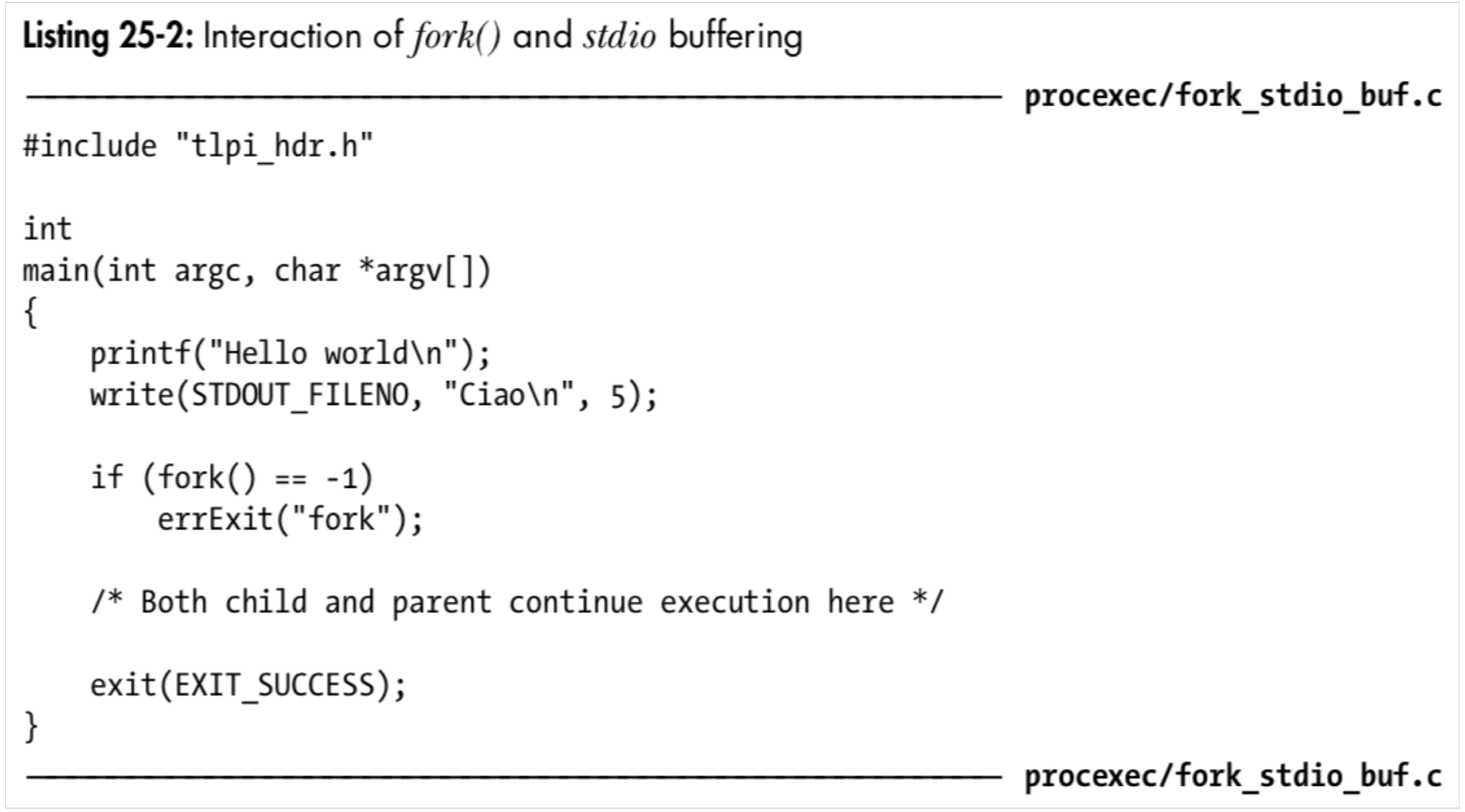
Processes iii

Fork with file buffers

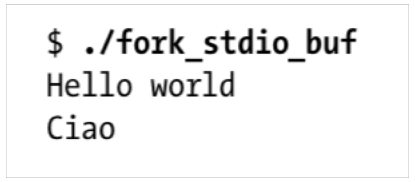


Listing 25-2: Interaction of *fork()* and *stdio* buffering

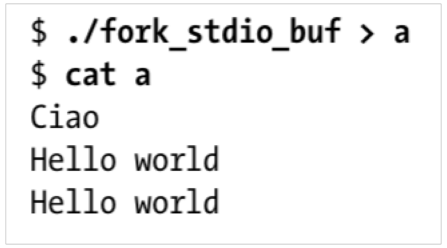
procexec/fork_stdio_buf.c

```c
#include "tlpi_hdr.h"

int
main(int argc, char *argv[])
{
    printf("Hello world\n");
    write(STDOUT_FILENO, "Ciao\n", 5);

    if (fork() == -1)
        errExit("fork");

    /* Both child and parent continue execution here */

    exit(EXIT_SUCCESS);
}
```

procexec/fork_stdio_buf.c



```
$ ./fork_stdio_buf
Hello world
Ciao
```



```
$ ./fork_stdio_buf > a
$ cat a
Ciao
Hello world
Hello world
```

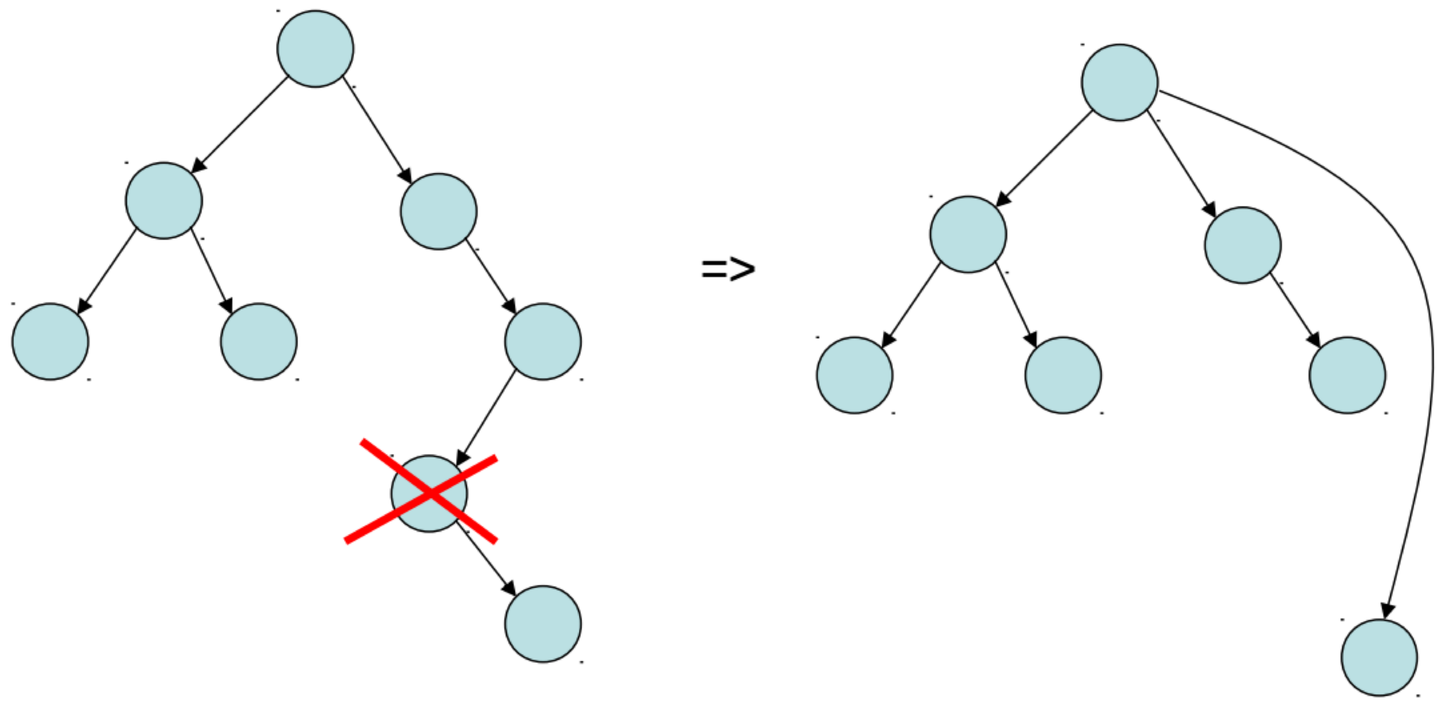
Process Termination

- Voluntary : exit(status)
  - OS passes exit status to parent via wait(&status)
  - OS frees process resources

- Involuntary : kill(pid, signal)
  - Signal can be sent by another process or by OS
  - pid is for the process to be killed
  - signal a signal that the process needs to be killed
    - Examples : SIGTERM, SIGQUIT (ctrl+\), SIGINT (ctrl+c), SIGHUP

Orphans

- When a parent process terminates before its child
- Adopted by first process (/sbin/init)



Zombies

- When a process terminates it becomes a zombie (or defunct process)
  - PCB in OS still exists even though program no longer executing
  - Why? So that the parent process can read the child's exit status (through wait system call)
- When parent reads status,
  - zombie entries removed from OS… process reaped!
- Suppose parent does'nt read status
  - Zombie will continue to exist infinitely … a resource leak
  - These are typically found by a reaper process