In the first example of the ps command:

ps -o pid,ppid,pgid,sid,comm

The parent of the ps command is the shell, which we would expect. Both the shell and  
the ps command are in the same session and foreground process group (949). We say  
that 949 is the foreground process group because that is what you get when you execute  
a command with a shell that doesn’t support job control. If we execute the command in the background, the only value that changes is the process ID of the command. This shell doesn’t know about job control, so the background job is not put into its ownprocess group and the controlling terminal isn’t taken away from the background job.

About how the Bourne shell handles a pipeline:

+ When we execute  
ps -o pid,ppid,pgid,sid,comm | cat1

It appears that the shell forks a copy of itself and that this copy then forks to make each of the  
previous processes in the pipeline.

With job control, this is handled by placing the background job into a background  
process group, which causes the signal SIGTTIN to be generated if the background job  
tries to read from the controlling terminal. The way this is handled without job control  
is that the shell automatically redirects the standard input of a background process to  
/dev/null, if the process doesn’t redirect standard input itself. A read from  
/dev/null generates an end of file. This means that our background cat process  
immediately reads an end of file and terminates normally

+ When we execute

ps -o pid,ppid,pgid,sid,comm | cat1 | cat2

pipeline generates the following output:

|  |  |
| --- | --- |
| PID PPID PGID | SID COMMAND 949 sh |
| 949 | 947 | 949 |

1988 949 949 949 cat2  
1989 1988 949 949 ps

|  |  |  |  |
| --- | --- | --- | --- |
| 1990 | 1988 | 949 | 949 cat1 |

.

**= > the last process in the pipeline is the child of the shell**

\*Note that the order in which a shell creates processes can differ depending on the  
particular shell in use.