UNIVERSIDAD DE MÁLAGA

PRÁCTICA 4

SHELL CON CONTROL DE TAREAS

SISTEMAS OPERATIVOS

EL TERMINAL

- **Historia**
  - **1869: stock ticker → precursor del teletipo**
    - Máquina de escribir conectada por cable a una impresora
    - *Propósito*: distribuir precios de acciones a larga distancia en tiempo real
  - **Teletipo (TTY): comienzos del siglo XX**
    - Basado en ASCII
    - Conectados por todo el mundo:
      - *Red Telex*: red conmutada similar a la telefónica
    - Usados para comunicación de información:
      - Interna de gobiernos e industria
      - Militar
      - Pronóstico del tiempo
      - Prensa
      - Policía

Teletipos en la WWII. *Fuente*: Wikipedia

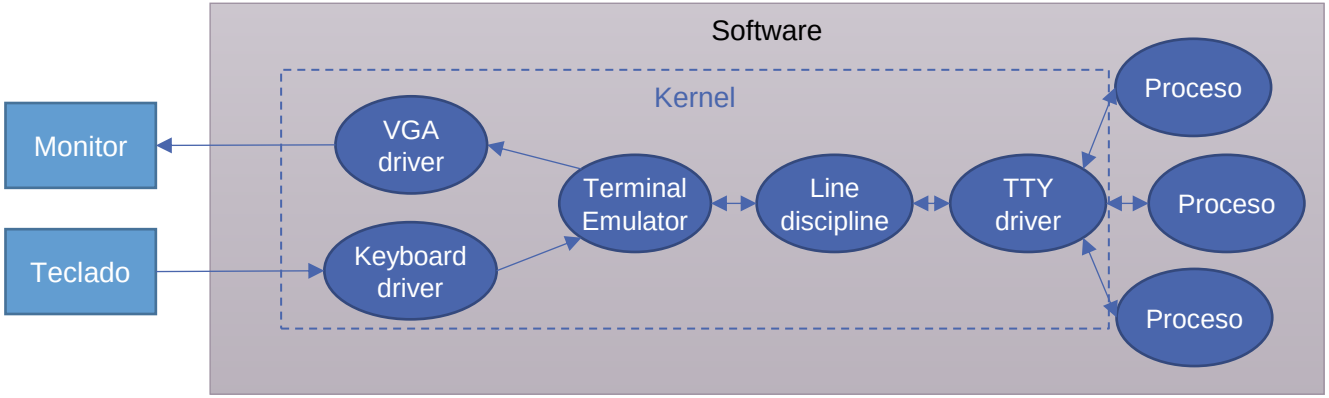
EL TERMINAL

- **Historia**
  - **Con la aparición de los computadores:**
    - En la 3ª gen (1965-71) → interacción con usuarios en tiempo real
    - Primero se utilizan teletipos y luego **terminales** con pantalla y teclado. En el lado del servidor, un **TTY device** se compone de:
      - **UART driver**: Universal Asynchronous Receiver-Transmitter driver. Se utiliza para la comunicación serie entre el terminal y el computador
      - **Line discipline**: búfer de edición que interpreta caracteres (del, ^C,…)
      - **TTY driver**: se encarga de gestión de sesión. Manda señales a procesos, mantiene el que está en primer plano,…

Fuente: http://www.linusakesson.net/programming/tty/index.php

EL TERMINAL

- **En Linux:**
  - Ya no existe el terminal físico → Ahora se **simula** (búfer para *frames* y máquina de estados)
  - Line discipline y TTY driver se mantienen. La UART ya no tiene sentido (aunque se pueden ver los baudios: `stty -a`)
  - **TTY device**: Terminal emulator + Line discipline + TTY driver
    - /dev/tty# → Ctrl+Alt+F# para cambiar entre tty's

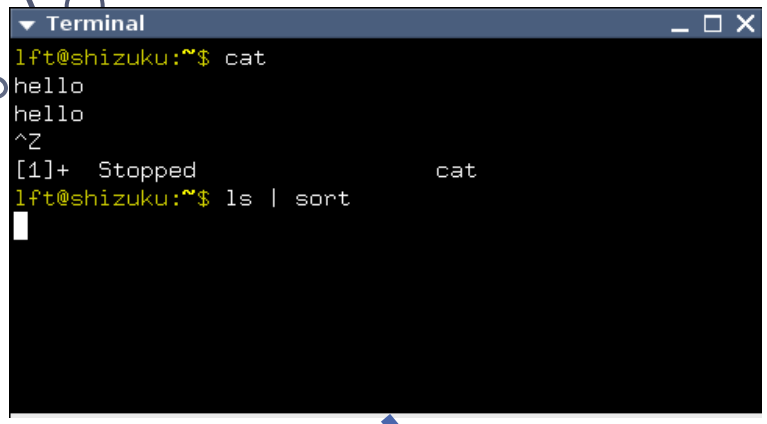


  - **Pseudo-terminal (PTY)**: Terminal llevado al espacio de usuario
    - /dev/pts/# → Lo que se abre en una ventana de terminal (*e.g.*, xterm)
  - Comando `tty`: para conocer el *TTY/PTY* del terminal actual

TERMINAL, SESIÓN Y SHELL

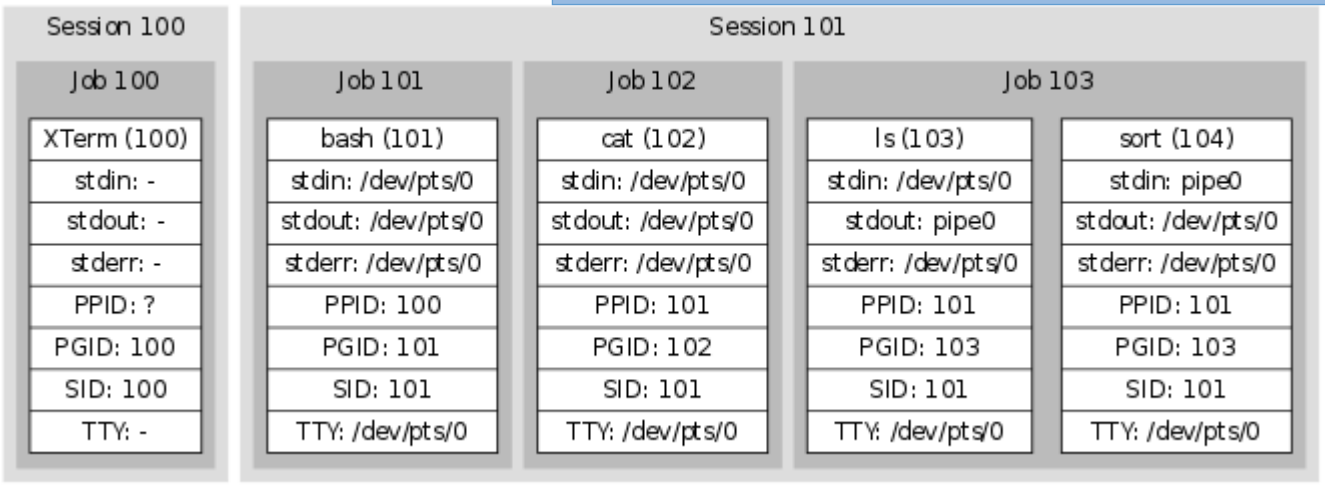
- Cuando se abre un terminal:
  - El demonio de login (*logind*) nos pedirá las credenciales
    - Se crea un identificador de sesión (SID)
  - Si son correctas se inicia un *shell* (e.g., `bash`)
    - El proceso *shell* es el líder de la sesión (PID == SID)
    - Todos sus hijos heredarán el mismo SID, TTY, y stdin, stdout, stderr
    - Si el terminal se cierra, se manda SIGHUP al líder, que mandará SIGHUP a los hijos (véase comando `nohup`)
  - Cuando se teclea un comando en el *shell* se crea un proceso para ejecutarlo (`fork` y `exec`)
    - El comando puede ser una combinación de comandos con *pipe,* `|`
    - O el comando puede hacer `fork` y crear hijos
    - **Control de tareas:** Para facilitar el control de estos **grupos de procesos** se les asigna un *Process Group ID (PGID)*
      - ***Job o tarea****:* conjunto de procesos con el mismo PGID
      - *Ejemplo*: cuando el usuario pulsa ^Z se envía SIGTSTP al grupo/tarea/job

TERMINAL, SESIÓN Y SHELL



Estructuras del kernel

- TTY Device (/dev/pts/0):
  - Size: 45x13
  - Controlling process group: (101)
  - Foreground process group: (103)
  - UART configuration (ignored, since this is an XTerm): Baud rate, parity, word length and much more.
  - Line discipline configuration: cooked/raw mode, linefeed correction, meaning of interrupt characters etc.
  - Line discipline state: edit buffer (currently empty), cursor position within buffer etc.
- Pipe0:
  - Readable end (connected to PID 104 as file descriptor 0)
  - Writable end (connected to PID 103 as file descriptor 1)
  - Buffer



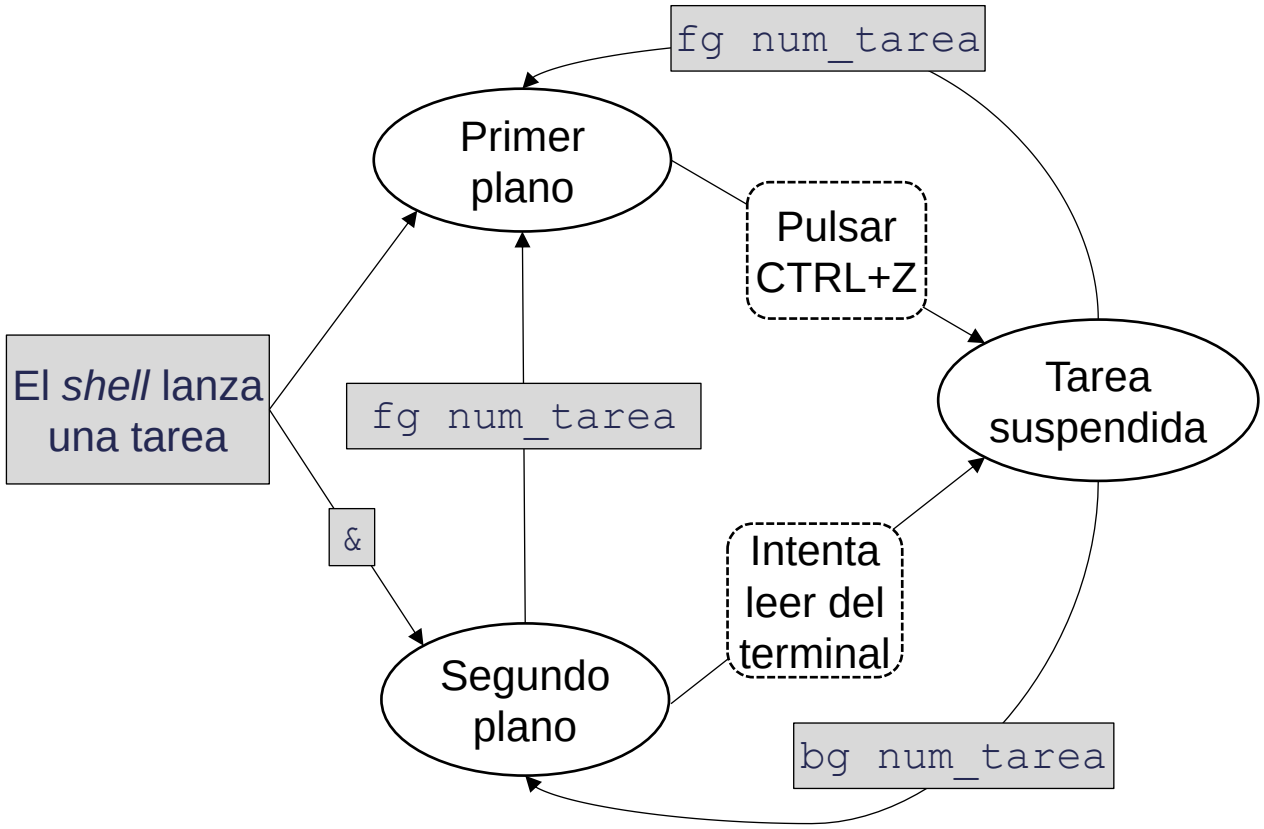
CONTROL DEL TERMINAL

- El **terminal** se asocia a un grupo de procesos (*job*)
  - El grupo de procesos con el terminal es la tarea en **primer plano** (*fg*)
  - El *shell* es la tarea en primer plano mientras lee los comandos
  - Las demás que se ejecutan sin terminal se conocen como tareas en **segundo plano** (*bg*)
- El *shell* **controla** qué tarea (*job*) accede al terminal en cada momento (`tcsetpgrp`)
  - Si un proceso en segundo plano intenta acceder al terminal recibirá `SIGTTOU` y se suspenderá
- El *shell* debe identificar a cada tarea (`setpgid`)
  - Después del `fork`, el trabajo hereda el PGID del *shell*
  - El *shell* debe emancipar a su hijo asignándole otro PGID, para realizar el control de tareas y la asignación del terminal adecuadamente

CONTROL DE TAREAS

- El *shell* debe:
  - Mantener una **lista de tareas** (*job list*)
    - Corriendo en segundo plano (*background*): pueden ser varias
      - Tareas en ejecución sin acceso al terminal
    - Suspendidas (*stopped*): pueden ser varias
      - Puede ser la de primer plano (^Z)
      - Pueden suspenderse las de segundo plano (*e.g.*, `kill -STOP`)
  - Controlar el **estado de las tareas**
    - Sistema de señales: permite controlar los cambios de estado
      - SIGCHLD: notifica al *shell* si un hijo cambia de estado
        - *E.g,* la tarea se suspende, continúa o termina
      - El *shell* debe instalar manejador:
        - `signal(SIGCHLD,manejador)`
    - Comandos internos:
      - `fg` y `bg`: permiten cambiar de plano las tareas
      - `jobs`: permite listar las tareas en segundo plano

DIAGRAMA DE CONTROL DE TAREAS



TERMINAL Y SEÑALES

- Señales generadas por el *TTY driver*:
  - El *shell* debe ignorarlas. Las tareas no
  - Provocadas tras el *parsing* del *Line discipline*:
    - `SIGINT`: carácter INTR (`^C`). *Interrupt* desde terminal
    - `SIGQUIT`: carácter QUIT (`^\`). Como `^C` pero con *core dump*
    - `SIGTSTP`: carácter STOP (`^Z`). Suspender desde terminal
  - Provocadas por procesos:
    - `SIGTTIN`: si un proceso de un *job* en segundo plano intenta leer del TTY
      - El TTY driver le manda esta señal a todo el *job*
      - Acción por defecto: suspensión (*stopped*)
    - `SIGTTOU`: si un proceso de un *job* en segundo plano intenta escribir en el TTY o asignarse el terminal
      - El TTY driver manda esta señal a todo el *job*
      - Acción por defecto: suspensión (*stopped*)
      - Se puede desactivar (desactivada por defecto): `stty -tostop`

LLAMADAS AL SISTEMA

- A usar por el *shell* (se proporcionan *wrappers* para un uso más sencillo)
  - `setpgid(pid, pgid):`

```
#define new_process_group(pid)  setpgid (pid, pid)
```

    - Asigna un ID de grupo (PGID) a un proceso
    - Se usa su propio PID para un nuevo PGID
    - Uso: siempre que creemos una tarea nueva
  - `tcsetpgrp(fd, pgid):`

```
#define set_terminal(pid)  tcsetpgrp(STDIN_FILENO,pid)
```

    - Asigna el terminal a un ID de grupo
    - El terminal se identifica como un *file descriptor*
    - Uso:
      - Siempre que pasemos una tarea a *foreground*
      - Siempre que una tarea *foreground* termine o se suspenda

LLAMADAS AL SISTEMA

- A usar por el *shell* (se proporcionan *wrappers* para un uso más sencillo)
  - `sigprocmask(how,set,oldset)`:

```
#define block_SIGCHLD()      block_signal(SIGCHLD, 1)
#define unblock_SIGCHLD()    block_signal(SIGCHLD, 0)
void block_signal(int signal, int block)
{
    sigset_t block_sigchld;
    sigemptyset(&block_sigchld);
    sigaddset(&block_sigchld,signal);
    if(block) {
        sigprocmask(SIG_BLOCK, &block_sigchld, NULL);
    } else {
        sigprocmask(SIG_UNBLOCK, &block_sigchld, NULL);
    }
}
```

    - Enmascara señales
    - Uso: para proteger el acceso concurrente a la lista de *jobs*

LLAMADAS AL SISTEMA

- Para implementar las redirecciones
  - `fileno(FILE *stream)`:
    - Devuelve el número de descriptor de fichero del *stream*
    - Uso: para pasar los parámetros de `dup2`
  - `dup2(int oldfd, int newfd)`:
    - Cierra el descriptor apuntado por `newfd` de la tabla de ficheros del proceso, y hace que apunte al descriptor con número `oldfd`, de manera atómica
    - Uso: *E.g.*, `ls -la > listado.txt`
      - Se abre `listado.txt` después del `fork` y antes del `exec`
      - Asignar `fileno(stdout)` a `newfd` y el número de descriptor de `listado.txt` a `oldfd`
      - Llamar a `dup2(oldfd, newfd)`. Tras la llamada, cada función que use `stdout` escribirá en `listado.txt` realmente
      - Llamar a `exec`. Los descriptores de ficheros se preservan tras la llamada, por lo que `ls` escribirá en `listado.txt` su salida