Práctica de Sistemas Operativos
3er Curso, Ingeniería Informática

Descripción

La práctica consiste en la realización de un intérprete de comandos simple para Plan 9 en el lenguaje de programación C. El trabajo de dicho programa consiste en leer líneas de la entrada estándar y ejecutar los comandos pertinentes. Para leer de la entrada estándar es obligatorio usar la biblioteca *bio(2).*

Parte obligatoria

Para solicitar la ejecución del programa `comando` con un número indeterminado de argumentos `(arg1 arg2 ...),` la línea de comandos tendrá la siguiente sintaxis:

```
comando arg1 arg2 ...
```

Dicho comando podrá ser, en este orden de prioridad, un *built-in* de el shell, un fichero ejecutable que se encuentre en el directorio de trabajo, o un fichero ejecutable que se encuentre en el directorio `/bin.`

Una línea de comandos podrá describir en algunos casos un *pipeline* (tubería):

```
cmd1 arg11... | cmd2 arg21 arg22... | cmd3 arg31... | ...
```

Dicho *pipeline* habrá de ejecutarse creando un proceso por cada comando, de tal modo que la salida estándar de cada comando esté alimentada (mediante un pipe) a la entrada estándar del siguiente. En general, un *pipeline* se debe considerar como un único comando.

Un comando puede tener una indicación de redirección de entrada estándar como se muestra en este ejemplo:

```
cmd1 arg1   < fichero
```

que indica que la entrada del comando ha de proceder de `fichero`.

Un comando puede tener adicionalmente una indicación de redirección de salida estándar como se muestra en este ejemplo:

```
cmd1 arg1   >  fichero
```

que indica que la salida estándar del comando tiene como destino `fichero`.

Las redirecciones pueden aparecer en cualquier orden.

El shell deberá esperar a que una línea de comandos termine su ejecución antes de leer la siguiente, a no ser que dicha línea termine en *ampersand:*

```
cmd arg1 arg2 &
```

Cuando la línea termina en *ampersand*, el comando ejecutado no deberá leer de la consola (`/dev/cons`). En caso de que no se haya redirigido la entrada estándar del comando a otro fichero, el shell debe redirigirla a `/dev/null` para no leer de la consola.

Un ejemplo de línea de comandos que incluye todos los elementos mencionados sería:

```
cat | grep ex | wc < /tmp/fich >/tmp/out &
```

Por último, hay que implementar un comando *built-in* en el shell llamado `cd` que debe cambiar el directorio actual al indicado como argumento en dicho comando. Si no recibe argumentos, cambiará el directorio de trabajo al directorio *home* del usuario.

Entre los caracteres especiales de el shell (pipe, redirección, etc.) y los comandos/argumentos podrá haber cero, uno, o más espacios y/o tabuladores (debe funcionar en los tres casos).

El intérprete debe funcionar correctamente cuando se alimenta su entrada estándar desde otro proceso o directamente de un fichero, esto es, cuando no se está usando de forma interactiva a través de la consola. Dos ejemplos:

```
; cat mi_script | shell
; shell < mi_script
```

siendo `mi_script` un fichero con varias líneas de comandos.

Trabajo opcional I

Las líneas que no utilizan *ampersand*, ni redirección de entrada o salida, podrán terminar en `[` para indicar un *here document.* En este caso, el comando utilizará como entrada estándar las líneas escritas por el usuario hasta aquella que conste sólo de `].` Un ejemplo de este uso es el que sigue:

```
cat | wc -l [
una y
otra lin.
]
```

que ejecuta `cat | wc -l` de tal modo que la entrada estándar de `cat` recibirá las líneas

```
una y
otra lin.
```

como entrada.

Trabajo opcional II

El comando `x=y` deberá dar el valor `y` a la variable de entorno `x`. Para cualquier variable el shell deberá reemplazar `$var` por el valor de `var`. Por ejemplo:

```
cmd=ls
arg=/tmp
$cmd $arg
```

es equivalente a ejecutar el comando

```
ls /tmp
```

Trabajo opcional III

Al ejecutar una línea como la que sigue:

```
a % ls -l
```

el contenido de la variable de entorno `a` será la salida estándar del comando que se indica después del carácter '%'. En este caso, `a` contendría la salida del comando `ls` con el modificador `-l.` Detrás del carácter '%' podrá haber un *pipeline,* por ejemplo:

```
b % ls | grep hola
```

En este caso, la variable de entorno `b` contendrá la salida del comando

```
ls | grep hola
```

Trabajo opcional IV

Mantener una variable de entorno `$result` que debe estar siempre disponible y que contenga el estatus devuelto por el último comando (equivalente a `$status` en el shell de Plan 9).

Incluir los siguientes comandos comandos *built-in* en la shell: `ifok, ifnot, true` y `false.` El comando `ifok` deberá ejecutar sus argumentos como un comando simple si el comando anterior terminó

su ejecución correctamente. Ejemplo:

```
test -e /tmp
ifok ls -l /tmp
```

Deberá ser equivalente a

```
ls -l /tmp
```

cuando el directorio `/tmp` existe, y no deberá ejecutar `ls` en caso contrario. Si `ifok` no ejecuta el comando, la variable `$result` contendrá una cadena vacía (ejecución terminada con éxito). El comando `ifnot` deberá ejecutar sus argumentos como un comando simple si el comando anterior terminó su ejecución de modo incorrecto. Los comandos `true` y `false` deben ser análogos a los comandos UNIX con ese mismo nombre.

Entrega

La entrega está especificada en la página web de la asignatura. El fichero con la función `main` deberá llamarse `shell.c` y el programa deberá compilar con los *flags* `-FVw` del compilador de C de Plan 9 sin *warnings* ni errores. El shell no deberá imprimir absolutamente nada que no sea necesario para informar al usuario de comandos inexistentes u otros errores.

Si ha realizado partes opcionales, entregue un fichero llamado `README` con la lista de partes implementadas.

Mediante la entrega de únicamente la parte obligatoria no se podrá obtener una calificación por encima de 6 sobre 10 en la nota de prácticas de la asignatura. Puede encontrar más detalles sobre la evaluación en la normativa de la asignatura. **La práctica deberá estar entregada a las 23:59 del día anterior al examen de la asignatura.**