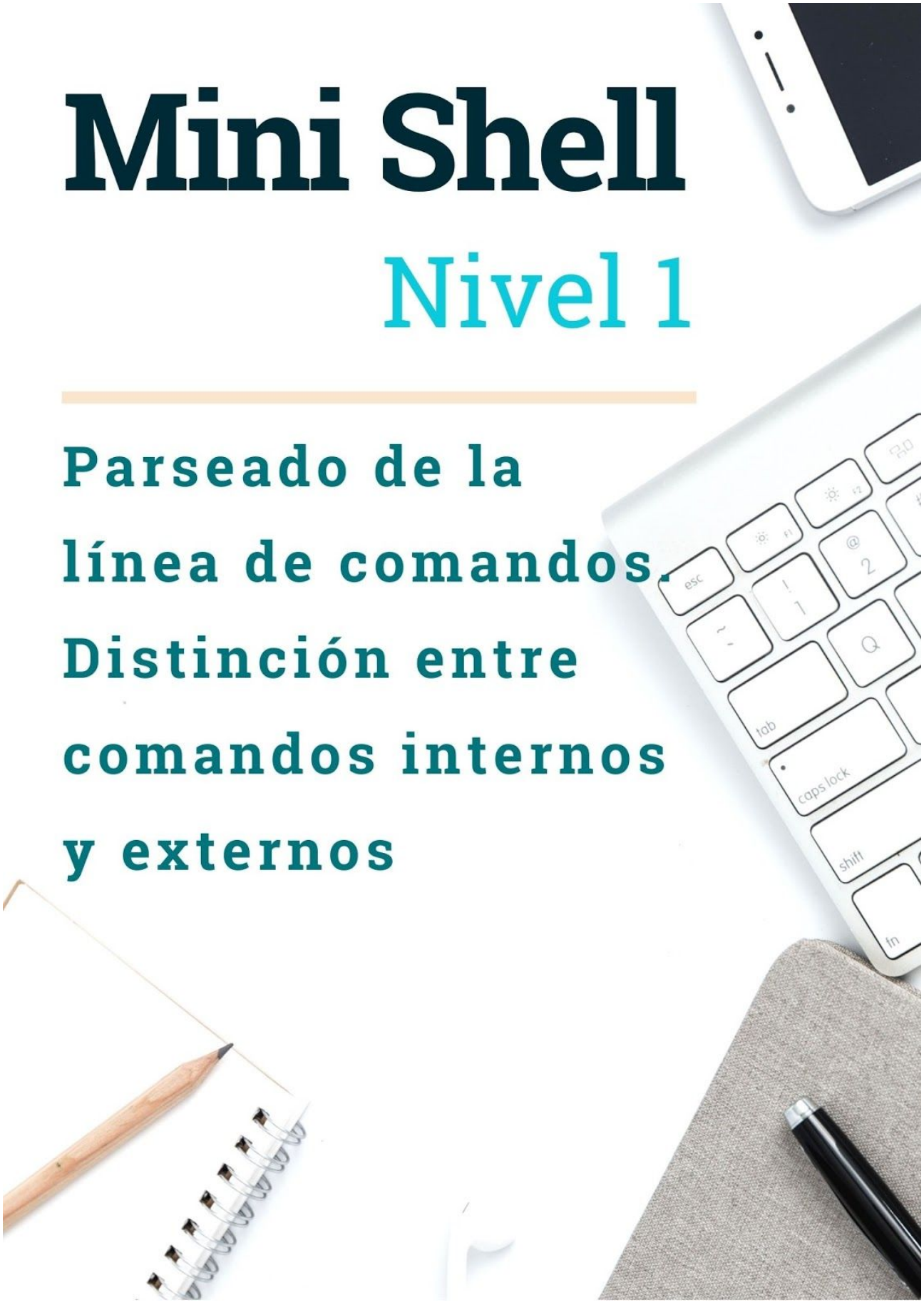
Mini Shell

Nivel 1

Parseado de la línea de comandos. Distinción entre comandos internos y externos



Indice

Mini Shell
Nivel 1

Parseado de la línea de comandos. Distinción comandos internos y externos.

En este primer nivel crearemos el esqueleto de nuestro mini shell. Tendremos un bucle infinito consistente en 2 acciones: leer una línea de comandos desde el terminal y ejecutarla. Para poderla ejecutar, primero tendremos que descomponer esa línea en *tokens* (elementos significativos), y analizar si el primero de ellos (el comando en cuestión) se trata de un comando interno (los que implementaremos nosotros) o uno externo (cuya ejecución externa delegaremos en un proceso hijo).

¿Cuáles son los retos de este nivel?

Hay que leer de manera continua las líneas de comandos y trocearlas en tokens.
Si el comando introducido en una línea es uno de los comandos internos ("**cd**", "**export**", "**source**", "**jobs**", "**fg**", "**bg**" o "**exit**") imprimiremos por pantalla una frase indicando qué hará tal comando.

¿Qué conceptos trabajamos con estos retos?

- Manejo de variables de entorno del sistema en C con **getenv**(). Ejecutar ejemplo
- Lectura de una línea de comandos desde consola con **fgets**()[1]
- Descomposición de una línea en tokens con **strtok**()

- Punteros dobles (manejaremos arrays de strings)[2]
- Uso de colores y negrita en el prompt del sistema

[1] Diferencia entre gets(), fgets() y scanf()
[2] Ver foro: https://stackoverflow.com/questions/33746434/double-pointer-vs-array-of-pointersarray-vs-array

¿Cuáles son los detalles de implementación?

Hay que leer de manera continua las líneas de comandos y trocearlas en tokens mediante la función **strtok**(). El último token de cada línea ha de ser NULL[3] y hay que ignorar los comentarios (precedidos por '#').
Si el comando introducido en una línea es uno de los comandos internos ("**cd**", "**export**", "**source**", "**fg**", "**bg**" o "**jobs**") imprimir por pantalla una frase indicando qué hará tal comando.
Utilizaremos también otro comando interno "**exit**", para salir del mini shell (además de poder hacerlo con **Ctrl+D**)[4].

El **main()** puede ser un bucle del siguiente estilo:

```
char line[COMMAND_LINE_SIZE]; // teniendo #define COMMAND_LINE_SIZE 1024
Mientras (1) hacer
  Si (read_line(line)) entonces
     execute_line(line);
  fsi;
fmientras;
```

Utilizaremos las siguientes funciones:

char *read_line(char *line);
Imprime el prompt.

Lo más simple es usar un símbolo como constante simbólica, por ejemplo: **#define PROMPT '$',** o un **char const PROMPT ='$'**. A la hora de imprimirlo será de tipo carácter, %c, y le podéis añadir un espacio en blanco para separar la línea de comandos.
Opcionalmente se puede implementar una función auxiliar, **imprimir_prompt()**, para crear un prompt personalizado tipo string, yuxtaponiendo variables de entorno como USER, HOME o PWD. El valor de estas variables se puede obtener con la función **getenv**() y también se pueden usar colores. El directorio actual también se puede obtener con la función **getcwd**().

[3] La línea de comandos será ejecutada posteriormente por la función execvp() y ésta no sabe cuántos parámetros recibiŕa pero lo detecta mediante el NULL al final de todos. NULL es una macro definida en los archivos de cabecera stdef.h, stdio.h, stdlib.h y string.h que se usa para inicializar un puntero cuando queremos que "no apunte a ningún sitio".

[4] Más adelante trataremos la captura de Ctrl +D para salir del mini shell

Ejemplo[5]:
uib:~/Documentos/Asignaturas/21708-SOI$

Para forzar el vaciado del buffer de salida se puede utilizar la función **fflush**(stdout)).

Lee una linea de la consola (stdin) con la función fgets()[6].
Devuelve un puntero a la línea leída.

int execute_line(char *line);
De momento sólo llama a parse_args()[7] para obtener la linea fragmentada en tokens y le pasa los tokens a la función booleana check_internal() para determinar si se trata de un comando interno.

int parse_args(char **args, char *line);
Trocea la línea obtenida en tokens[8], mediante la función **strtok**(), y obtiene el vector de los diferentes tokens, args[]. No se han de tener en cuenta los comentarios (precedidos por #). **El último token ha de ser NULL**.
En este nivel, muestra por pantalla el **número de token** y su **valor** para comprobar su correcto funcionamiento (en fases posteriores eliminarlo).
Devuelve el número de tokens (sin contar NULL).

int check_internal(char **args);
Es una función booleana que averigua si args[0] se trata de un comando interno, mediante la función **strcmp**(), y llama a la función correspondiente para tratarlo (internal_cd(), internal_export(), internal_source(), internal_jobs(), internal_fg(), internal_bg()).
En el caso del comando interno **exit** podéis ya llamar directamente a la función **exit**().
La función devuelve 0 o FALSE si no se trata de un comando interno o la llamada a la función correspondiente, cada una de las cuales a su vez devolverá un 1 o TRUE para indicar que se ha ejecutado un comando interno.

[5] Este ejemplo lo he realizado yuxtaponiendo el USER en color azul, ":" en blanco, "~" en crema, el PWD (habiéndole eliminado previamente el HOME de delante mediante una función propia de tratamiento de cadenas) en crema y la constante "$" en blanco. Además lo he puesto todo en negrita.
Hasta el próximo nivel, que implementemos el comando interno cd, no podréis probar la yuxtaposición de la ruta y sus colores.
[6] http://www.taringa.net/post/ciencia-educacion/14450390/Lectura-de-cadenas-en-C.html
[7] Podéis declarar la lista de argumentos como: char *args[ARGS_SIZE];, teniendo #define ARGS_SIZE 64
[8] Consideraremos los siguientes separadores: \t \n \r y espacio en blanco (todos en una misma cadena de delimitadores yuxtapuestos: " \t\n\r")

int internal_cd(char **args); En este nivel, imprime una explicación de que hará esta función (en fases posteriores eliminarla).
int internal_export(char **args); En este nivel, imprime una explicación de que hará esta función (en fases posteriores eliminarla).
int internal_source(char **args); En este nivel, imprime una explicación de que hará esta función (en fases posteriores eliminarla).
int internal_jobs(char **args); En este nivel, imprime una explicación de que hará esta función (en fases posteriores eliminarla).
int internal_fg(char **args); En este nivel, imprime una explicación de que hará esta función (en fases posteriores eliminarla).
int internal_bg(char **args); En este nivel, imprime una explicación de que hará esta función (en fases posteriores eliminarla).

¿Qué recursos metemos en nuestras mochilas?

Documentos de apoyo:

- Cómo usar colores en C
- Cómo colorear el output de la terminal en Linux

Consejos:

- Antes de la inclusión de las librerías incorporad la siguiente directiva para el preprocesador[9]:
  #define _POSIX_C_SOURCE 200112L
- Utilizad las siguientes constantes para indicar el tamaño de la línea de comandos y el nº de elementos del array de argumentos (tokens):
  #define COMMAND_LINE_SIZE 1024
  #define ARGS_SIZE 64

¿Cómo comprobamos que hemos superado este nivel?

- Podéis compilar con: **gcc -o nivel1 nivel1.c** o utilizar el siguiente makefile:

```
CC=gcc
CFLAGS=-c -g -Wall -std=c99
#LDFLAGS=

SOURCES=nivel1.c #nivel2.c nivel3.c nivel4.c nivel5.c nivel6.c my_shell.c
LIBRARIES= #.o
INCLUDES= #.h
PROGRAMS=nivel1 #nivel2 nivel3 nivel4 nivel5 nivel6  my_shell
OBJS=$(SOURCES:.c=.o)

all: $(OBJS) $(PROGRAMS)

#$(PROGRAMS): $(LIBRARIES) $(INCLUDES)
#   $(CC) $(LDFLAGS) $(LIBRARIES) $@.o -o $@

nivel1: nivel1.o
    $(CC) $@.o -o $@ $(LIBRARIES)

#my_shell: my_shell.o
```

[9] https://stackoverflow.com/questions/18948661/what-does-the-flag-d-posix-c-source-200112l-mean

```
#    $(CC) $@.o -o $@ $(LDFLAGS) $(LIBRARIES)

%.o: %.c $(INCLUDES)
  $(CC) $(CFLAGS) -o $@ -c $<

.PHONY: clean
clean:
  rm -rf *.o *~ *.tmp $(PROGRAMS)
```

Observación: El sangrado del makefile se ha de hacer con el tabulador, no con la barra espaciadora

- Podéis ejecutar vuestro programa **nivel1.c** e ir escribiendo en la línea de comandos los ejemplos que se exponen a continuación, comparando vuestros resultados con los mostrados aquí.

```
uib:$ ./nivel1
$ pwd
[parse_args()→token 0: pwd]
[parse_args()→ token 1: (null)]
$ pws #inexistente
[parse_args()→ token 0: pws]
[parse_args()→ token 1: #inexistente]
[parse_args()→ token 1 corregido: (null)]
$
[parse_args()→ token 0: (null)]
$ jobs
[parse_args()→ token 0: jobs]
[parse_args()→ token 1: (null)]
[internal_jobs()→Esta función mostrará el PID de los procesos que no estén en
foreground]
$ source miscomandos.sh
[parse_args()→ token 0: source]
[parse_args()→ token 1: miscomandos.sh]
[parse_args()→ token 2: (null)]
[internal_source()→Esta función ejecutará un fichero de líneas de comandos]
$ export LANGUAGE=en #comentario
[parse_args()→ token 0: export]
[parse_args()→ token 1: LANGUAGE=en]
[parse_args()→ token 2: #comentario]
[parse_args()→ token 2 corregido: (null)]
[internal_export()→Esta función asignará valores a variables de entorno]
$ cd hola#adios
[parse_args()→ token 0: cd]
[parse_args()→ token 1: hola#adios]
[parse_args()→ token 2: (null)]
[internal_cd()→ Esta función cambiará de directorio]
```

```
$ ps
[parse_args()→ token 0: ps]
[parse_args()→ token 1: (null)]
$ ps #comentario1 #comentario2
[parse_args()→ token 0: ps]
[parse_args()→ token 1: #comentario1]
[parse_args()→ token 1 corregido: (null)]
$ ps nocomentario#
[parse_args()→ token 0: ps]
[parse_args()→ token 1: nocomentario#]
[parse_args()→ token 2: (null)]
$ exit
[parse_args()→ token 0: exit]
[parse_args()→ token 1: (null)]
uib:$
```

Observación: Todo lo que aparece entre [ ] son mensajes exclusivamente para comprobar el funcionamiento de este nivel, en los siguientes niveles se han de eliminar.

Anexo 1: Ejemplo de impresión de valores de variables de entorno con C

```c
#include <stdio.h>
#include <stdlib.h>
int main () {
 printf("PWD : %s\n", getenv("PWD"));
 printf("HOME : %s\n", getenv("HOME"));
 printf("LANGUAGE : %s\n", getenv("LANGUAGE"));
 printf("USER : %s\n", getenv("USER"));
 return(0);
}
```

También podríamos haber obtenido el valor de esas variables de sitema desde consola con el bash:

```
uib:$ printenv PWD
uib:$ printenv HOME
uib:$ printenv LANGUAGE
uib:$ printenv USER
```