Boletín 2: Redirección, *prompt* y comandos internos

Ampliación de Sistemas Operativos

Dpto. Ingeniería y Tecnología de Computadores (DITEC)

Universidad de Murcia

Curso 2019/2020

Índice

1. Redirecciones en `simplesh`

Redirecciones en simplesh

- Cuando se redirige la salida a un fichero, sus permisos no permiten leer su contenido:

```
simplesh> echo linea1 > listado
simplesh> cat listado
cat: listado: Permiso denegado
simplesh> chmod u+r listado
simplesh> cat listado
linea1
```

- Además, el comportamiento de >> no es el esperado:

```
simplesh> echo linea1 >  listado
simplesh> echo linea2 >> listado
simplesh> cat listado
linea2
```

- Y el de > tampoco:

```
simplesh> echo linea1 >  listado
simplesh> echo linea2 >> listado
simplesh> echo linea3 >  listado
simplesh> cat listado
linea3
linea2
```

Redirecciones en `simplesh`

- (0.25p) EJERCICIO 1: Modifica `simplesh` para que los ficheros se creen con permisos 700 y la semántica de las redirecciones sea la misma que tendrían si se ejecutasen los comandos en `bash`
- Para realizar el ejercicio, utiliza los *flags* y el *mode* de `open()`

2. *Prompt* en `simplesh`

El símbolo del sistema o *prompt* de `simplesh`

- La llamada `readline` muestra el *prompt* y, a continuación, devuelve la línea de comandos introducida por el usuario

```
char* get_cmd()
{
    char* buf= readline("simplesh> ");

    if (buf)
        add_history(buf);

    return buf;
}
```

- Por ejemplo:

```
simplesh> ls > listado # ‘buf‘ apunta a "ls > listado"
```

El símbolo del sistema o *prompt* de `simplesh`

- (0.25p) EJERCICIO 2: Modifica el *prompt* para que muestre el usuario y el directorio de trabajo actual separados por el carácter `@`:

```
1  usuario@directorio>␣...
```

- Para ello, únicamente se pueden usar llamadas POSIX o funciones de la biblioteca estándar de C (C11)
- Llamadas `POSIX` y/o `glibc` a considerar:
  - `getuid()`
  - `getpwuid()`
  - `getcwd()` (Versión `POSIX` sin extensiones `GNU`)[1]
  - `basename()` (Versión `POSIX`)
  - `sprintf()`

[1] Tamaño máximo de la ruta al directorio de trabajo actual: `http://man7.org/linux/man-pages/man0/limits.h.0p.html`.

3. Comandos internos

Comandos internos versus comandos externos

- Los comandos internos son proporcionados por el propio *shell*
  - `help cd` / `man cd` / `which cd` / `type -a cd`
- Los comandos externos son programas independientes del *shell*
  - `help ls` / `man ls` / `which ls` / `type -a ls`
- Algunos comandos son a la vez internos y externos
  - `help pwd` / `man pwd` / `which pwd` / `type -a pwd`
- Comandos como `cd` o `exit` sólo pueden ser internos, ¿por qué?[2]

[2] Véase `man 2 chdir` y `man 2 exit`.

El comando interno `cwd`

- (0.5p) EJERCICIO 3: Implementa el comando interno `cwd`:

```
simplesh> cwd
cwd: /ruta/a/directorio
```

- Para ello, únicamente se pueden usar llamadas al sistema POSIX o funciones de la biblioteca estándar de C (C11)
- Llamadas `POSIX` y/o `glibc` a considerar:
  - `getcwd()` (Versión `POSIX` sin extensiones `GNU`)
- Para implementar `cwd`, escribe la función `run_cwd()` e inserta llamadas a la misma en `simplesh.c` donde sea necesario

El comando interno `exit`

- (1.0p) EJERCICIO 4: Implementa el comando interno `exit`:

```
simplesh> exit
[bash]
```

- Para ello, únicamente se pueden usar llamadas al sistema POSIX o funciones de la biblioteca estándar de C (C11)
- Llamadas `POSIX` y/o `glibc` a considerar: `exit`
- Para implementar `exit`, escribe la función `run_exit()` e inserta llamadas a la misma en `simplesh.c` donde sea necesario
- Nótese que `exit` sólo termina el proceso que ejecuta la llamada
- Verifica que `simplesh` finaliza siempre que encuentra el comando interno `exit` (el segundo `echo` no se ejecuta)

```
simplesh> echo 1 ; exit ; echo 2
1
[bash]
```

- Comprueba que tras ejecutar `exit` se libera **toda la memoria**

El comando interno cd

- (0.5p) EJERCICIO 5 (a): Implementa el comando interno `cd dir`:

```
simplesh> cwd
cwd: /home/usuario
simplesh> cd directorio
simplesh> cwd
cwd: /home/usuario/directorio
```

- Para ello, únicamente se pueden usar llamadas al sistema POSIX o funciones de la biblioteca estándar de C (C11)
- Llamadas `POSIX` y/o `glibc` a considerar: `chdir()`
- Para implementar `cd`, escribe la función `run_cd()` e inserta llamadas a la misma en `simplesh.c` donde sea necesario
- Nótese que `chdir()` sólo cambia el directorio actual del proceso que ejecuta la llamada
- Verifica que `simplesh` modifica el directorio actual siempre que encuentra el comando interno `cd`

```
simplesh> cwd ; cd directorio ; cwd ; cd .. ; cwd
cwd: /home/usuario
cwd: /home/usuario/directorio
cwd: /home/usuario
```

El comando interno cd

- (0.5p) EJERCICIO 5 (b): Implementa el comando interno `cd [-]`:

```
simplesh> cd -
run_cd: Variable OLDPWD no definida
simplesh> cwd
cwd: /home/usuario/directorio
simplesh> cd
simplesh> cwd
cwd: /home/usuario
simplesh> cd -
simplesh> cwd
cwd: /home/usuario/directorio
```

- El directorio por defecto es el valor de la variable de entorno `HOME`, mientras que `OLDPWD` contiene el directorio de trabajo previo[3]
- Llamadas `POSIX` y/o `glibc` a considerar: `getcwd()`, `getenv()`, `setenv()` y `unsetenv()`
- Verifica que `simplesh` modifica el directorio actual siempre que encuentra el comando interno `cd`

```
simplesh> cwd ; cd ; cwd ; cd - ; cwd
cwd: /home/usuario/directorio
cwd: /home/usuario
cwd: /home/usuario/directorio
```

[3] Cuando se inicia `simplesh`, se debe eliminar la variable de entorno `OLDPWD`. Sólo tras la ejecución con éxito del primer comando interno `cd`, se definirá la variable `OLDPWD` con el directorio de trabajo inicial. A partir de entonces, la variable `OLDPWD` se actualizará cada vez que se ejecute con éxito el comando interno `cd`, asignándole el directorio de trabajo previo.

Notas (I)

¿Por dónde empiezo a implementar los comandos internos?

1. Escribe la función `run_cwd()` **respetando fielmente** el enunciado
2. Verifica que la función `run_cwd()` funciona correctamente
3. Escribe una función que dado un comando, determine si es interno
4. Usa la función anterior para saber cuando llamar a `run_cwd()`
5. Analiza cada llamada a la función `exec_cmd()` dentro de `run_cmd()` para saber dónde hay que llamar a `run_cwd()`, para ejecutar el comando interno `cwd`, en lugar de llamar a `exec_cmd()`
6. Escribe e inserta llamadas a las funciones `run_exit()` y `run_cd()` siguiendo la misma estrategia que con `run_cwd()`
7. Asegúrate de que los tres comandos internos del boletín exhiben el comportamiento esperado con `./simplesh.py -i boletin2.json`
8. Comprueba que no hay fugas de memoria con `./valgrind.sh`

Notas (II)

Comportamiento esperado de los comandos internos en `simplesh`

- El comportamiento de `simplesh` cuando se ejecuta cualquiera de los comandos internos tiene que ser similar al de `bash` en estos casos:

```
simplesh> echo 1; exit; echo 2
1
[bash]
# El comando "echo 2" no se ejecuta.

simplesh> echo 1 | exit
simplesh>
# El comando "echo 1" se ejecuta pero su salida es ignorada por "exit".
# El comando "exit" se ejecuta pero simplesh no finaliza su ejecución.

simplesh> exit &
simplesh>

simplesh> (exit)
simplesh>
# El comando "exit" se ejecuta pero simplesh no finaliza su ejecución.

simplesh> exit > salida
[bash]
# La salida de "exit" se redirige al fichero salida que se crea con tamaño cero.
# Se requiere el uso de llamadas para manipulación de descriptores de fichero.
```