Linux For Embedded Systems

For Arabs

Course 102:

Understanding Linux

Ahmed ElArabawy

Lecture 15:
Process Management (Part 1)

On a UNIX system, everything is a file; if something is not a file, it is a process

What is a Process

- A Process is an instance of a running program
- Linux is a multi-tasking OS, This means it can run multiple tasks simultaneously
- The Linux Kernel distribute the processor time among the running processes
- Even a single application, may have multiple threads (for doing multiple actions in parallel)
- In Linux, a thread is just another process (of special nature) so a multi-threaded application is an application that have multiple processes running in parallel
- We can also have multiple instances of the same application running simultaneously in different processes

Process Owner

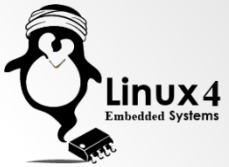
- Linux is a multi-user System, so multiple users can be using the system
- Each user starting a process becomes its owner
- Note that the process owner does not have to be the same as the owner of the binary file for the process
- Each process have an owner, some processes started by the system can be owned by the root user
- The process owner has privileges on his process. He can kill it, pause it, resume it
- The ‘root’ user have super powers on all system processes
- The process inherits its user privileges when trying to access resources (for example when a process tries to write in a file)

Remember: if the process file has the permission “s”, the process inherits its permissions from its file owner (and not the process owner)

Parent & Child Processes

- Processes are organized in parent-child relationships
- Each process that creates another becomes the parent, and the new process becomes the child process
- First process to run is the "***init***" process that is started at system boot… this is the grand parent of all processes in the whole system
- If a process dies, then its orphan children are re-parented to the ***init*** process

Process IDs



- Each Process has a unique number to identify it
- It is called Process ID (***PID***)
- Each process will maintain its **PID** and the **PID** of its parent (***PPID***)
- The **PID** and **PPID** enable us to build the process hierarchy tree
- The **init** process is the parent of all processes, which has

  PID = 1

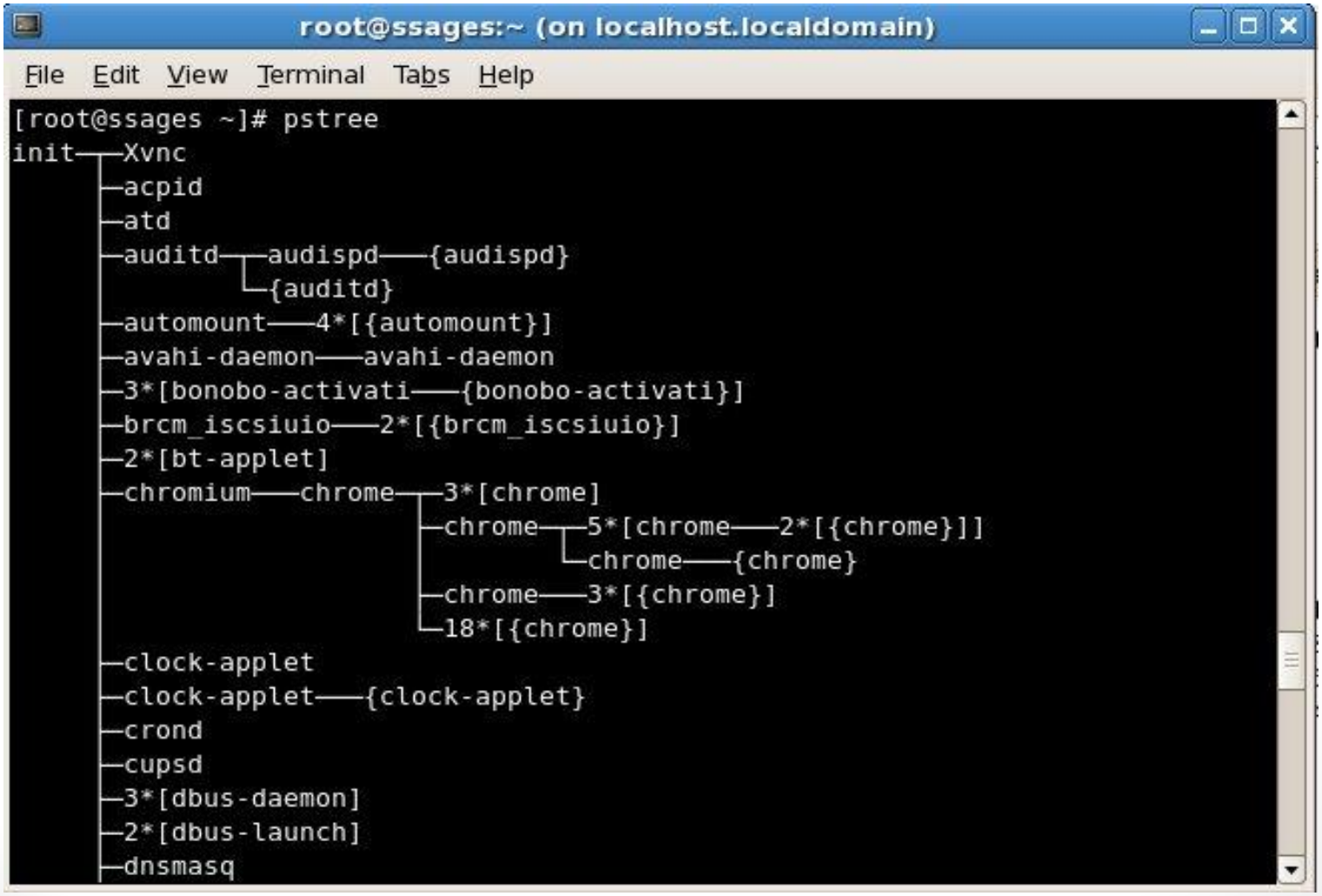
  PPID = 0
- To show the Process tree hierarchy

  $ pstree (Show tree starting at **init** process)

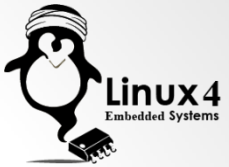
  $ pstree -p (to show PIDs of all processes)

  $ pstree 1000 (Show tree starting at process with PID = 1000)

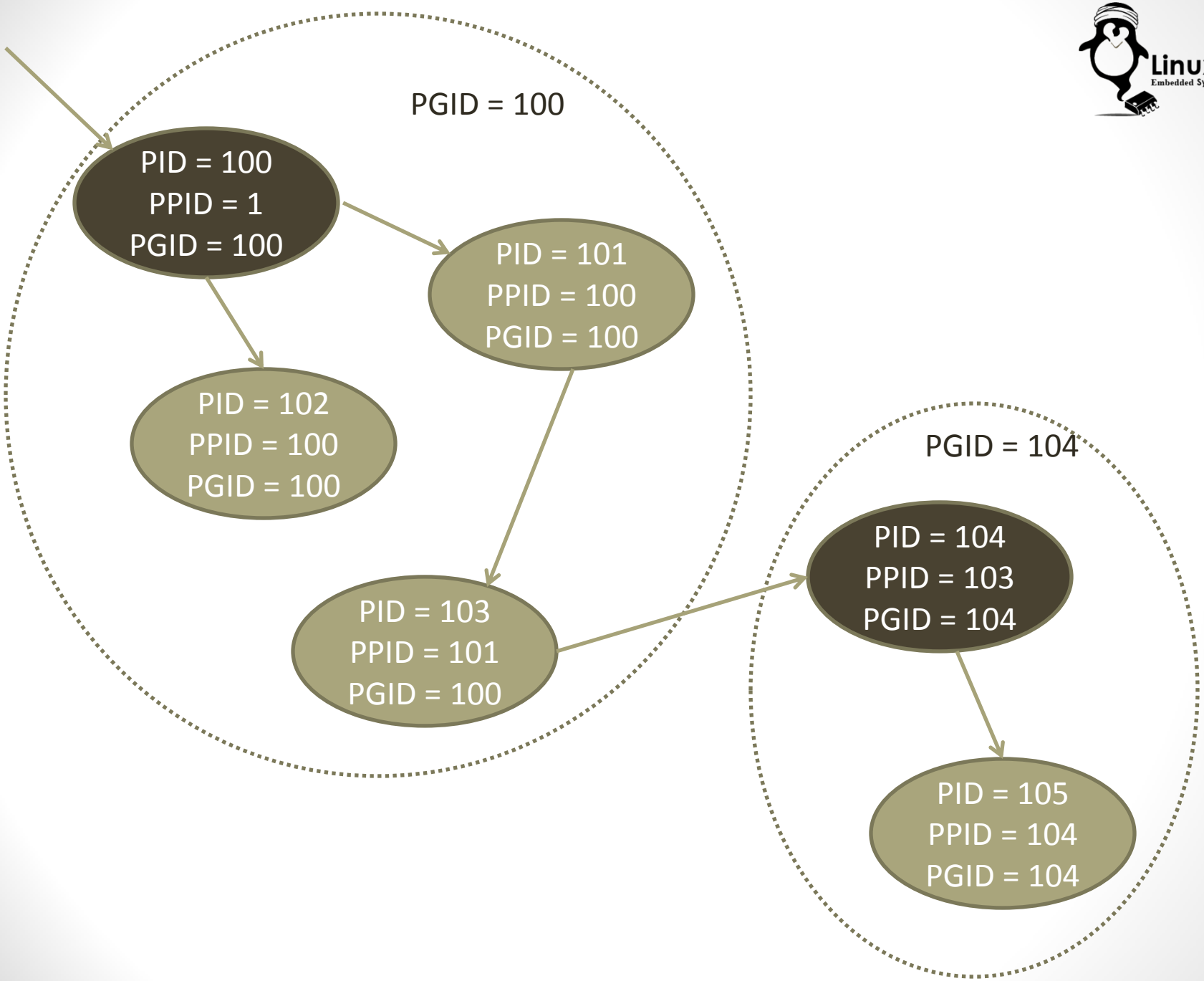
Showing Process Tree (pstree Command)



Process Group

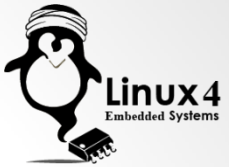


- Process Group is a family of processes (A process, its children, grand-children, …etc)
- When a process is created it becomes a member of the process group of its parent
- Some processes may be started in its new group, and it will detach from its parent group, and become a Process Group Leader
- All descendants will follow the new group
- Each process maintain the ID of its process group (**PGID**)
  - For a normal process, its **PGID** is the same as its parent **PGID**
  - For a Process Group Leader, its **PGID** is the same as its own **PID**


PGID = 100
PID = 100
PPID = 1
PGID = 100
PID = 101
PPID = 100
PGID = 100
PID = 102
PPID = 100
PGID = 100
PID = 103
PPID = 101
PGID = 100
PGID = 104
PID = 104
PPID = 103
PGID = 104
PID = 105
PPID = 104
PGID = 104

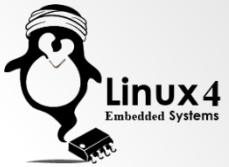
Process Types

- Processes can be classified into one of the following,
  - Interactive Processes
  - Automatic Processes (Batch Processes)
  - Daemon Processes



PROCESS TYPES
INTERACTIVE PROCESSES

Process Types
Interactive Process



- The process is started by a user within a terminal
- It is controlled via that terminal
- It is attached to its terminal, and will be killed if its terminal is closed
- It is called interactive, cause it communicates with the user through the terminal
- Examples:

  $ ls

  $ cat *.log | grep "error" | sort

  $ echo "Good Morning" > my-file

The “Job” Concept

- When a command is issued, the execution of this command is called a Job
- The Job can be,
  - A single process

    $ gedit

    $ cat my-file
  - Multiple connected processes

    $ ls | sort
  - A script that runs multiple processes (within a sub-shell)

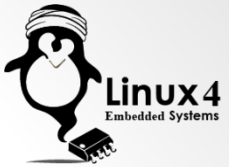
    $ ./my-script
- Jobs can be manipulated in the shell via “Job Control”

The Job Concept

Jobs can run in the,

- Foreground
  - All input and Output of the terminal is exclusively for this job
  - User can not use the terminal for any other activity or start other jobs
  - Only One Job can be a foreground job
  - Initially the shell is in the foreground until a job is launched
- Background
  - Job Input/Output does not utilize the terminal
  - However, it is still attached to the terminal
  - Possibility of multiple Jobs in the background for the same terminal
  - Sometimes it is useful when,
    - The process in the job has a Graphical User Interface and does not need the terminal for its Input/Output
    - The process takes a long time in processing, and user needs to use the terminal for other purposes
    - User needs to launch another job on the same terminal

Job Control



- Start a job in the foreground

  $ gedit

- Start a job in the background

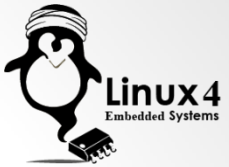
  $ gedit &

- Stop the foreground Job

  $ gedit

  Ctrl-z

Job Control



- Resume the Paused Job in the foreground

  $ gedit

  Ctrl-z

  $ fg

- Interrupt a foreground Job

  $ gedit

  Ctrl-c

- Switch the foreground Job to the background

  $ gedit

  Ctrl-z

  $ bg

Job Control

- List Jobs within the current shell session

  $ jobs

  This will show which job runs in the FG, and which run in BG

- Switch a background job into the foreground

  $ jobs

  $ fg %n (where n is the process number in the list)

- Kill a background Job (all processes in this Job)

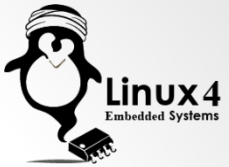
  $ jobs

  $ kill %n

Jobs and Process Groups

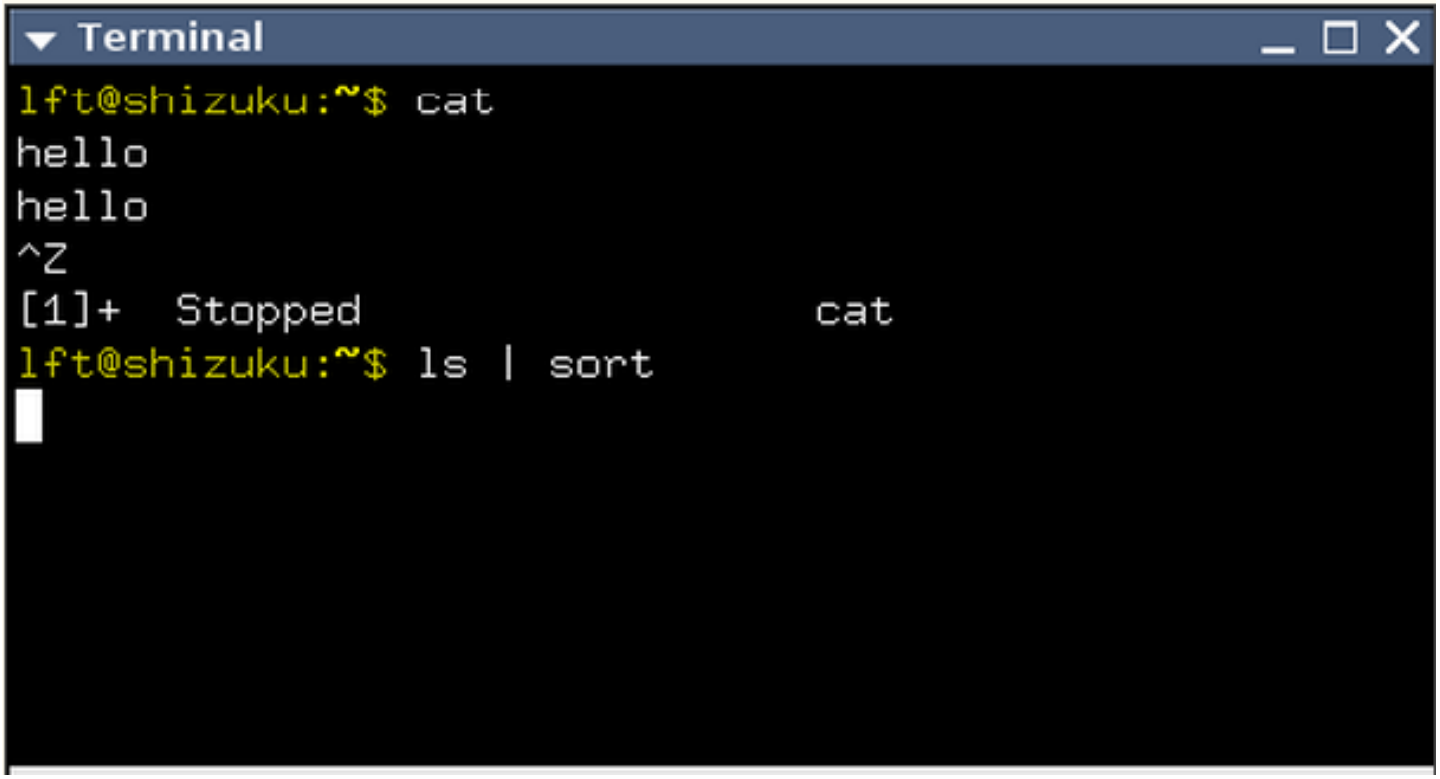
- For each new Job, a new Process Group is created for processes inside this job
- This means that each job has its own process group
- If the Job is about running a script, then the sub-shell that runs the script becomes the Process Group Leader
- When you perform Job control commands (send it to background, stop it, bring it back to foreground. …) , it applies on whole process group for this Job and not on single process

Shell Session

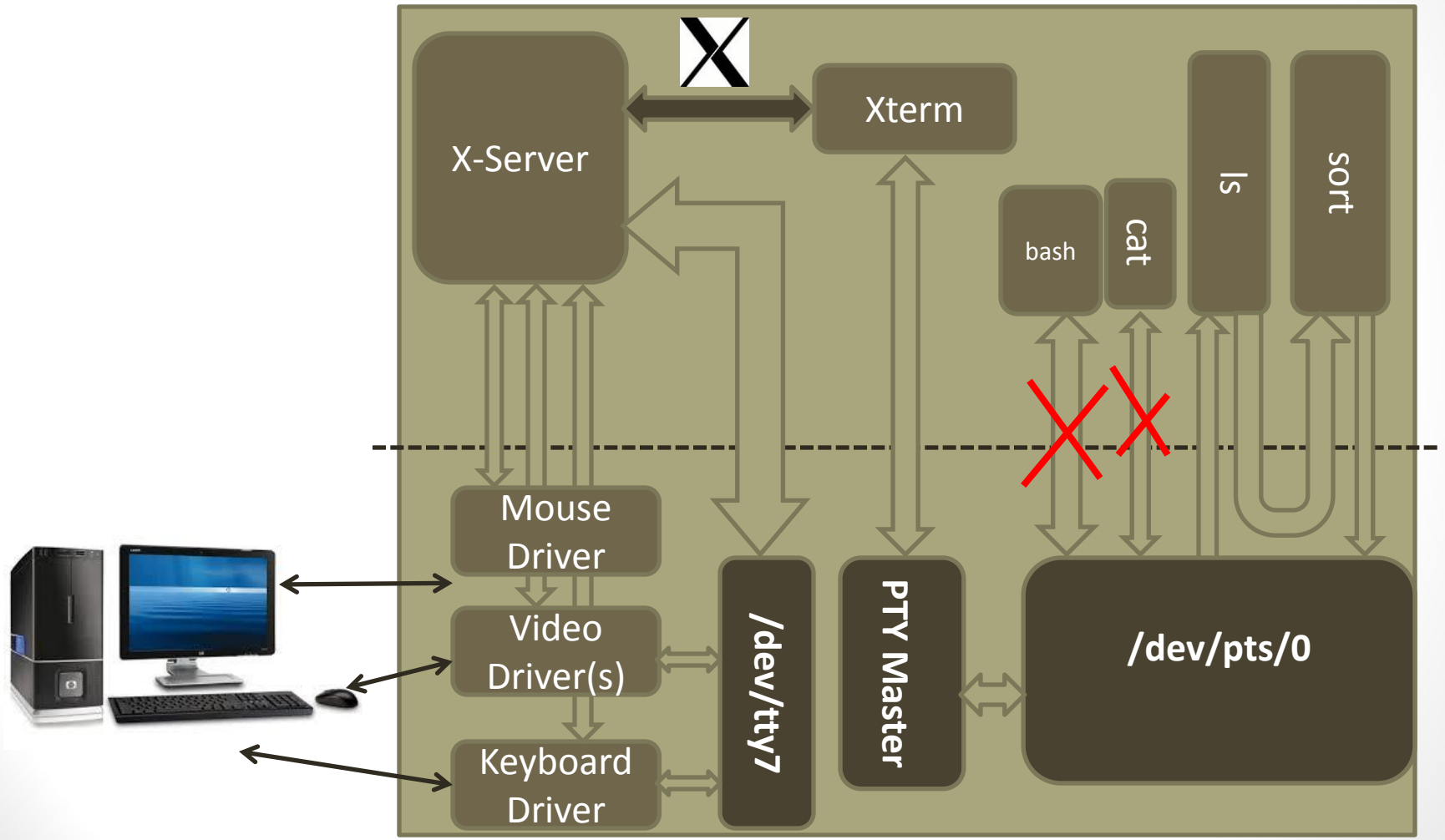


- Each shell has its own session
- All processes running inside this session, will carry the same ***SID*** (which is the PID or the shell owning the session)
- The shell is named the session leader
- A shell session contains one or more Jobs (the ones launched under it)
- Jobs inside a session, one of this happens,
  - All Jobs run in the background, and the shell runs in the foreground
  - A single Job runs in the foreground, while the shell and all other Jobs run in the background

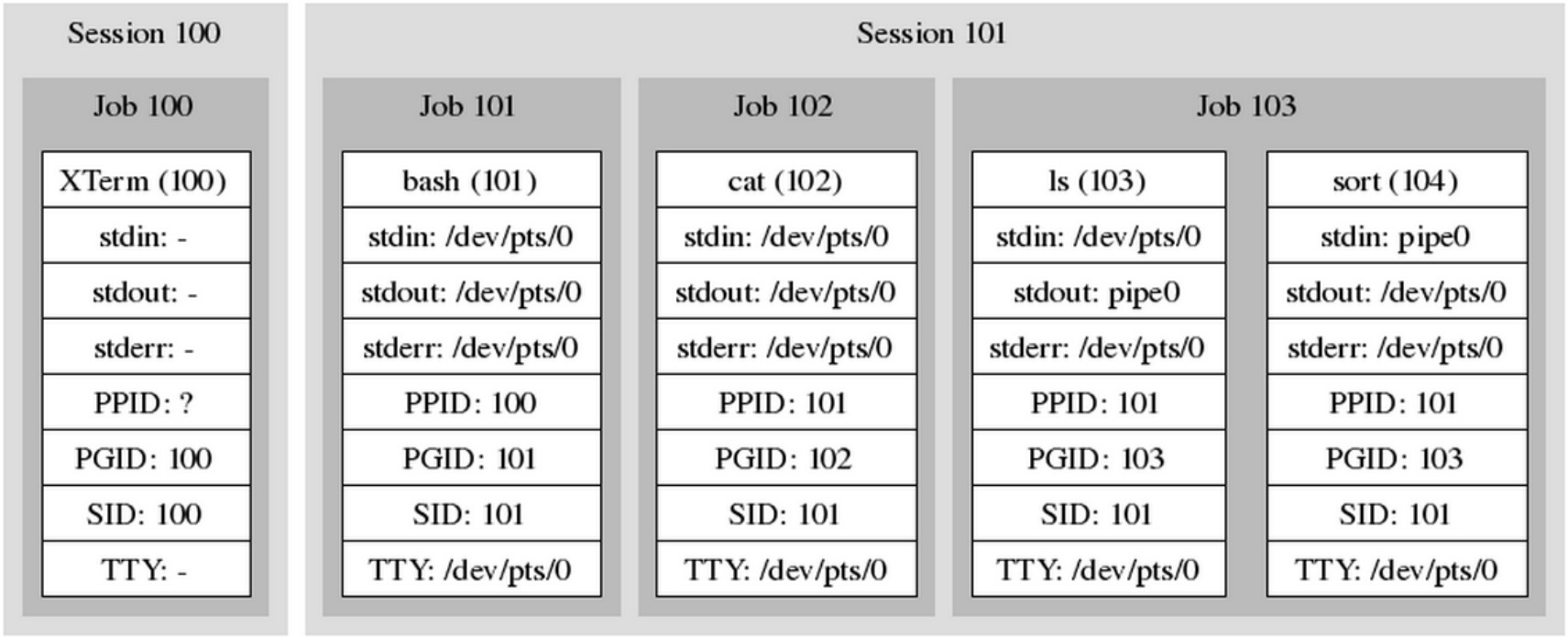
Example:


Terminal
lft@shizuku:~$ cat
hello
hello
^Z
[1]+ Stopped cat
lft@shizuku:~$ ls | sort

Example: An Inside View



Example: An even deeper View



Linux 4 Embedded Systems

http://Linux4EmbeddedSystems.com