



# Exceptional Control Flow: Signals and Nonlocal Jumps

These slides adapted from materials provided by the textbook authors.

# Signals and Nonlocal Jumps

- Review
- Shells
- **Signals**
  - Sending Signals
  - Receiving Signals
- Nonlocal jumps

# Signals

- A ***signal*** is a small message that notifies a process that an event of some type has occurred in the system
  - Akin to exceptions and interrupts
  - Sent from the kernel (sometimes at the request of another process) to a process
  - Signal type is identified by small integer ID's (1-30)
  - Only information in a signal is its ID and the fact that it arrived

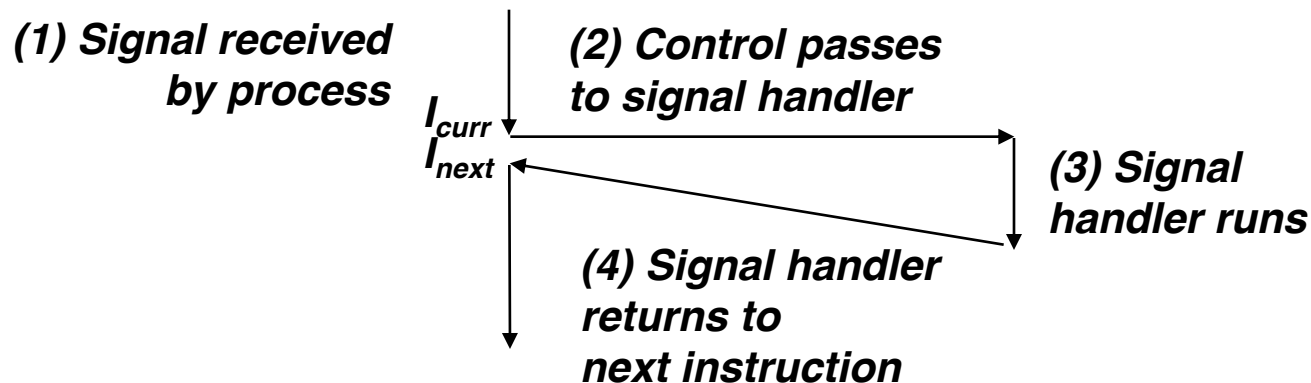
<i>ID</i>	<i>Name</i>	<i>Default Action</i>	<i>Corresponding Event</i>
2	SIGINT	Terminate	User typed ctrl-c
9	SIGKILL	Terminate	Kill program (cannot override or ignore)
11	SIGSEGV	Terminate	Segmentation violation
14	SIGALRM	Terminate	Timer signal
17	SIGCHLD	Ignore	Child stopped or terminated

# Signal Concepts: Sending a Signal

- Kernel *sends* (delivers) a signal to a *destination process* by updating some state in the context of the destination process
- Kernel sends a signal for one of the following reasons:
  - Kernel has detected a system event such as divide-by-zero (SIGFPE) or the termination of a child process (SIGCHLD)
  - Another process has invoked the `kill` system call to explicitly request the kernel to send a signal to the destination process

# Signal Concepts: Receiving a Signal

- A destination process *receives* a signal when it is forced by the kernel to react in some way to the delivery of the signal
- Some possible ways to react:
  - *Ignore* the signal (do nothing)
  - *Terminate* the process (with optional core dump)
  - *Catch* the signal by executing a user-level function called *signal handler*
    - Akin to a hardware exception handler being called in response to an asynchronous interrupt:



# Signal Concepts: Pending and Blocked Signals

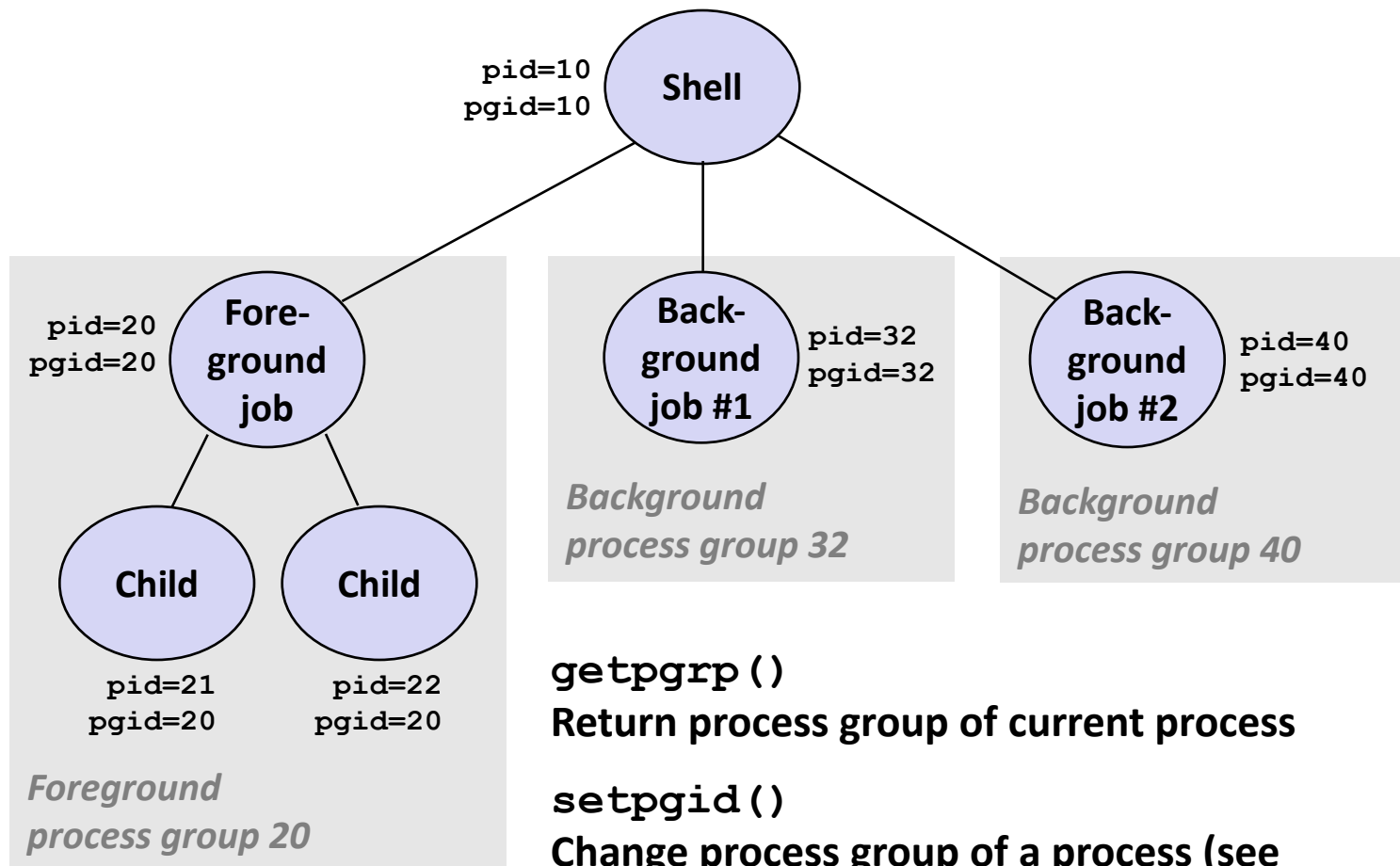
- A signal is *pending* if sent but not yet received
  - There can be at most one pending signal of any particular type
  - Important: Signals are not queued
    - If a process has a pending signal of type k, then subsequent signals of type k that are sent to that process are discarded
- A process can *block* the receipt of certain signals
  - Blocked signals can be delivered, but will not be received until the signal is unblocked
- A pending signal is received at most once

# Signal Concepts: Pending/Blocked Bits

- Kernel maintains **pending** and **blocked** bit vectors in the context of each process
  - **pending**: represents the set of pending signals
    - Kernel sets bit *k* in **pending** when a signal of type *k* is delivered
    - Kernel clears bit *k* in **pending** when a signal of type *k* is received
  - **blocked**: represents the set of blocked signals
    - Can be set and cleared by using the **sigprocmask** function
    - Also referred to as the *signal mask*.

# Sending Signals: Process Groups

- Every process belongs to exactly one process group





# Sending Signals with `/bin/kill` Program

- `/bin/kill` program  
sends arbitrary signal to a  
process or process group

- Examples

- `/bin/kill -9 24818`  
Send SIGKILL to process 24818

- `/bin/kill -9 -24817`  
Send SIGKILL to every process  
in process group 24817

```
linux> ./forks 16
Child1: pid=24818 pgrp=24817
Child2: pid=24819 pgrp=24817
```

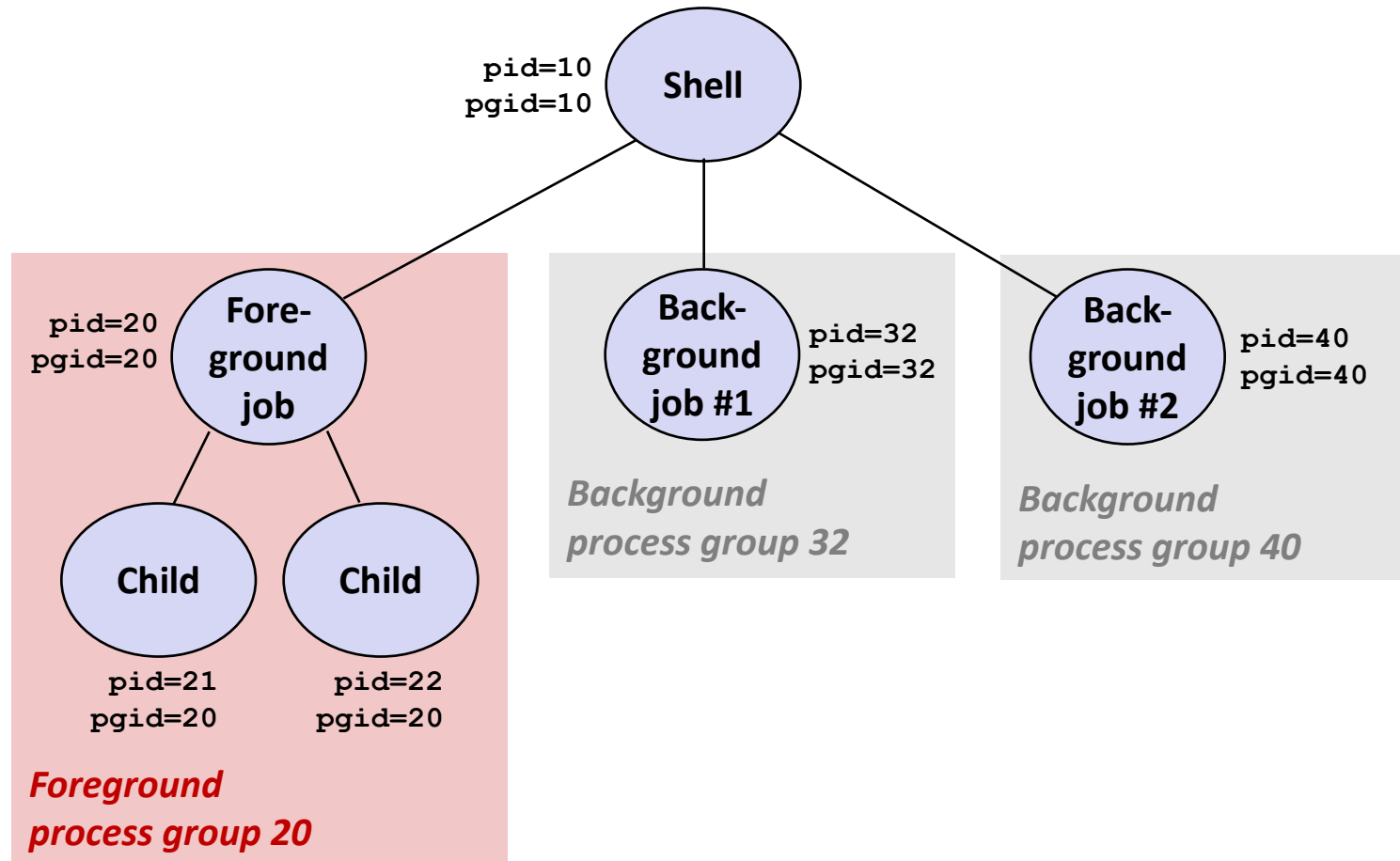
```
linux> ps
  PID TTY          TIME CMD
24788 pts/2        00:00:00 tcsh
24818 pts/2        00:00:02 forks
24819 pts/2        00:00:02 forks
24820 pts/2        00:00:00 ps
```

```
linux> /bin/kill -9 -24817
```

```
linux> ps
  PID TTY          TIME CMD
24788 pts/2        00:00:00 tcsh
24823 pts/2        00:00:00 ps
linux>
```

# Sending Signals from the Keyboard

- Typing ctrl-c (ctrl-z) causes the kernel to send a SIGINT (SIGTSTP) to every job in the foreground process group.
  - SIGINT – default action is to terminate each process
  - SIGTSTP – default action is to stop (suspend) each process



# Example of `ctrl-c` and `ctrl-z`

```
bluefish> ./forks 17
Child: pid=28108 pgrp=28107
Parent: pid=28107 pgrp=28107
<types ctrl-z>
Suspended
bluefish> ps w
  PID TTY          STAT       TIME COMMAND
 27699 pts/8        Ss          0:00 -tcsh
 28107 pts/8        T           0:01 ./forks 17
 28108 pts/8        T           0:01 ./forks 17
 28109 pts/8        R+          0:00 ps w
bluefish> fg
./forks 17
<types ctrl-c>
bluefish> ps w
  PID TTY          STAT       TIME COMMAND
 27699 pts/8        Ss          0:00 -tcsh
 28110 pts/8        R+          0:00 ps w
```

STAT (process state) Legend:

**First letter:**

S: sleeping

T: stopped

R: running

**Second letter:**

s: session leader

+: foreground proc group

See “man ps” for more details

# Sending Signals with `kill` Function

```
void fork12()
{
    pid_t pid[N];
    int i;
    int child_status;

    for (i = 0; i < N; i++)
        if ((pid[i] = fork()) == 0) {
            /* Child: Infinite Loop */
            while(1)
                ;
        }

    for (i = 0; i < N; i++) {
        printf("Killing process %d\n", pid[i]);
        kill(pid[i], SIGINT);
    }

    for (i = 0; i < N; i++) {
        pid_t wpid = wait(&child_status);
        if (WIFEXITED(child_status))
            printf("Child %d terminated with exit status %d\n",
                wpid, WEXITSTATUS(child_status));
        else
            printf("Child %d terminated abnormally\n", wpid);
    }
}
```

*forks.c*