Proyecto
SHELL básico

Tarea 1

Tarea 1

Proceso Padre: Shell

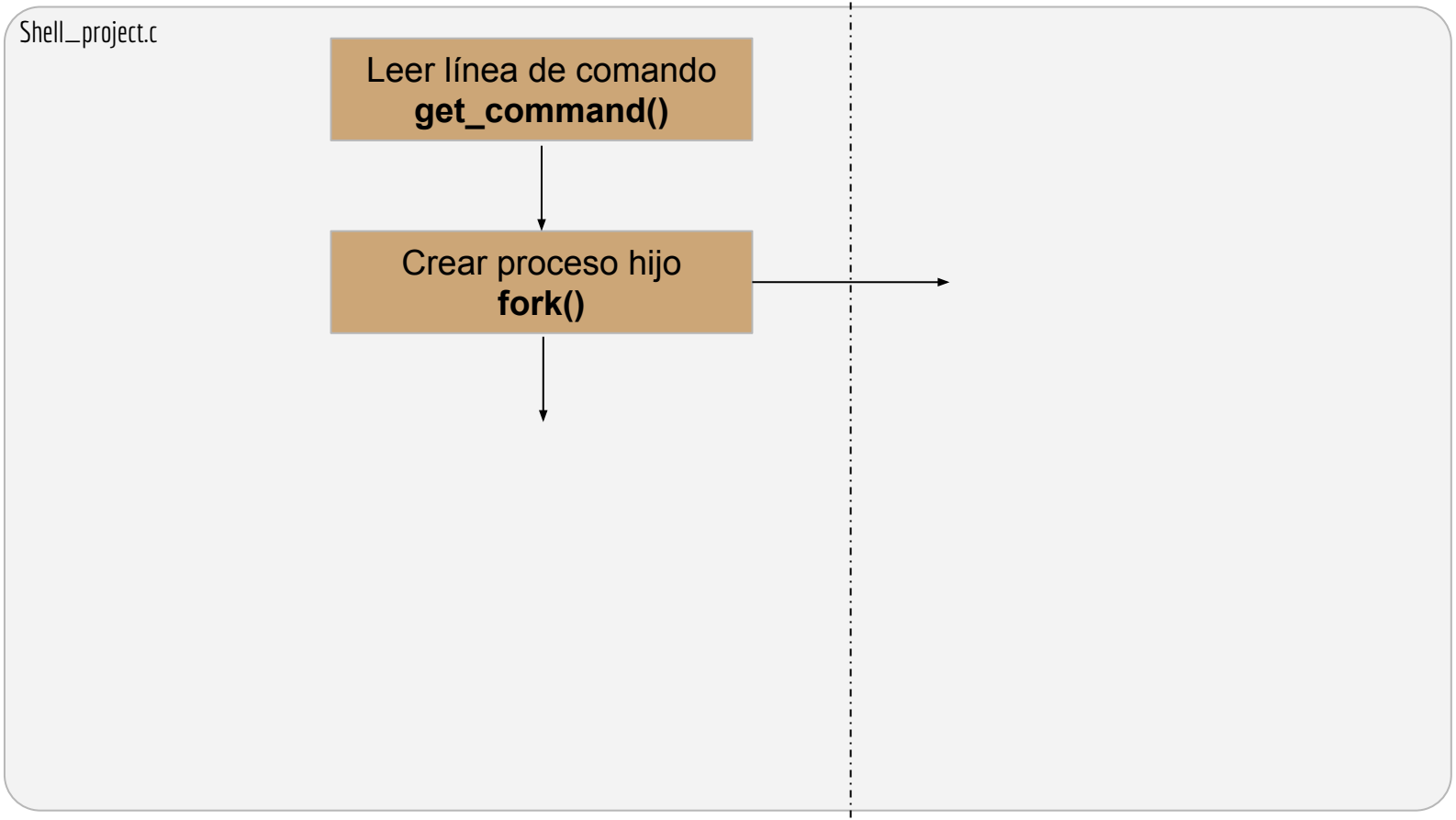
Shell_project.c

Leer línea de comando
get_command()

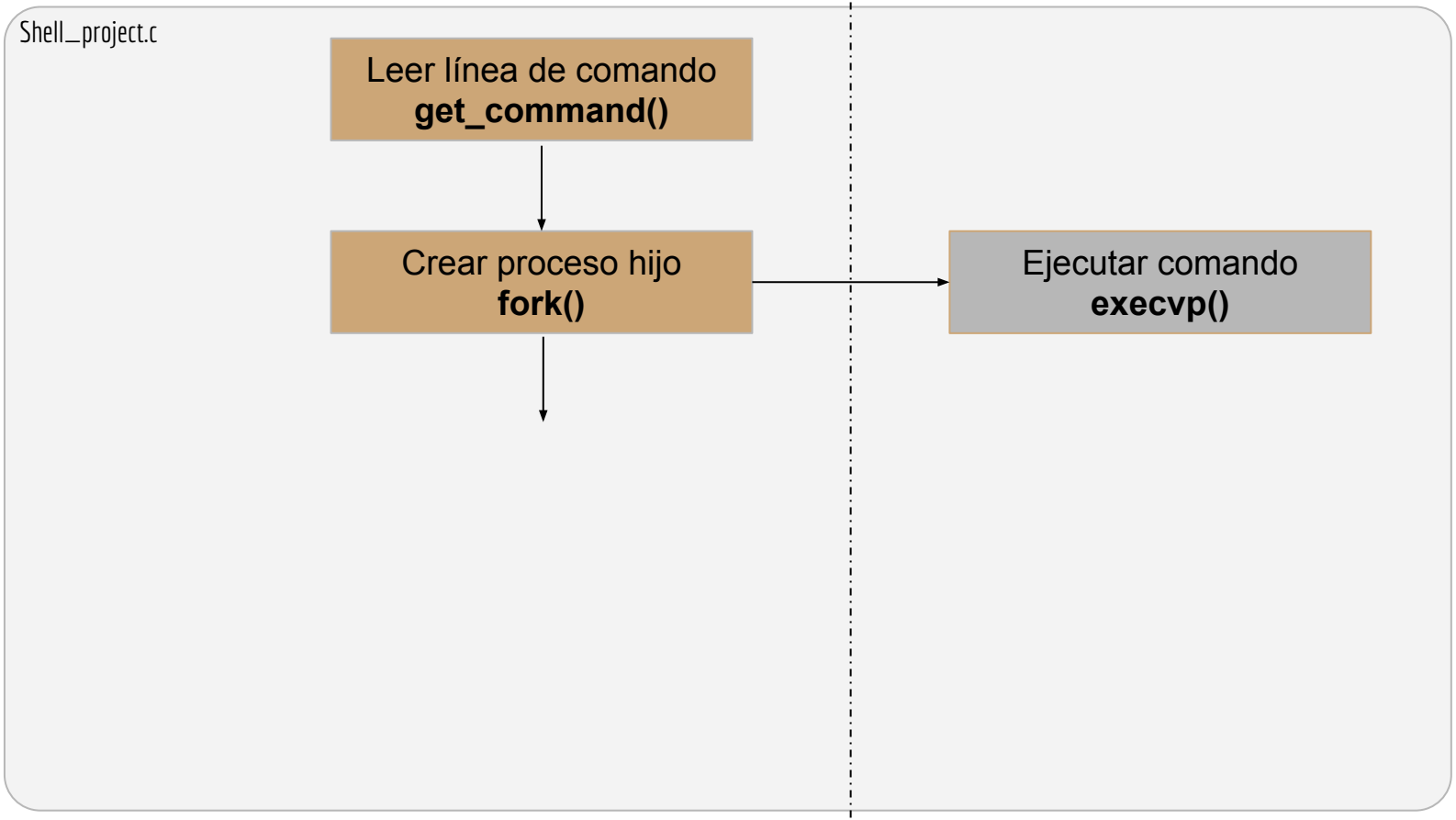
Tarea 1

Proceso Padre: Shell

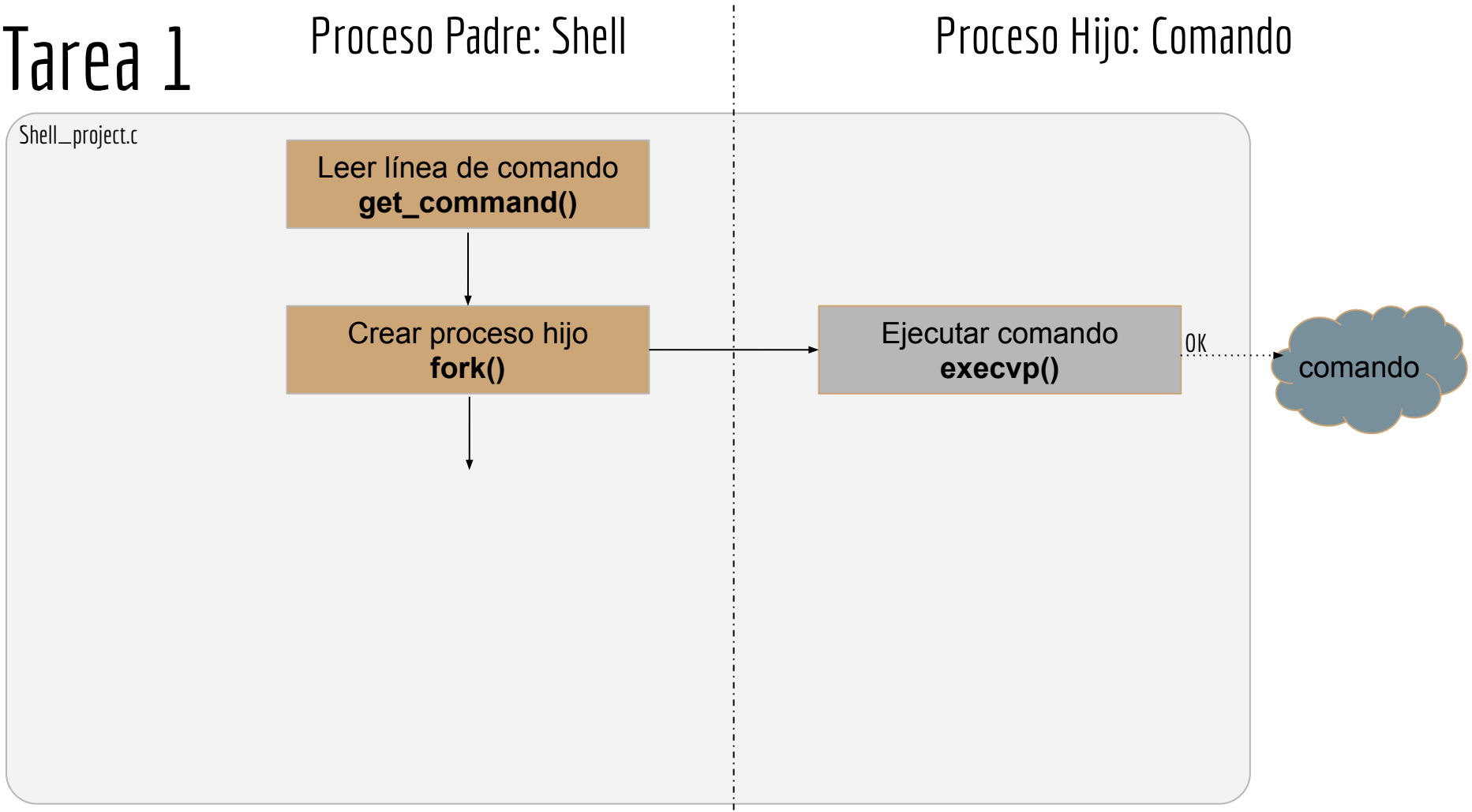
Proceso Hijo: Comando



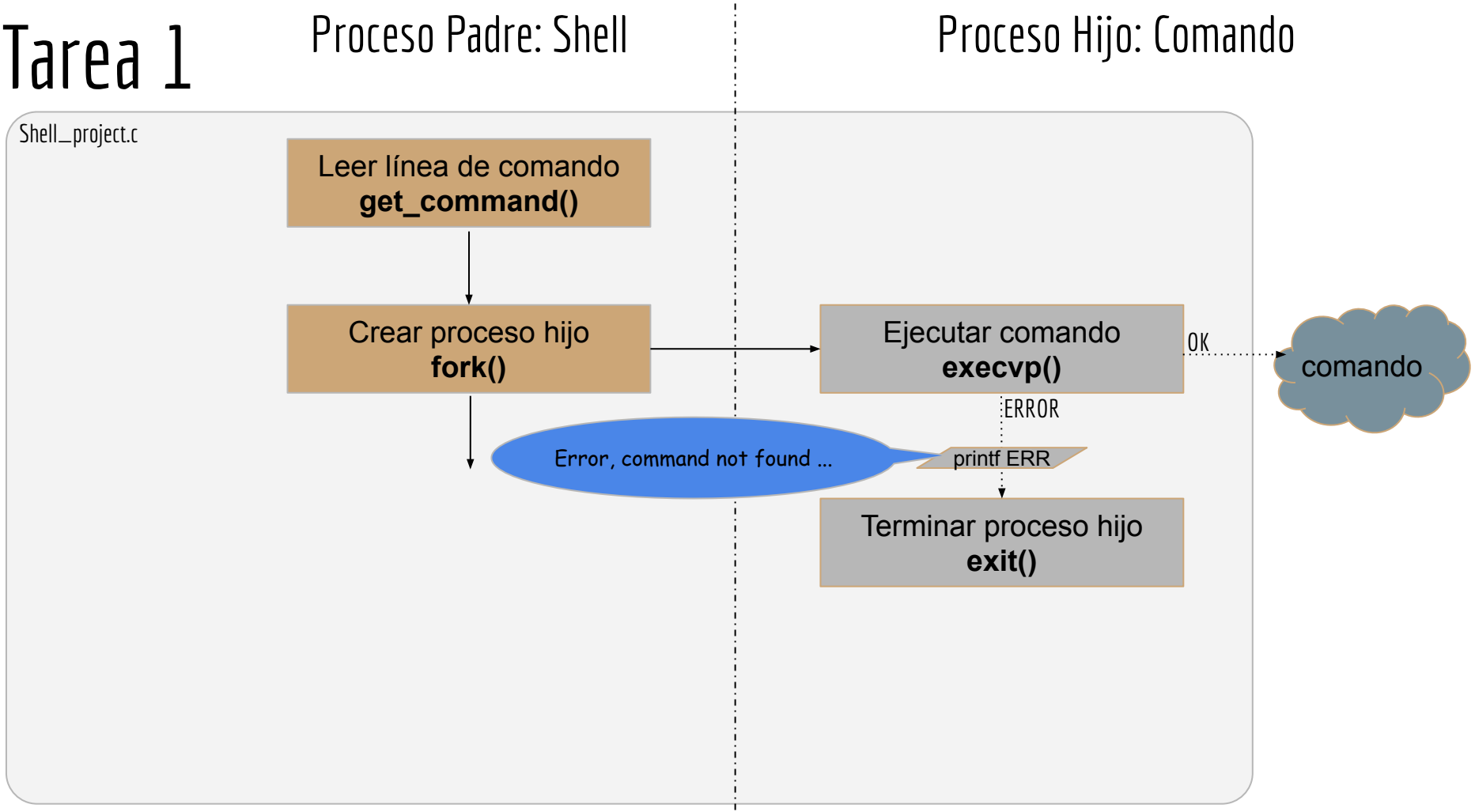
Tarea 1


Proceso Padre: Shell
Proceso Hijo: Comando
Shell_project.c
Leer línea de comando
get_command()
Crear proceso hijo
fork()
Ejecutar comando
execvp()

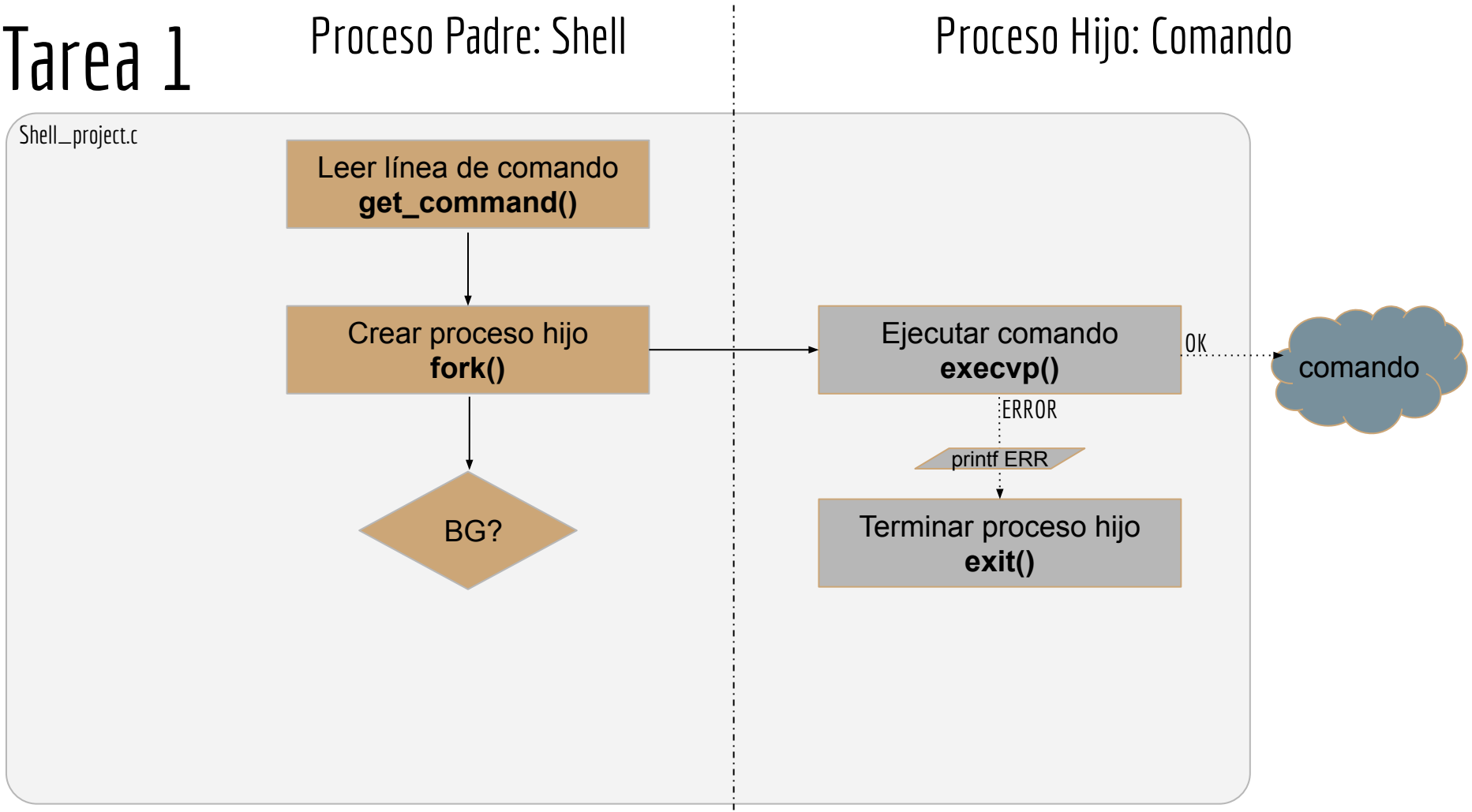
Tarea 1


Proceso Padre: Shell
Proceso Hijo: Comando
Shell_project.c
Leer línea de comando
get_command()
Crear proceso hijo
fork()
Ejecutar comando
execvp()
OK
comando

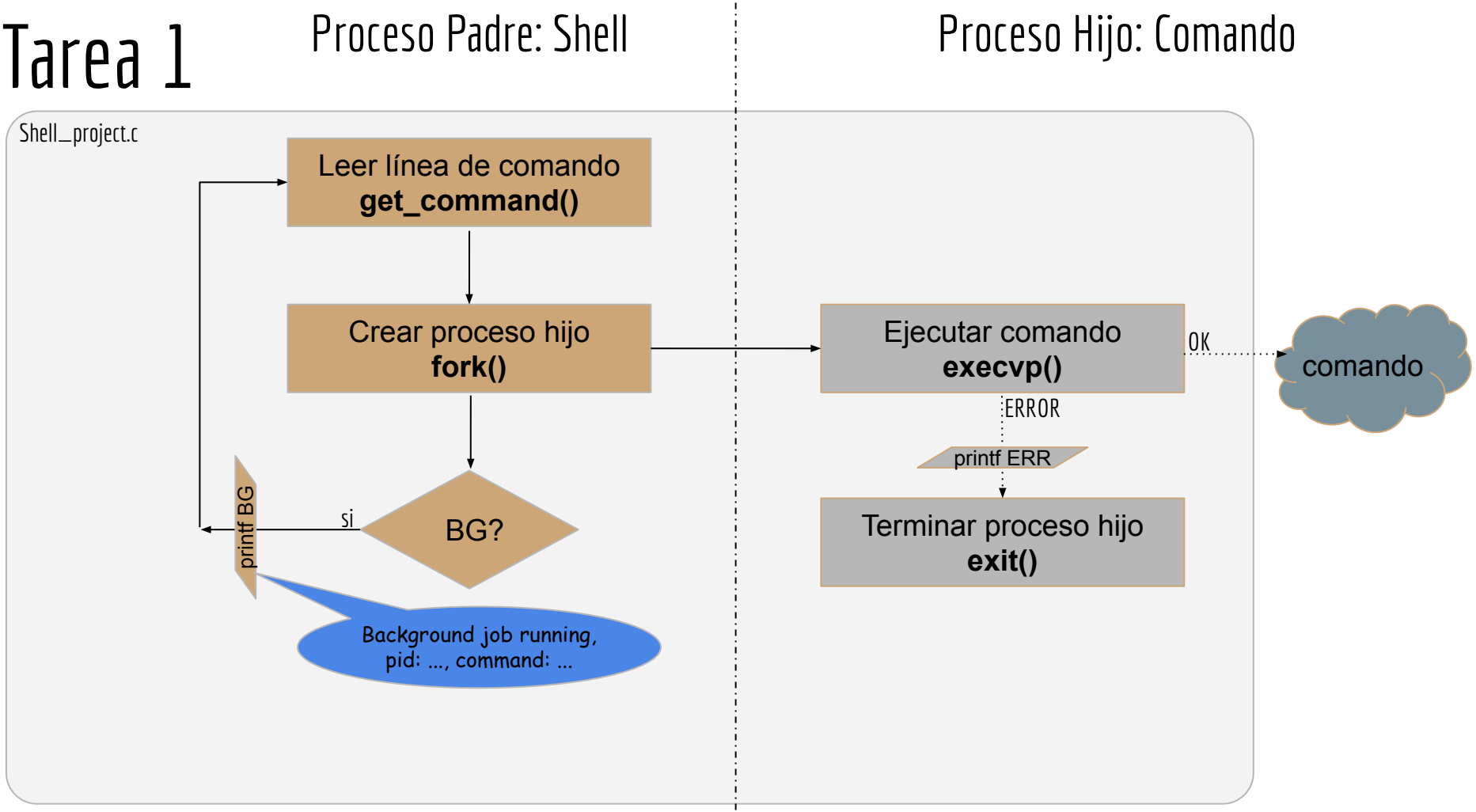
Tarea 1


Proceso Padre: Shell
Proceso Hijo: Comando
Shell_project.c
Leer línea de comando
get_command()
Crear proceso hijo
fork()
Ejecutar comando
execvp()
OK
comando
ERROR
Error, command not found ...
printf ERR
Terminar proceso hijo
exit()

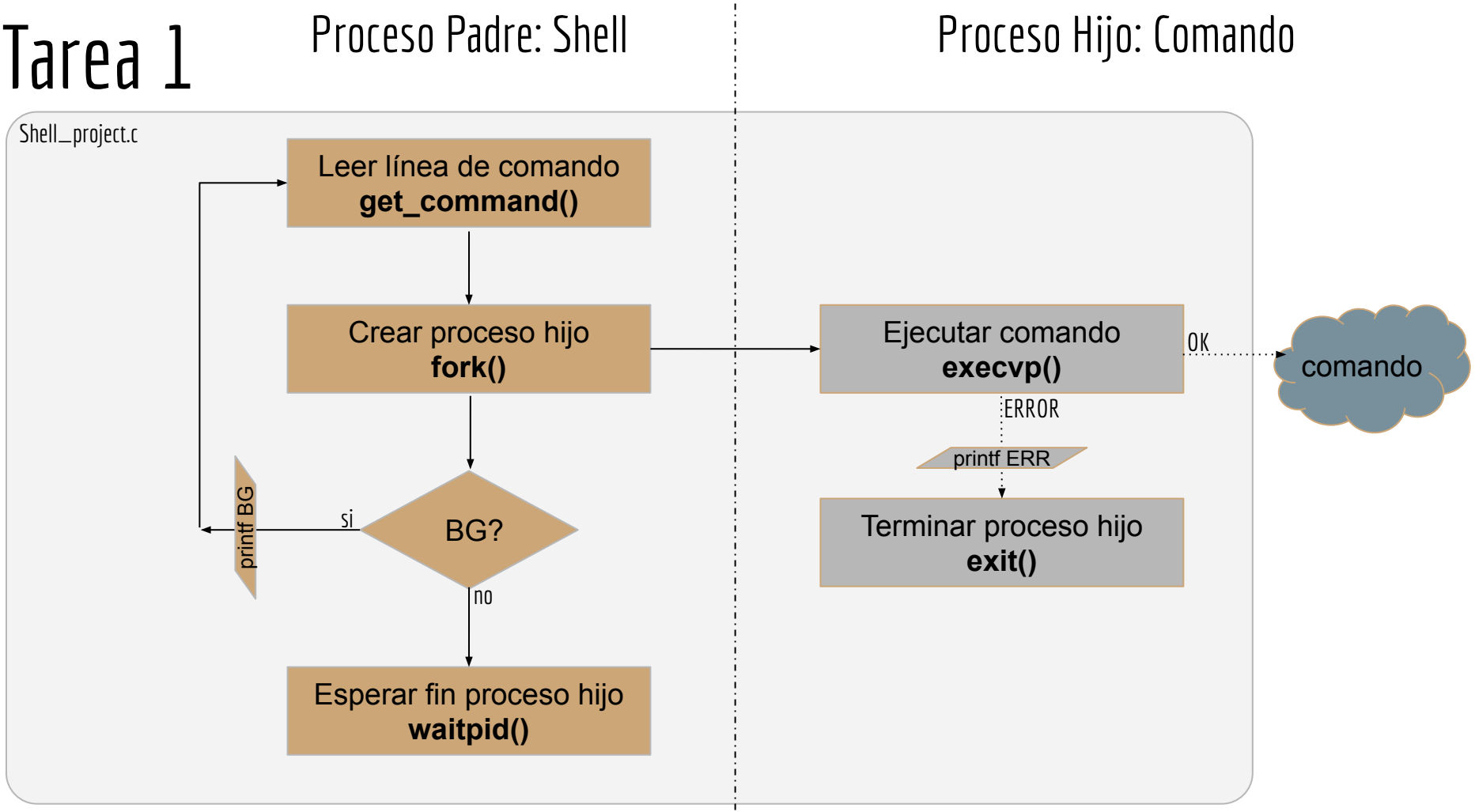
Tarea 1


Proceso Padre: Shell
Proceso Hijo: Comando
Shell_project.c
Leer línea de comando
get_command()
Crear proceso hijo
fork()
BG?
Ejecutar comando
execvp()
OK
comando
ERROR
printf ERR
Terminar proceso hijo
exit()

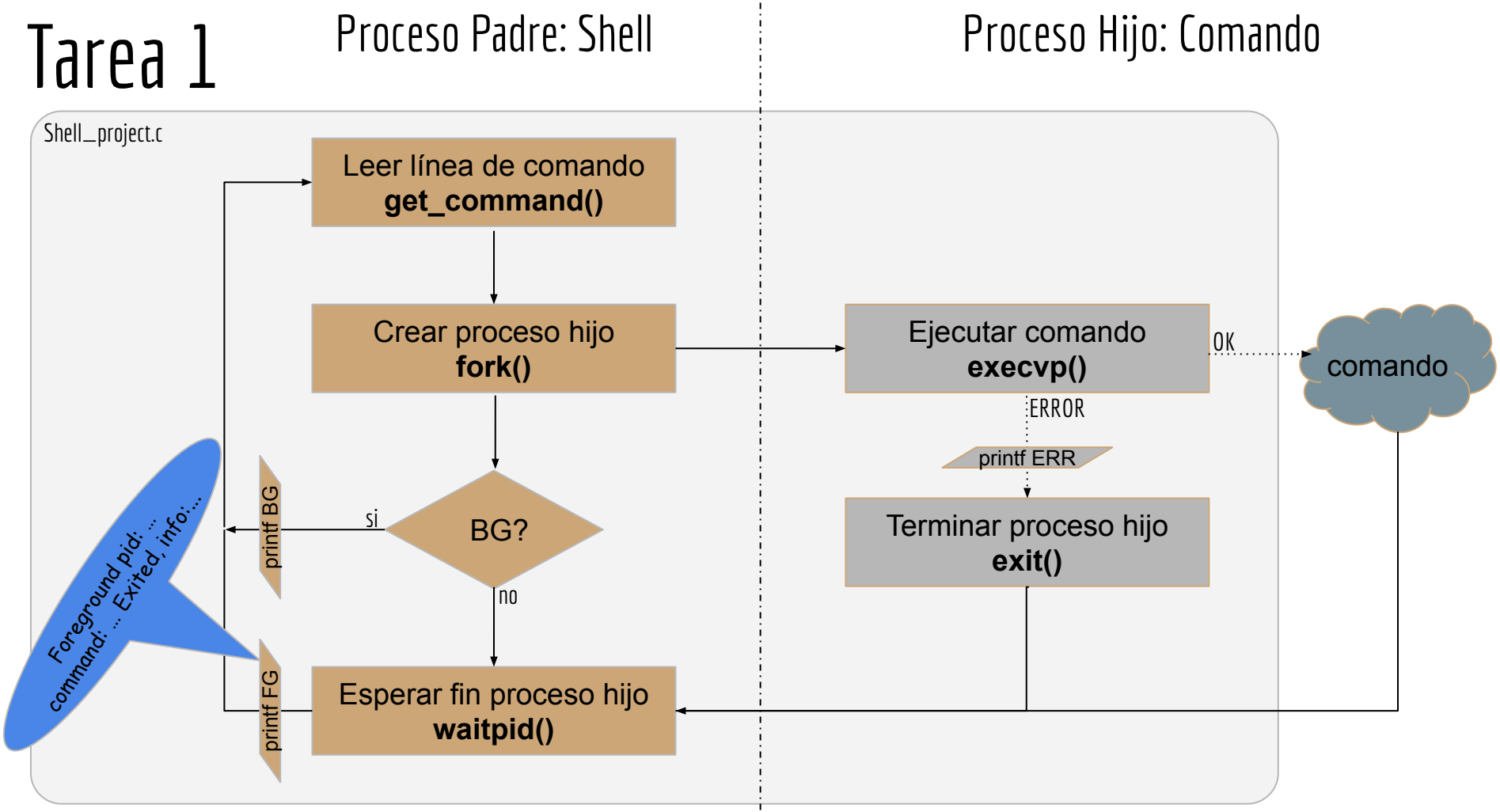
Tarea 1


Proceso Padre: Shell
Proceso Hijo: Comando
Shell_project.c
Leer línea de comando
get_command()
Crear proceso hijo
fork()
BG?
si
printf BG
Background job running,
pid: ..., command: ...
Ejecutar comando
execvp()
OK
comando
ERROR
printf ERR
Terminar proceso hijo
exit()

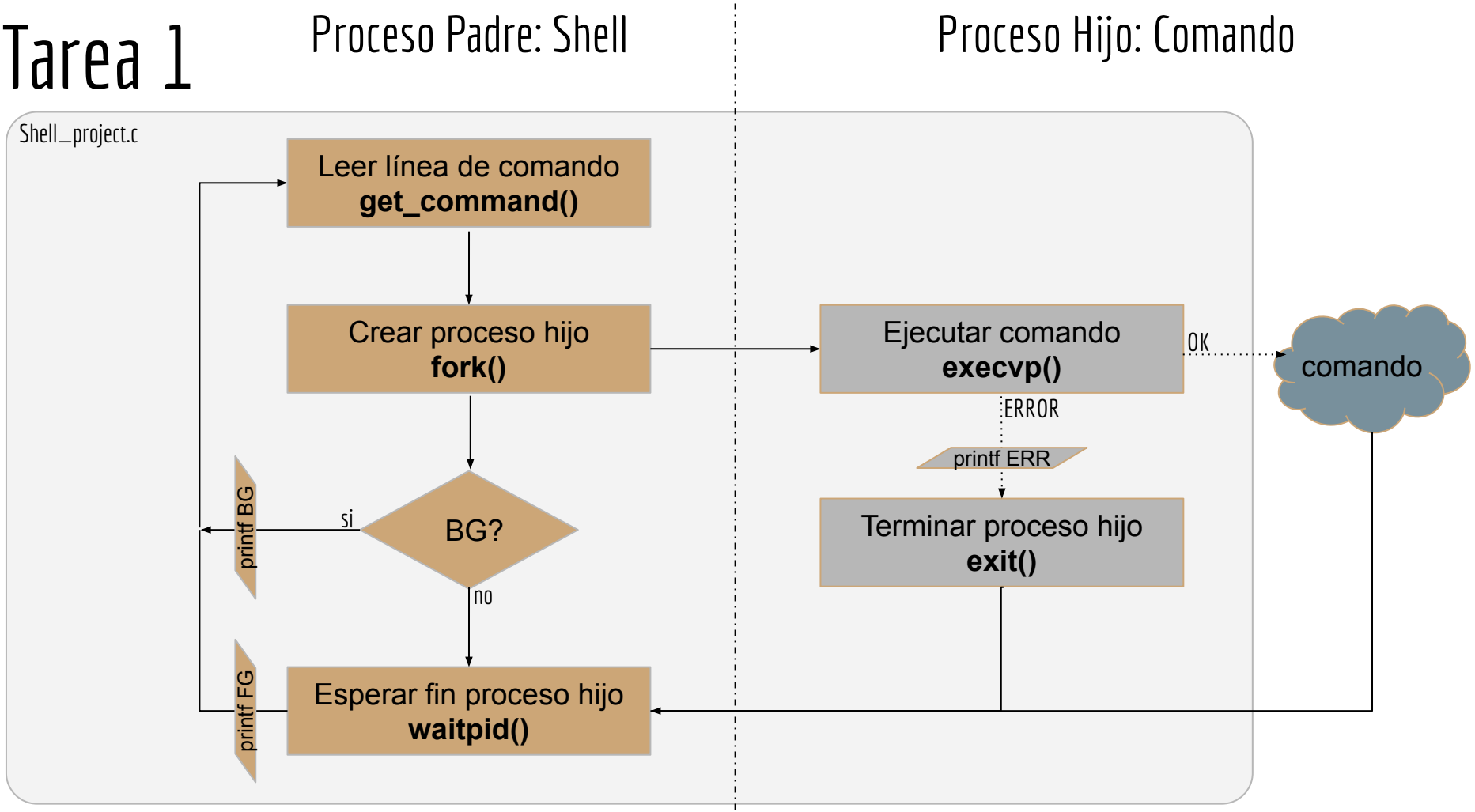
Tarea 1


Proceso Padre: Shell
Proceso Hijo: Comando
Shell_project.c
Leer línea de comando
get_command()
Crear proceso hijo
fork()
BG?
si
printf BG
no
Esperar fin proceso hijo
waitpid()
Ejecutar comando
execvp()
OK
comando
ERROR
printf ERR
Terminar proceso hijo
exit()

Tarea 1


Proceso Padre: Shell
Proceso Hijo: Comando
Shell_project.c
Leer línea de comando
get_command()
Crear proceso hijo
fork()
Ejecutar comando
execvp()
OK
comando
ERROR
printf ERR
Terminar proceso hijo
exit()
BG?
si
printf BG
no
Foreground pid: ...
command: ... Exited, info:...
printf FG
Esperar fin proceso hijo
waitpid()

Tarea 1


Proceso Padre: Shell
Proceso Hijo: Comando
Shell_project.c
Leer línea de comando
get_command()
Crear proceso hijo
fork()
Ejecutar comando
execvp()
OK
comando
ERROR
printf ERR
Terminar proceso hijo
exit()
BG?
si
printf BG
no
Esperar fin proceso hijo
waitpid()
printf FG

Tarea 1

- **waitpid(pid, &status, options)**
  - Espera la finalización/suspensión del proceso hijo cuyo PID se indica en el primer parámetro **pid** (sólo de ese proceso).
  - En la variable **status** (pasada por referencia) nos devuelve el estado de finalización/suspensión del proceso.
  - Con el parámetro **options** podemos modificar el comportamiento de waitpid (por el momento lo pondremos a 0 para un comportamiento “normal”, más adelante veremos varias opciones que vamos a necesitar)
  - Ej: **waitpid(fork_pid, &status, 0)**:
    - Esperará a que termine el proceso cuyo PID está almacenado en la variable **fork_pid** y almacenará en **status** el estado de finalización.

Tarea 1

- execvp(command, args)
  - Intenta ejecutar el fichero ejecutable indicado por la cadena (char *) command en el proceso que ejecuta la función execvp.
  - En el array de cadenas (char **) args se le pasan los parámetros de ejecución del comando (si los hubiera). El primer parámetro (args[0] siempre es el nombre del comando).
  - **Si tiene éxito** la llamada execvp, el código del proceso es sustituido por el del fichero command, por lo que no se ejecuta ninguna de las instrucciones posteriores a execvp.
  - **Si no tiene éxito** (no encuentra comando, no es ejecutable, no tiene permisos, etc...) el proceso continúa ejecutando las instrucciones posteriores a execvp.