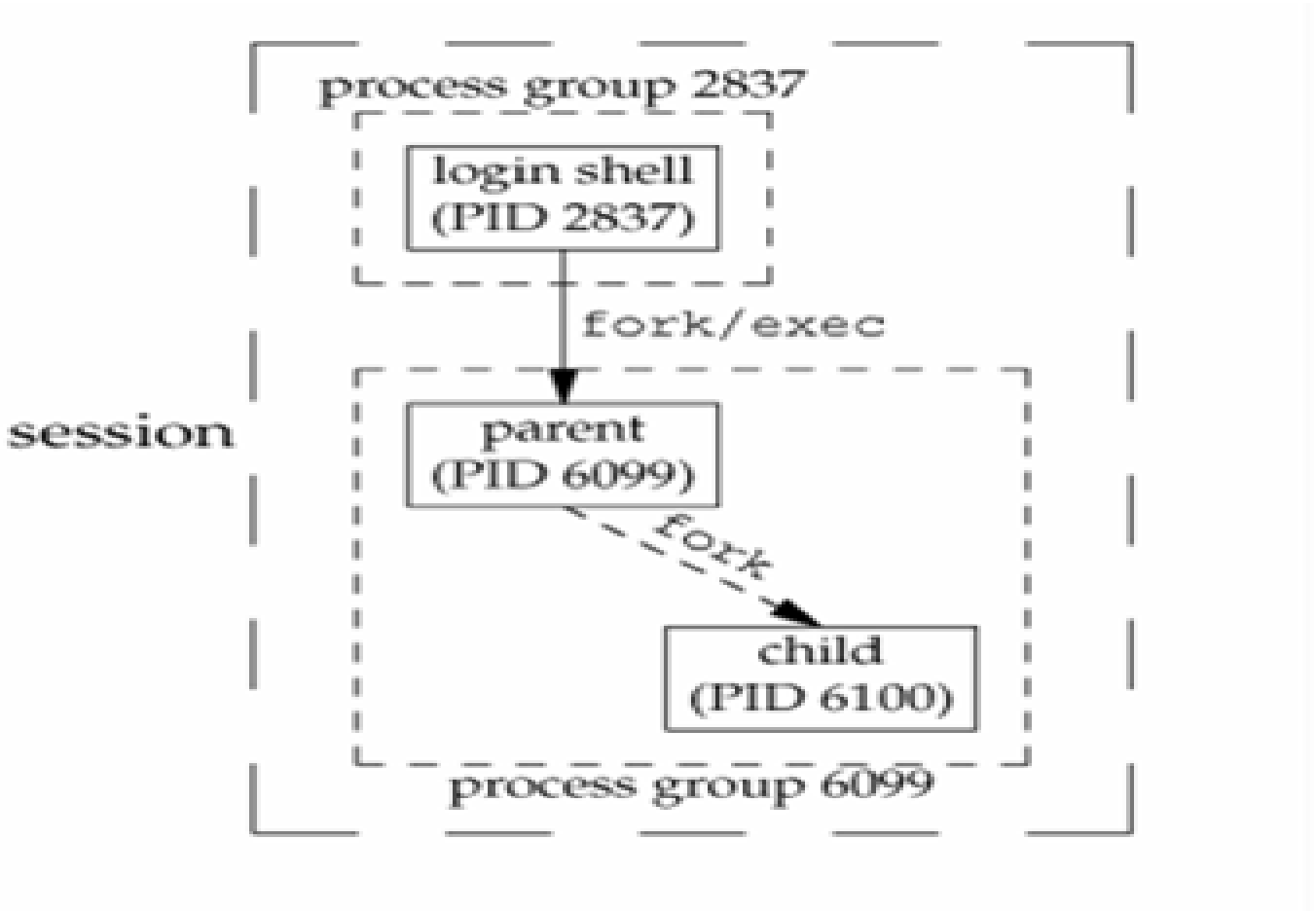
ORPHANED PROCESS GROUPS

- We know that a process whose parent terminates is called an orphan and is inherited by the init process.
- Sometimes the entire process groups can be orphaned.
- This is a job-control shell. The shell places the foreground process in its own process group(512 in the example) and the shell stays in its own process group(442). The child inherits the process group of its parent(512). After the fork,
- The parent sleeps for 5 seconds. This is the (imperfect) way of letting the child execute before the parent terminates
- The child establishes a signal handler for the hang-up signal (SIGHUP). This is so we can see if SIGHUP is sent to the child.
- The child itself the stop signal(SIGTSTP) with the kill function.
- When the parent terminates, the child is orphaned, so the child's parent process ID becomes 1, the init process ID.
- At this point the child is now a member of an *orphaned process group*.

- Since the process group is orphaned when the parent terminates, it is required that every process in the newly orphaned process group that is stopped be sent the hang-up signal (SIGHUP) followed by the continue signal.
- This causes the child to be continued, after processing the hang-up signal. The default action for the hang-up signal is to terminate the process, which is why we have to provide a signal handler to catch the signal



Creating an orphaned process group

```
#include        <sys/types.h>
#include        <errno.h>
#include        <fcntl.h>
#include        <signal.h>
#include        "ourhdr.h"
static void     sig_hup(int);
static void     pr_ids(char *);
int main(void)
{
```

```
        char    c;
        pid_t   pid;
   pr_ids("parent");
        if ( (pid = fork()) < 0)
                err_sys("fork error");
        else if (pid > 0)
    {                                   /* parent */
         sleep(5);
     exit(0);
     }
else {                          /* child */
                pr_ids("child");
                signal(SIGHUP, sig_hup);
       /* establish signal handler */
   kill (getpid(), SIGTSTP);                    pr_ids("child");
      /* this prints only if we're continued */
                if (read(0, &c, 1) != 1)
       printf ("read error from control
            terminal,errno = %d\n", errno);             exit(0);
        }
}
static void    sig_hup (int signo)
   {
        printf("SIGHUP received, pid = %d\n",
                                        getpid());
        return;
   }
  static void  pr_ids (char *name)
   {
        printf("%s: pid = %d, ppid = %d, pgrp =
```

```
    d\n", name, getpid(), getppid(), getpgrp());
            fflush(stdout);
    }
/* OUTPUT
$ a.out
Parent: pid = 512, ppid=442, pgrp = 512
Child: parent = 513, ppid = 512, pgrp = 512
$ SIGHUP received, pid = 513
Child: pid = 513 , ppid = 1, pgrp = 512
Read error from control terminal, errno = 5
*/
```

- The parent process ID of the child has become 1.
- After calling pr_ids in the child, the program tries to read from standard input. When the background process group tries to read from its controlling terminal, SIGTTIN is generated from the background process group.
- The child becomes the background process group when the parent terminates, since the parent was executed as a foreground job by the shell