردازد.

۴-۳-۱ دستورات داخلی

دستور cd: این دستور از تابع chdir به منظور تغییر directoy استفاده می‌کند. نکته‌ی لازم به ذکر دیگر در رابطه با پیاده‌سازی این دستور built-in آن است که از کاراکتر ~ پشتیبانی می‌کند. به این معنا که کاراکتر ~ در ابتدای مسیر ورودی را به عنوان یک shortcut برای مسیر HOME می‌شناسد. مسیر HOME سیستم به صورت یک متغیر محلی موجود است و با فراخوانی تابع getenv با ورودی "HOME" به‌دست می‌آید.

دستور cwd: این دستور با استفاده از فراخوانی getcwd پیاده‌سازی شده‌است. یک نکته در رابطه با working directory در Shell آن است که در ابتدای اجرای پوسته و در ماژول main، دایرکتوری جاری با فراخوانی chdir با ورودی "HOME" به این مسیر تغییر می‌کند. به عبارتی در ابتدای اجرای پوسته، در دایرکتوری HOME قرار خواهیم داشت.

۴-۳-۲ دستورات خارجی

همانطور که پیش‌تر ذکر شد، پوسته‌ی پیاده‌سازی‌شده از اجرای برنامه‌ها و دستورات خارجی با دریافت آدرس و آرگومان‌های ورودی آن‌ها پشتیبانی می‌کند. نحوه‌ی resolve کردن آدرس‌های ورودی کاربر از هریک از انواع آدرس‌دهی‌های نسبی، مطلق و محلی در بخش بعد آمده‌است و در این بخش فرض می‌کنیم که آدرس مطلق اجرای هر برنامه را در اختیار داریم.

اجرای یک تک‌ـبرنامه: به منظور اجرای یک برنامه از داخل برنامه‌ی خود پوسته، از ترکیب دستورات fork و دستورات خانواده‌ی exec در c استفاده شده‌است. علت استفاده از fork آن است که دستور exec پردازه‌ی جاری را کاملا با پردازه‌ی جدید جایگزین می‌کند و اجرای برنامه‌ی پوسته پس از فراخوانی آن از سر گرفته نمی‌شود. تابع fork در پردازه‌ی فرزند مقدار صفر و در پردازه‌ی پدر، pid فرزند را return می‌کند. با استفاده از همین تمایز، در پردازه‌ی فرزند تابع exec و در پردازه‌ی پدر تابع waitpid با ورودی pid فرزند صدا زده می‌شود. به این ترتیب فرزند، برنامه‌ی خارجی را اجرا کرده و والد برای پایان آن منتظر می‌ماند.

اجرای موازی برنامه‌ها: اگر دستورات توسط کاربر به صورت پشت سر هم و در حالیکه بین‌شان نویسه‌ی ؛ وجود دارد وارد شود، پوسته قادر است تمامی دستورات را به طور همزمان و به شکل موازی اجرا کند.

روش کار پوسته به این صورت است که ورودی داده شده برای اجرای موازی را تشخیص داده و سپس آرگومان‌های دستورات را از یکدیگر جدا می‌کند. هر دستور برای اجرا به تابع exec_parallel داده می‌شود که در آن پردازه‌ی فرزندی به وسیله‌ی fork ایجاد شده و دستور مورد نظر را اجرا می‌کند. توجه شود که به دلیل اجرای موازی پردازه‌ها ممکن است خروجی‌ها با هم تداخل کرده و خروجی دستورات به ترتیب اجرای آن‌ها نباشد.

به طور دقیق‌تر، ابتدا به‌جای نویسه‌های ؛، NULL قرار داده می‌شود. این مقادیر NULL درواقع جداکننده‌ی argv های دستورات هستند. مسیر اجرایی مطلق/نسبی هر دستور به کمک تابع set_executable_path به‌دست می‌آید و سپس، به ازای هر دستور، یک fork انجام می‌شود که در آن، والد برای فرزند منتظر نمی‌ماند. پردازه‌ی فرزند، دستور مربوط به خود را با پاس دادن argv خود به execv اجرا می‌کند. تابع execv محدوده‌ی هایargument آن فرزند را از روی کاراکتر NULL تشخیص می‌دهد. در نهایت، پس از start کردن عملیات‌های پردازه‌های فرزند، پردازه‌ی پدر، در یک حلقه، منتظر terminate شدن تمام پردازه‌های فرزند خود می‌شود و سپس برای دریافت دستور بعدی از وروردی آماده می‌شود. حلق

اجرای pipe شده‌ی برنامه‌ها:

هدایت ورودی و خروجی: پس از تشخیص این که دستوری باید ورودی فایل را به عنوان آرگومان به دستور فراخوانی شده بدهد، ابتدا پردازه فرزندی فورک شده و در صورت ایجاد بدون خطای

این پردازه در آن با فراخوانی تابع input redirect محتوای فایل شامل آرگومان (های) ورودی خوانده شده و در صورت عدم بروز خطا با فراخوانی فراخوان سیستمی dup2 ورودی مورد نظر را به input standard منتقل می‌کند. ورودی‌های dup2 را به ترتیب فایل ورودی و ورودی استاندارد می‌دهیم. بعد از این انتقال می‌توان دستور مورد نظر را با آرگومان‌هایی که الان گویا در ورودی استاندارد منتقل شده‌اند به حالت عادی بقیه اجراها با execv اجرا کرد. برای output redirect هم روال کار دقیقا به همین صورت است با این تفاوت که ابتدا اجرای دستور با execv انجام شده و خروجی به طور کامل تولید می‌شود ولی خروجی استاندارد با همان تابع dup2 به فایل مورد نظر که باز شده داده می‌شود و در آن نوشته می‌شود.

در این قسمت‌ها نیز مشابه سایر دستورات در هر جایی که امکان بروز خطا از جمله خطای باز نشدن فایل، خطای سیستم‌کال‌های فراخوانی شده و... وجود داشته باشد این خطا با stderr و پیغام مناسب به کاربر نمایش داده شده و سیستم اجازه نمی‌دهد خطا باعث توقف Shell بشود.

اجرای برنامه‌ها در پس‌زمینه: برای اجرای برنامه‌ای که پس از آن علامت & آمده و می‌خواهیم در پس‌زمینه سایر دستورات بعدی شل اجرا شود، تنها کافی است یک پردازه فرزند ایجاد کرده و در آن با همان تابع execv برنامه مورد نظر را اجرا کنیم با این تفاوت که اینجا دیگر نیاز نیست پردازه پدر منتظر پردازه فرزند بماند. فرزند در پس زمینه کارش را به اتمام می‌رساند و در این حین پدر هم مشغول ادامه اجرای شل و گرفتن دستورات مختلف است.

اجرای دستورات دلخواه: در داخل یک فایل به نام my_commands.txt که در کنار فایل‌های پروژه قرار دارد، می‌توان دستورات دلخواهی را ذخیره کرده و هنگام وارد کردن این دستورات، پوسته آن‌ها را به دستورات اصلی ترجمه کرده و اجرا می‌کند. ساختار این دستورات باید به شکل زیر باشد:

[desired command]:: main command

به طور مثال می‌توان دستوری را به شکل gst:: git status در این فایل ذخیره کرد. در این صورت با وارد شدن دستور gst توسط کاربر، دستور git status اجرا خواهد شد و کاربر خروجی مورد انتظار برای اجرای دستور git status را خواهد دید.

۴-۳-۳ نحوه‌ی resolve کردن مسیرهای اجرایی

لازم است که مسیرهای اجرایی کاربر، validate شده و به یک مسیر مطلق یا نسبی قابل اجرا تبدیل شوند. این اقدام در تابع set_executable_path در ماژول executor در دو مرحله انجام می‌شود.

مسیرهای مطلق و نسبی: ابتدا وجود کاراکتر / در مسیر ورودی بررسی می‌شود. در صورت وجود، این مسیر به عنوان یک مسیر نسبی یا مطلق شناخته می‌شود. چراکه کاراکتر / در نام فایل‌ها غیر مجاز است و نشان می‌دهد ورودی از نوع نام ساده‌ی برنامه نیست. همچنین توجه داریم که در Bash نیز آدرس‌دهی یک برنامه در دایرکتوری جاری، باید به صورت myfile/. انجام شود و نه myfile. در غیر این صورت پیغام

"command not found"

نمایش داده می‌شود. در این حالت سپس به ترتیب وجود برنامه و دسترسی اجرای آن بررسی می‌شود. این موارد به کمک تابع access به ترتیب با فلگ‌های F_OK و X_OK چک می‌شوند. در صورتی که فایل وجود نداشته باشد، پیغام

"Exection failed: No such file or directory"

و در صورتی که دسترسی اجرای آن وجود نداشته باشد، پیغام

"Exection failed: Permission denied"

نمایش داده می‌شود. در غیر این صورت، مسیر ورودی بدون تغییر باقی می‌ماند و برای execution ارسال می‌شود. چرا که تابع execv از هر دوی مسیرهای نسبی و مطلق پشتیبانی می‌کند.

نام ساده‌ی برنامه: نبود کاراکتر / در مسیر ورودی برنامه به معنای وارد کردن نام ساده‌ی آن است. در این حالت باید نام برنامه در مسیرهای شناخته شده در متغیر محلی PATH جستجو شوند. به این منظور، نام فایل به تک‌تک این مسیرها append می‌شود و وجود مسیر generate شده بررسی می‌شود. اولین مسیر valid به عنوان آدرس کامل آن برنامه شناخته شده و به عنوان مسیر اجرایی execv تنظیم می‌شود.

دسترسی به مسیرهای موجود در متغیر محلی PATH: برای استفاده از این ماژول، لازم است که یک تابع initializer به نام initialize_executor در ماژول فراخواننده صدا زده شود. این تابع پیش از هر چیز، مسیرهای تعریف شده در متغیر محلی PATH را load می‌کند و در اختیار executor قرار می‌دهد. به این منظور، ابتدا با استفاده از تابع getenv با ورودی "PATH"، مقدار این متغیر محلی استخراج می‌شود. این متغیر حاوی رشته‌ای است که در آن مسیرهای شناخته شده با کاراکتر : از هم جدا شده‌اند. سپس این مسیرها به کمک تابع strtok از هم جدا شده و در آرایه‌ای قرار می‌گیرند. به این ترتیب، فراخوانی‌های بعدی برای اجرای بدون آدرس کامل برنامه‌ها می‌توانند برنامه را در این مسیرها جستجو کنند.

۵ نمونه‌های اجرا

```
~$ cd Downloads/Semester7
~/Downloads/Semester7$
```

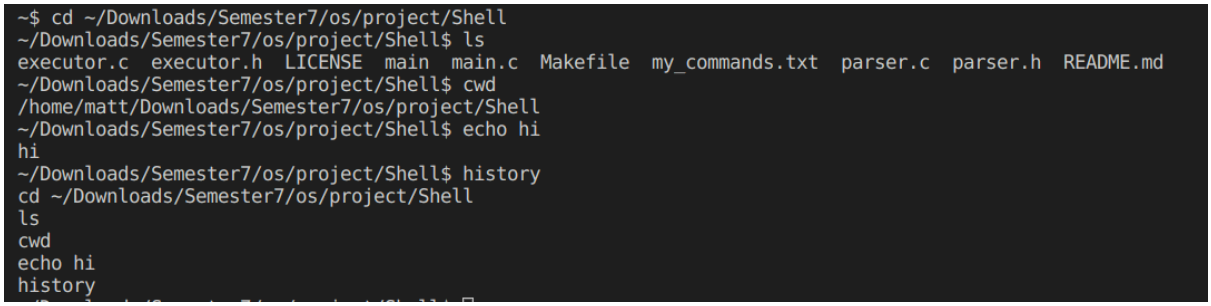
شکل ۴: نمونه‌ی اجرای دستور cd

```
~/Downloads/Semester7$ cwd
/home/matt/Downloads/Semester7
~/Downloads/Semester7$
```

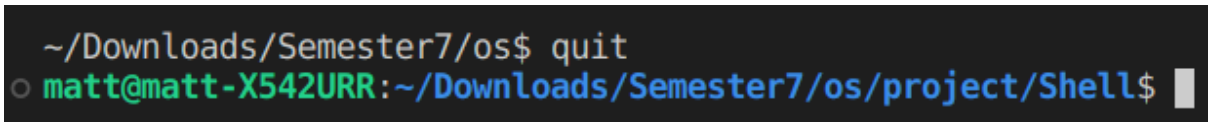
شکل ۵: نمونه‌ی اجرای دستور cwd

```
These shell commands are defined internally.  Type `help' to see this list.
quit                                          Exit the shell
cd [dir]                                      Change the shell working directory.
cwd                                           Print the name of the current working directory.
[program] [args ...]                          Execute program with absolute/relative path or found in PATH dirs with the given args.
[program] [args ...] &                        Execute program in background.
[program] [args ...] | [program] [args ...]   Executes second program with the first program output.
[program] [args ...] > [file]                 Executes the program and writes its output to file.
[program] [args ...] < [file]                 Executes the program with the file as its input stream.
help                                          Display information about builtin commands.
history                                       Display history of commands.
```

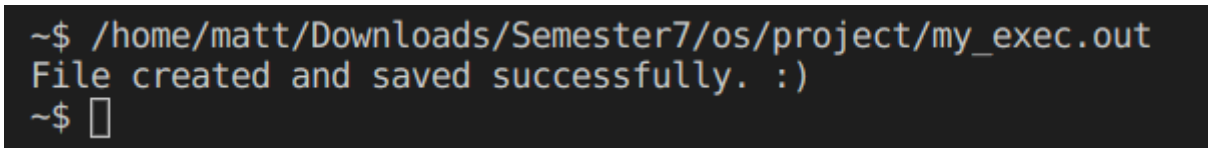
شکل ۶: نمونه‌ی اجرای دستور help



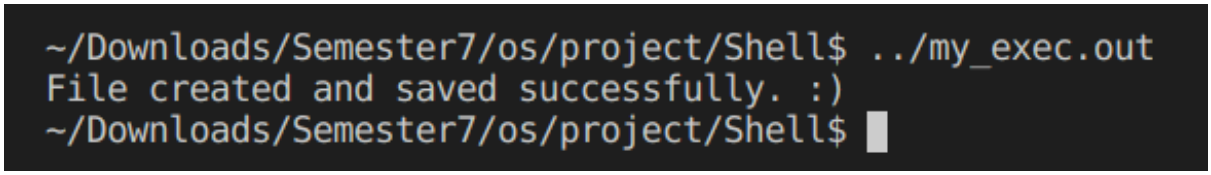
شکل ۷: نمونه‌ی اجرای دستور history



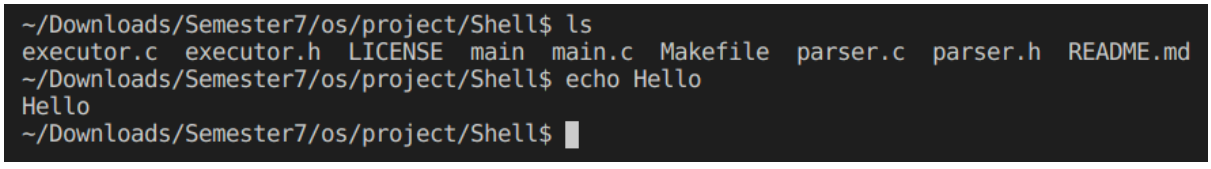
شکل ۸: نمونه‌ی اجرای دستور quit



شکل ۹: نمونه‌ی اجرای یک برنامه با آدرس کامل



شکل ۱۰: نمونه‌ی اجرای یک برنامه با آدرس نسبی



شکل ۱۱: نمونه‌ی اجرای یک دستور با نام ساده‌ی آن

```
gcc -o main main.c parser.c executor.c
./main
~$ cd Desktop
~/Desktop$ lsl
total 1896
drwx------@ 16 tapsi  staff     512 Feb  3 17:45 .
drwxr-xr-x+ 49 tapsi  staff    1568 Feb  3 17:43 ..
-rw-r--r--@  1 tapsi  staff   12292 Feb  3 16:55 .DS_Store
drwxr-xr-x   2 tapsi  staff      64 Jan 14 22:04 .ipynb_checkpoints
-rw-r--r--   1 tapsi  staff       0 Feb  2  2022 .localized
-rwx------@  1 tapsi  staff  861824 Feb  2 22:38 HW8_98105921.pages
drwxr-xr-x@  8 tapsi  staff     256 Apr 16  2022 Learning
drwxr-xr-x  14 tapsi  staff     448 Feb  3 20:09 Shell
-rwxr-xr-x   1 tapsi  staff   34096 Feb  3 17:45 a.o
-rw-r--r--@  1 tapsi  staff      30 Feb  3 10:56 my_commands.txt
drwxr-xr-x   7 tapsi  staff     224 Feb  2 11:50 personal
drwxr-xr-x@  4 tapsi  staff     128 Sep 13  2021 pictures
-rw-r--r--   1 tapsi  staff    5346 Feb  3 17:45 test.c
drwxr-xr-x@  9 tapsi  staff     288 Jan 27 12:06 uni
drwxr-xr-x  41 tapsi  staff    1312 Jan 31 13:52 webapp
drwxr-xr-x   7 tapsi  staff     224 Apr 21  2022 work-related docs
~/Desktop$
```

شکل ۱۲: نمونه‌ی اجرای دستور دلخواه lsl (نماینده‌ای برای دستور ls -al)

```
~/Desktop/Shell$ ls; ls -l; echo 444
444
LICENSE         README.md       executor.h      main.c          parser.c
Makefile        executor.c      main            my_commands.txt parser.h
total 200
-rw-r--r--  1 tapsi  staff   1068 Feb  2 11:23 LICENSE
-rw-r--r--  1 tapsi  staff    193 Feb  2 17:01 Makefile
-rw-r--r--  1 tapsi  staff     51 Feb  2 11:23 README.md
-rw-r--r--  1 tapsi  staff   8833 Feb  4 11:36 executor.c
-rw-r--r--  1 tapsi  staff    589 Feb  3 20:21 executor.h
-rwxr-xr-x  1 tapsi  staff  53665 Feb  4 13:01 main
-rw-r--r--  1 tapsi  staff   2916 Feb  3 20:21 main.c
-rw-r--r--  1 tapsi  staff     29 Feb  3 18:05 my_commands.txt
-rw-r--r--  1 tapsi  staff   2658 Feb  4 11:20 parser.c
-rw-r--r--  1 tapsi  staff    647 Feb  2 17:01 parser.h
~/Desktop/Shell$
```

شکل ۱۳: نمونه‌ی اجرای دستورات موازی