Process Groups and Sessions (Chap7-Chap9)

Hongik University

Eunsung Jung

Disclaimer: The slides are borrowed from many sources!

Login Process

- init(8)
  - Reads /etc/ttys
- getty(8)
  1. opens terminal
  2. prints “login: ”
  3. reads username
- login(1)
  1. getpass(3), encrypt, compare to getpwnam(3)
  2. register login in system databases
  3. read/display various files
  4. initgroups(3)/setgid(2), initialize environment
  5. chdir(2) to new home directory
  6. chown(2) terminal device
  7. setuid(2) to user’s uid, exec(3) shell

Login Process

- Process relationship for a login:

```
kernel    ⇒  init(8)     # explicit creation
init(8)   ⇒  getty(8)    # fork(2)
getty(8)  ⇒  login(1)    # exec(3)
login(1)  ⇒  $SHELL      # exec(3)
$SHELL    ⇒  ls(1)       # fork(2) + exec(3)
```

Login Process

- Process relationship for a login:

```
kernel    ⇒  init(8)     # explicit creation
init(8)   ⇒  getty(8)    # fork(2)
getty(8)  ⇒  login(1)    # exec(3)
login(1)  ⇒  $SHELL      # exec(3)
$SHELL    ⇒  ls(1)       # fork(2) + exec(3)
```

Login Process

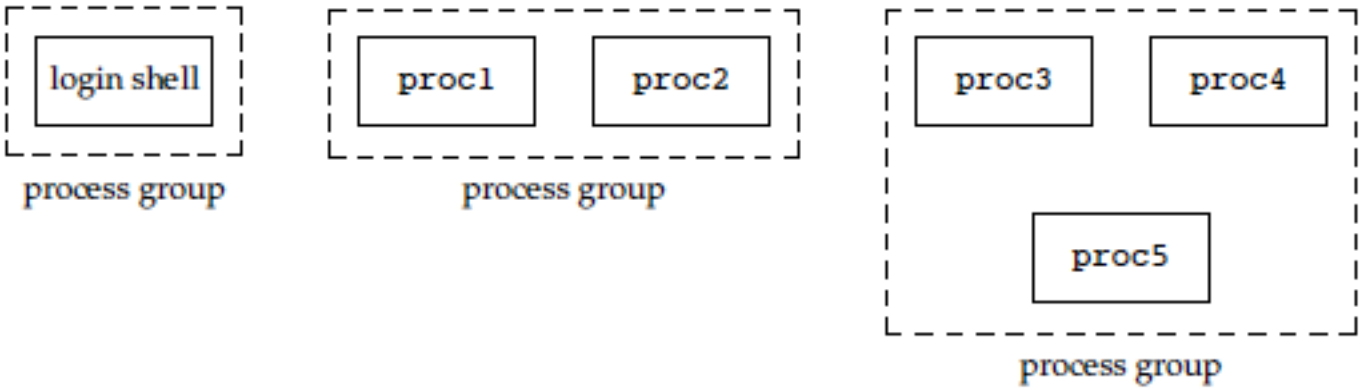
- Process relationship for a login:

```
init(8)      # PID 1, PPID 0, EUID 0

getty(8)     # PID N, PPID 1, EUID 0

login(1)     # PID N, PPID 1, EUID 0

$SHELL       # PID N, PPID 1, EUID U

ls(1)        # PID M, PPID N, EUID U
```

```
pstree -hapun | more
```

Process Groups

- The processes in a process group are usually placed there by a shell pipeline.




```
proc1 | proc2 &
proc3 | proc4 | proc5
```

Process Groups

```
#include <unistd.h>

pid_t getpgrp(void);
pid_t getpgid(pid_t pid);
                                Returns: process group ID if OK, -1 otherwise
```

- In addition to having a PID, each process also belongs to a process group (collection of processes assocaited with the same job /terminal)
- Each process group has a unique process group ID
- Process group IDs (like PIDs) are positive integers and can be stored in a pid_t data type

Process Groups

```
#include <unistd.h>

pid_t getpgrp(void);
pid_t getpgid(pid_t pid);
                                Returns: process group ID if OK, -1 otherwise
```

- Each process group can have a process group leader
  - leader identified by its process group ID == PID
  - leader can create a new process group, create processes in the group
- A process can set its (or its children's) process group using setpgid(2)

Process Sessions

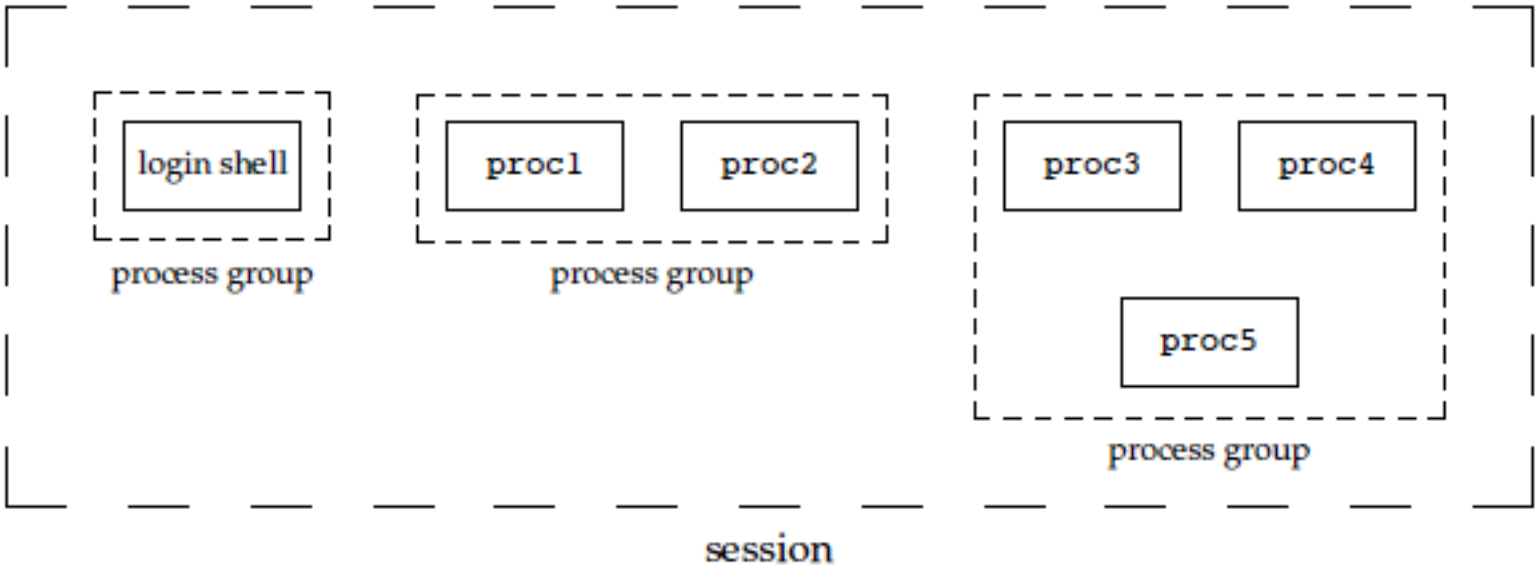


Figure 9.6 Arrangement of processes into process groups and sessions

- A session is a collection of one or more process groups.

Process Sessions

```
#include <unistd.h>

pid_t setsid(void);
            Returns: process group ID if OK, -1 otherwise
```

- If the calling process is not a process group leader, this function creates a new session. Three things happen:
  - the process becomes the session leader of this new session
  - the process becomes the process group leader of a new process group
  - the process has no controlling terminal

Controlling Terminal

- Sessions and process groups have a few other characteristics.
  - A session can have a single controlling terminal.
  - The session leader that establishes *the connection to the controlling terminal* is called the controlling process .
  - The process groups within a session can be divided into a single foreground process group and one or more background process groups.
  - Whenever we press the terminal's interrupt key (often DELETE or Control-C), the interrupt signal is sent to all processes in the foreground process group.

Controlling Terminal

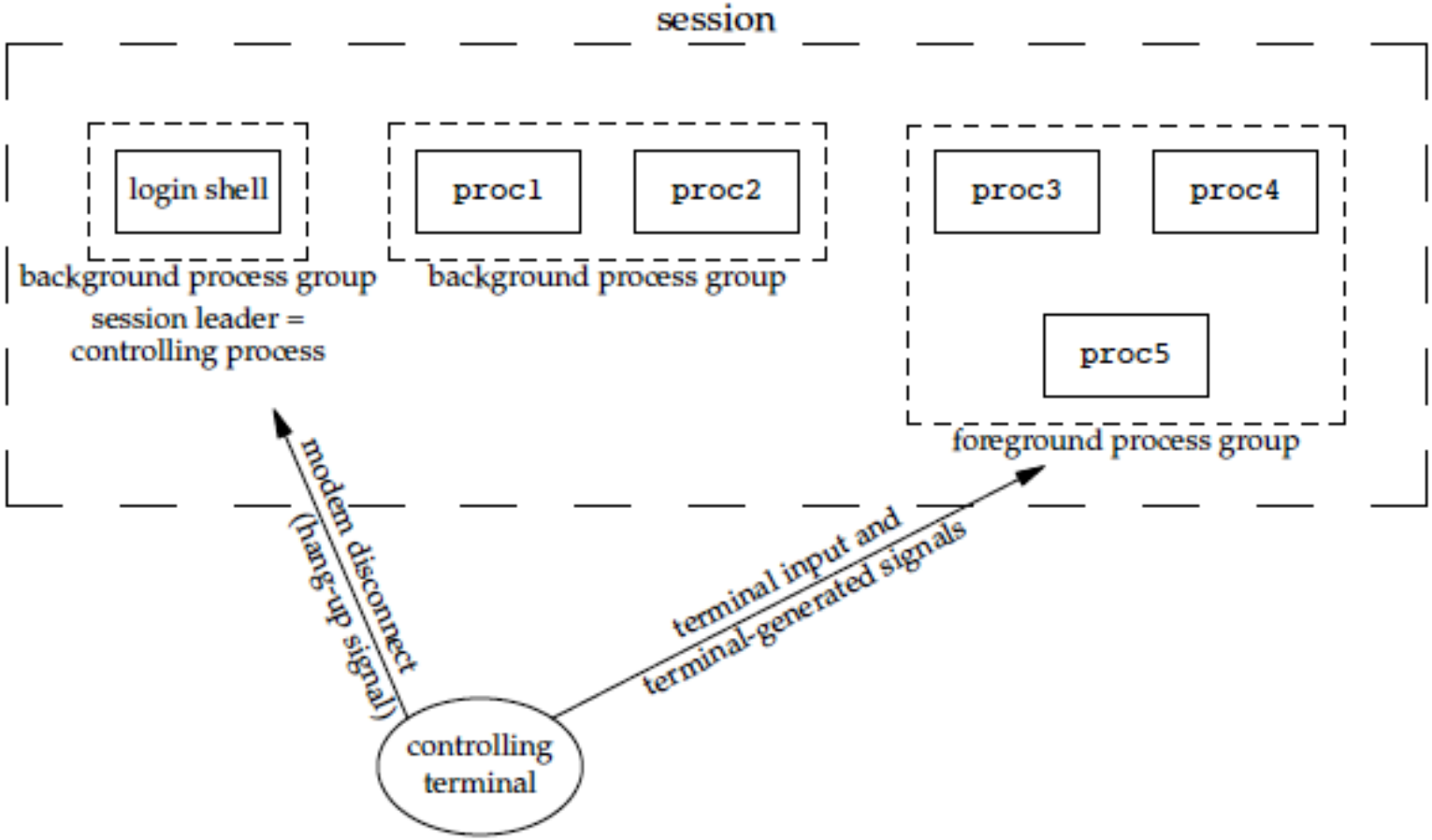
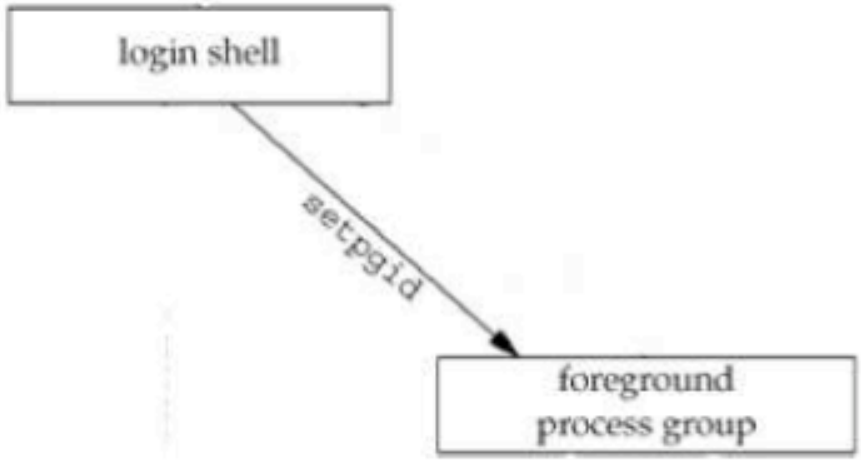


Figure 9.7 Process groups and sessions showing controlling terminal

Job Control

- Job control is a feature that was added to BSD around 1980.
- This feature allows us to start multiple jobs (groups of processes) from a single terminal.

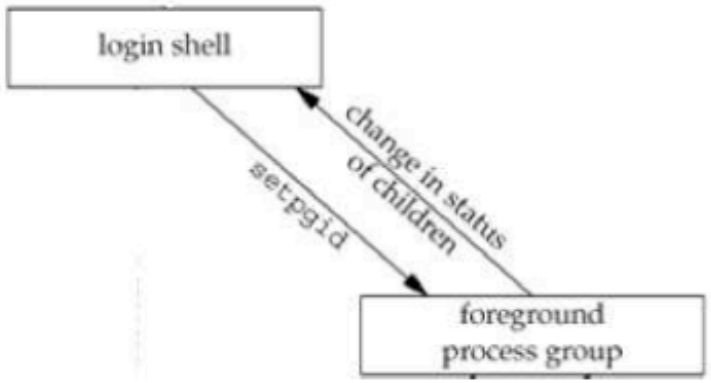
Job Control




```
$ ps -o pid,ppid,pgid,sess,comm
  PID  PPID  PGRP  SESS COMMAND
24251 24250 24251 24251 ksh
24620 24251 24620 24251 ps
$
```

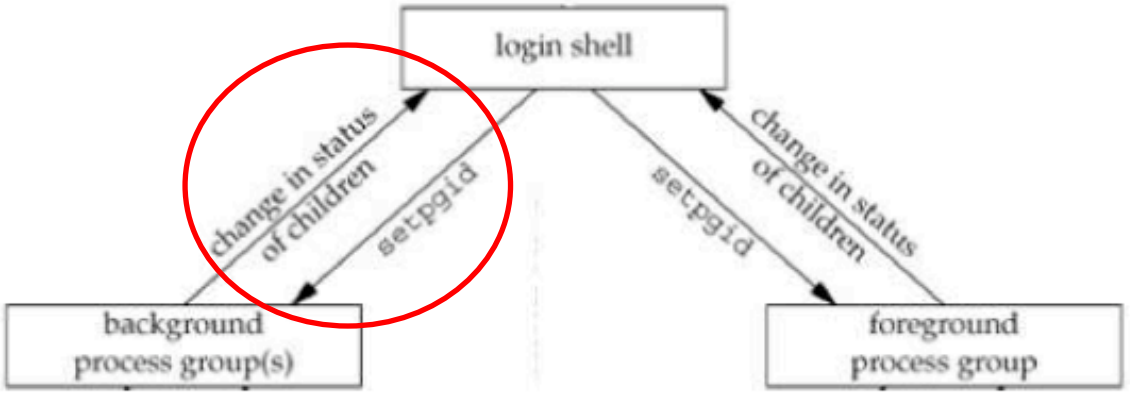
Job Control

- Check the return value.




```
$ ps -o pid,ppid,pgid,sess,comm
  PID  PPID  PGRP  SESS COMMAND
24251 24250 24251 24251 ksh
24620 24251 24620 24251 ps
$ echo $?
0
$
```

Job Control




```
$ /bin/sleep 30 &
[1] 24748
$ ps -o pid,ppid,pgid,sess,comm
  PID  PPID  PGRP  SESS COMMAND
24251 24250 24251 24251 ksh
24748 24251 24748 24251 sleep
24750 24251 24750 24251 ps
$
[1] +  Done     /bin/sleep 30 &
$
```

Job Control

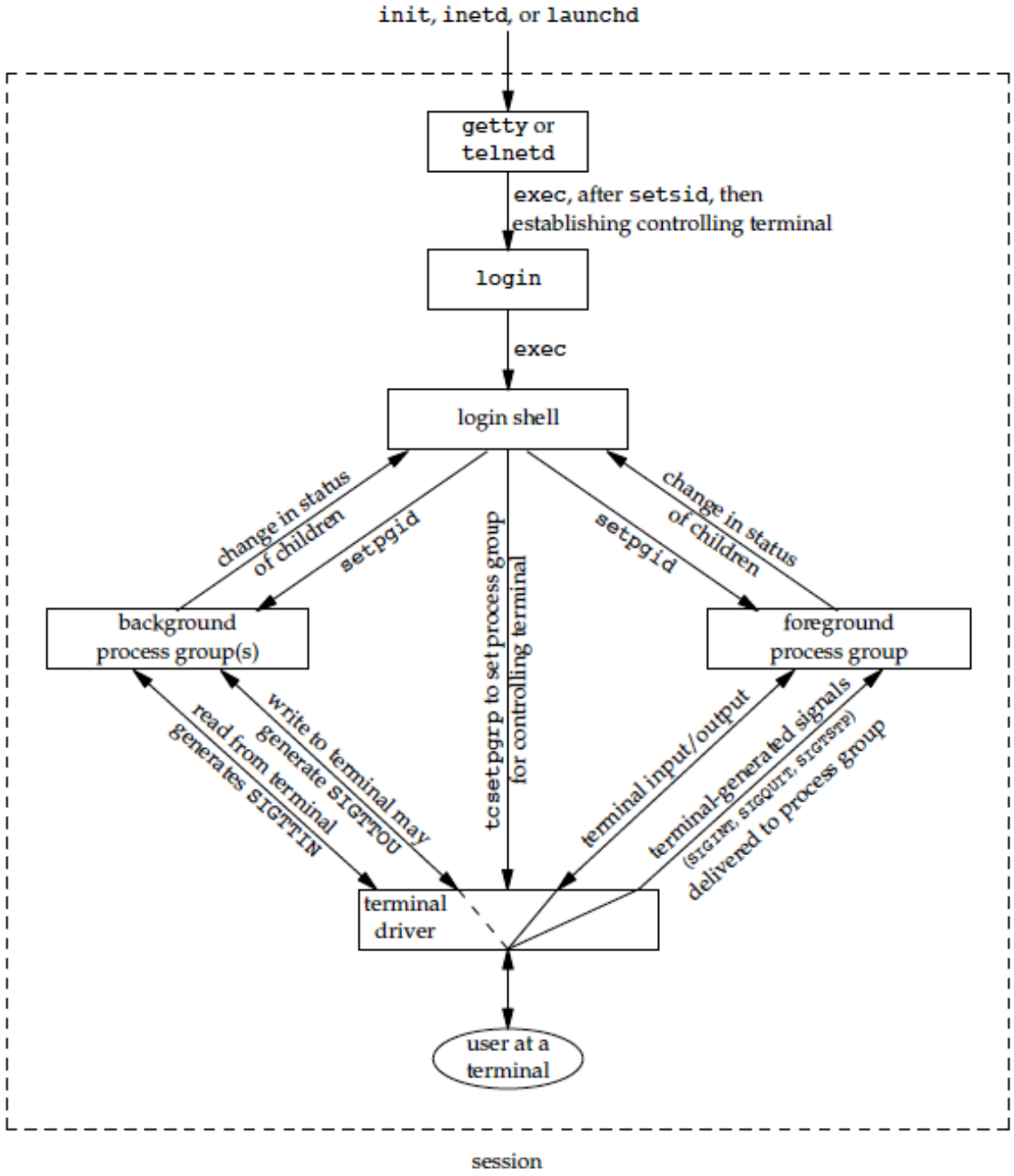
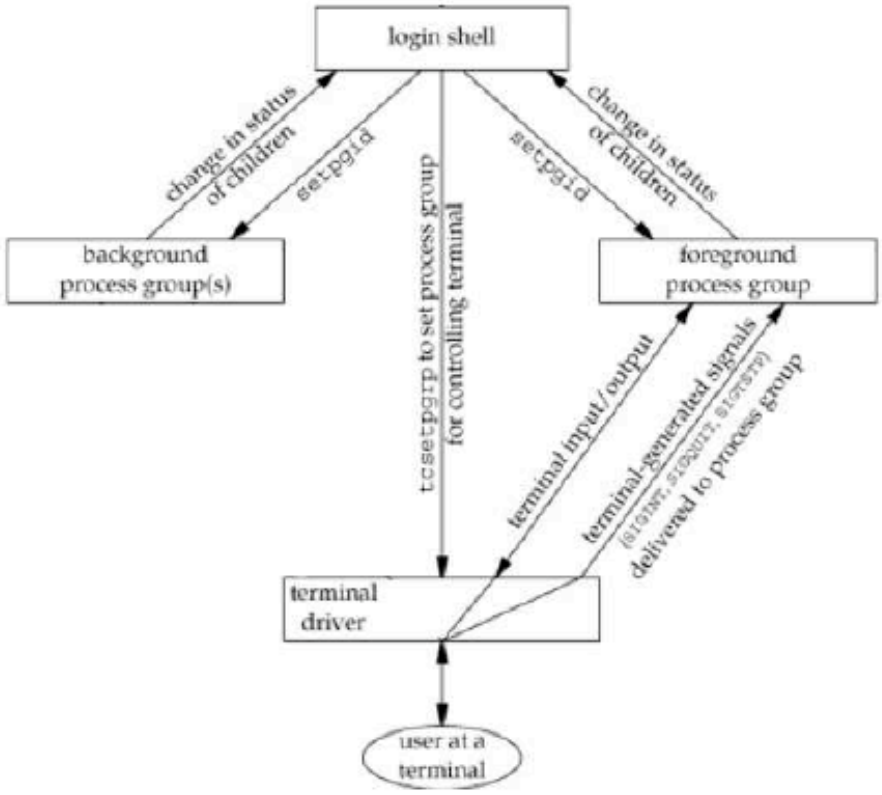


Figure 9.9 Summary of job control features with foreground and background jobs, and terminal driver

Job Control

```
$ cat >file
Input from terminal,
Output to terminal.
^D
$ cat file
Input from terminal,
Output to terminal.
$ cat >/dev/null
Input from terminal,
Output to /dev/null.
Waiting forever...
Or until we send an interrupt signal.
^C
$
```



Shell Execution of Programs

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

This pipeline generates the following output:

```
  PID  PPID  PGID   SID COMMAND
  949   947   949   949 sh
 1988   949   949   949 cat2
 1989  1988   949   949 ps
 1990  1988   949   949 cat1
```

```
[esjung@hpclab ~]$ ps -o pid,ppid,pgid,sess,comm | cat | cat
   PID   PPID   PGID   SESS COMMAND
169343 169342 169343 169343 bash
169981 169343 169981 169343 ps
169982 169343 169981 169343 cat
169983 169343 169981 169343 cat
[esjung@hpclab ~]$
```

Shell Execution of Programs

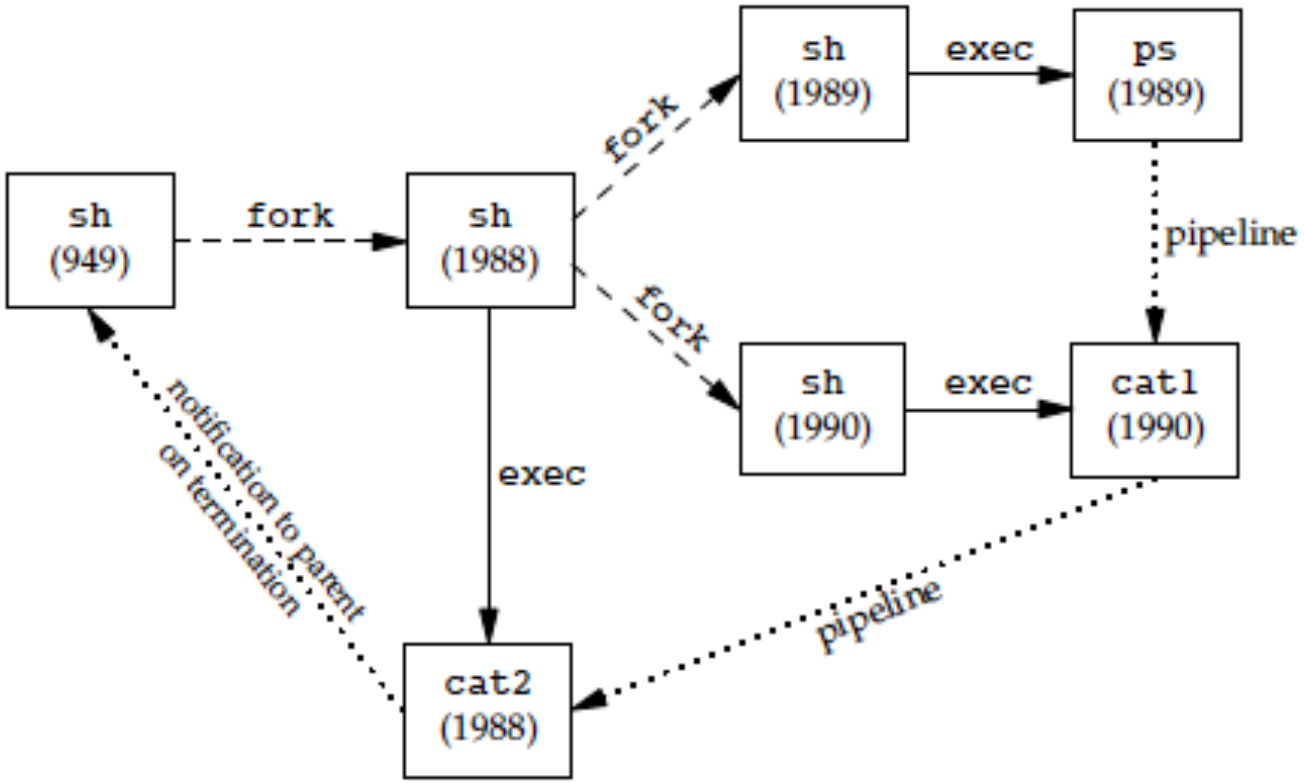


Figure 9.10 Processes in the pipeline `ps | cat1 | cat2` when invoked by Bourne shell