Signals

Prof. Joonwon Lee(joonwon@skku.edu)
TA – Taekyun Roh(tkroh0198@skku.edu)
Sewan Ha(hsewan2495@gmail.com)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Signal

- **Definition**
  - A signal is a small message that notifies a process that an event of some type has occurred in the system.
    - Kernel abstraction for exceptions and interrupts.
    - Sent from kernel (sometimes at the request of another process) to a process.
    - Different signals are identified by small integer ID's.
    - The only information in a signal is its ID and the fact that it arrived.

Signal Concepts (1)

- **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:
    - Generated internally:
      - » Divide-by-zero (**SIGFPE**)
      - » Termination of a child process (**SIGCHLD**), CTRL-C (**SIGINT**)
    - Generated externally:
      - » **kill** system call by another process to request signal to the destination process.

Signal Concepts (2)

- **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.
  - Three possible ways to react:
    - Explicitly ignore the signal
    - Execute the default action
    - Catch the signal by invoking signal-handler function
      - » Akin to a hardware exception handler being called in response to an asynchronous interrupt.

Signal Concepts (3)

- **Default actions**
  - Abort
    - The process is destroyed
  - Dump
    - The process is destroyed & core dump
  - Ignore
    - The signal is ignored
  - Stop
    - The process is stopped
  - Continue
    - If the process is stopped, it is put into running state

Signal Concepts Example

- Or you can see it from 'man 7 signal'

```
Standard signals
    Linux supports the standard signals listed below.  Several signal numbers are architecture-dependent, as in
    other architectures, and the last one for mips.  (Values for parisc are not shown; see the Linux kernel sou

    First the signals described in the original POSIX.1-1990 standard.

    Signal     Value     Action   Comment
    ──────────────────────────────────────────────────────────────────────
    SIGHUP        1       Term    Hangup detected on controlling terminal
                                  or death of controlling process
    SIGINT        2       Term    Interrupt from keyboard
    SIGQUIT       3       Core    Quit from keyboard
    SIGILL        4       Core    Illegal Instruction
    SIGABRT       6       Core    Abort signal from abort(3)
    SIGFPE        8       Core    Floating point exception
    SIGKILL       9       Term    Kill signal
    SIGSEGV      11       Core    Invalid memory reference
    SIGPIPE      13       Term    Broken pipe: write to pipe with no
                                  readers
    SIGALRM      14       Term    Timer signal from alarm(2)
    SIGTERM      15       Term    Termination signal
    SIGUSR1   30,10,16    Term    User-defined signal 1
    SIGUSR2   31,12,17    Term    User-defined signal 2
    SIGCHLD   20,17,18    Ign     Child stopped or terminated
    SIGCONT   19,18,25    Cont    Continue if stopped
    SIGSTOP   17,19,23    Stop    Stop process
    SIGTSTP   18,20,24    Stop    Stop typed at terminal
    SIGTTIN   21,21,26    Stop    Terminal input for background process
    SIGTTOU   22,22,27    Stop    Terminal output for background process

    The signals SIGKILL and SIGSTOP cannot be caught, blocked, or ignored.

    Next the signals not in the POSIX.1-1990 standard but described in SUSv2 and POSIX.1-2001.
```

Signal Concepts (4)

- **Signal semantics**
  - A signal is pending if it has been sent but not yet received.
    - There can be at most one pending signal of any particular type.
    - Signals are not queued!
  - A process can block the receipt of certain signals.
    - Blocked signals can be delivered, but will not be received until the signal is unblocked.
    - There is one signal that can not be blocked by the process. (**SIGKILL**, **SIGSTOP**)
  - A pending signal is received at most once.
    - Kernel uses a bit vector for indicating pending signals.

Signal Concepts (5)

- **Implementation**
  - 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** whenever a signal of type k is delivered.
      - » Kernel clears bit k in **pending** whenever a signal of type k is received.
    - **blocked** – represents the set of blocked signals
      - » Can be set and cleared by the application using the **sigprocmask** function.

Signal Concepts (6)

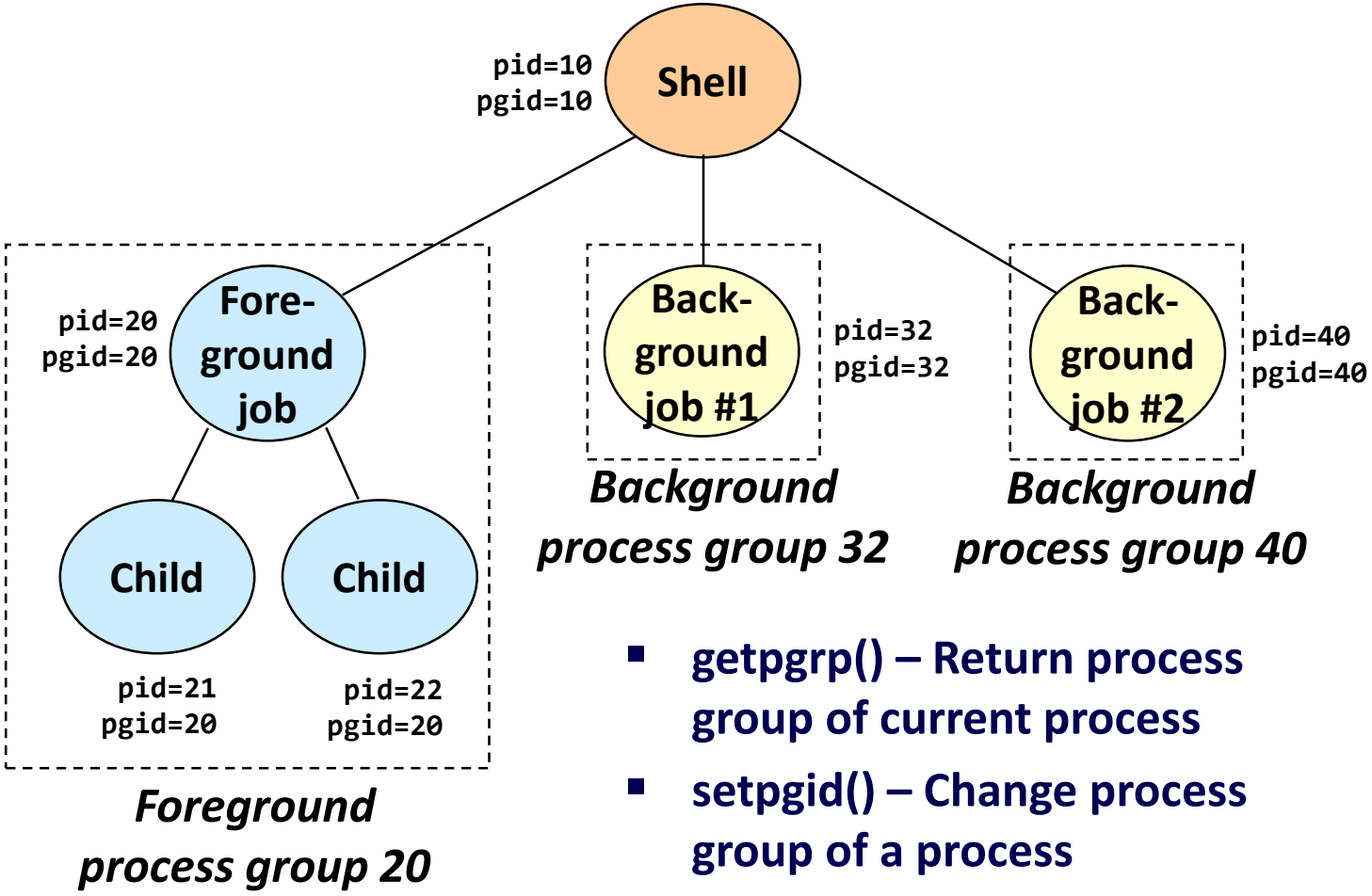
- Signal Block Example

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
```

```
int main(void)
{
    sigset_t sigset;
    int ndx;
    sigemptyset(&sigset);
    sigaddset(&sigset,SIGINT);

    //Block Mode
    sigprocmask(SIG_BLOCK, &sigset, NULL);

    //Start counting
    for(ndx = 3; 0 < ndx ; ndx--){
        printf("Count Down: %d\n",ndx);
        sleep(1);
    }

    //Unblock Mode
    printf("Unblock");
    sigprocmask(SIG_UNBLOCK,&sigset,NULL);
    printf("If you press CTRL-C, This sentence is not printed\n");

    while(1);
    return 0;
}
```

Process Groups

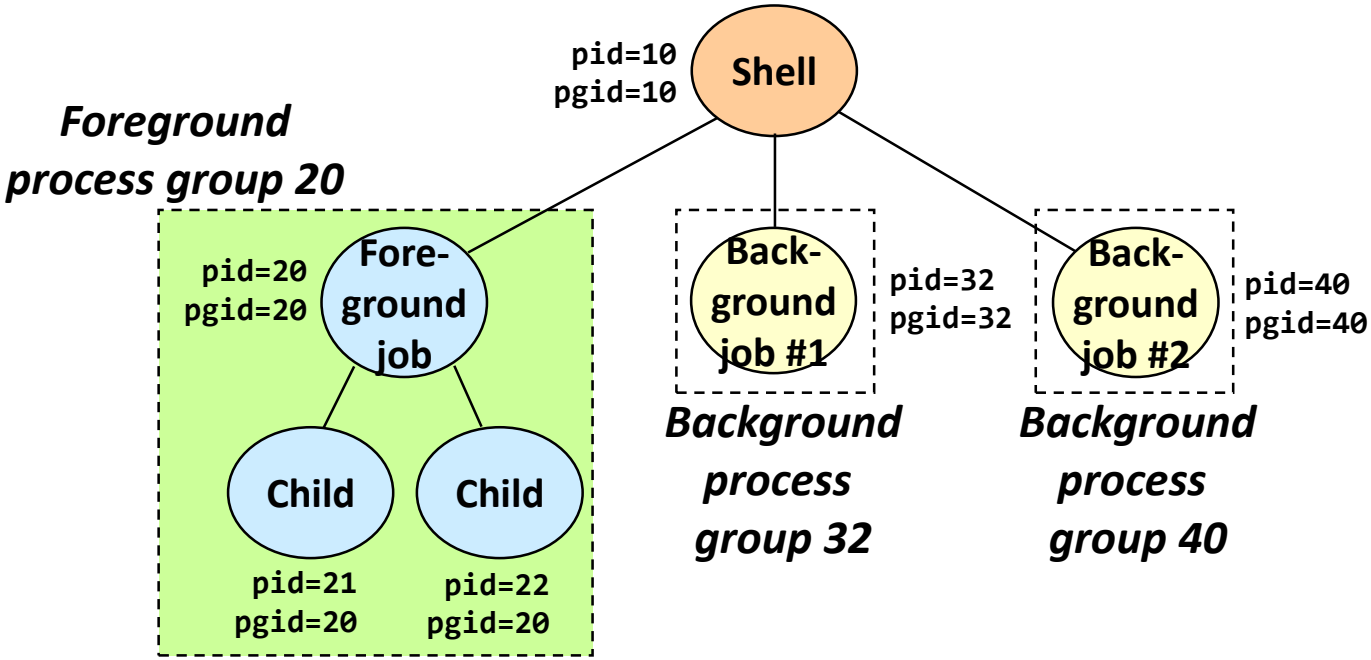
- Every process belongs to exactly one process group.



- getpgrp() – Return process group of current process
- setpgid() – Change process group of a process

Sending Signals (1)

- **Sending signals from the keyboard**
  - Typing **ctrl-c** (**ctrl-z**) sends 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.



Sending Signals (2)

- **int kill(pid_t pid, int sig)**
  - Can be used to send any signal to any process group or process.
    - **pid** > 0, signal **sig** is sent to **pid**.
    - **pid** == 0, **sig** is sent to every process in the process group of the current process.
    - **pid** == -1, **sig** is sent to every process except for process 1.
    - **pid** < -1, **sig** is sent to every process in the process group -**pid**.
    - **sig** == 0, no signal is sent, but error checking is performed.
- **/bin/kill program sends arbitrary signal to a process or process group.**

```
$ kill 10231        // SIGTERM : default signal
$ kill -9 10231     // SIGKILL
```

Exercise #1

- Terminate child processes using 'kill'

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>

#define N (10)
```

```
int main(void)
{
    pid_t pid[N];
    int i, child_status;
    for (i = 0; i < N; i++) {
        if ((pid[i] = fork()) == 0) {
            while(1); /* Child infinite loop */
        }
    }

    /* Parent terminates the child processes (Your C code) */

    /* Parent reaps terminated children */
    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);
    }

    return 0;
}
```

Installing Signal Handlers

- **sighandler_t signal (int sig, sighandler_t handler)**
  - **typedef void (*sighandler_t)(int);**
  - The signal function modifies the default action associated with the receipt of signal **`sig`**.
- **Different values for handler:**
  - SIG_IGN: ignore signals of type sig.
  - SIG_DFL: revert to the default action.
  - Otherwise, handler is the address of a signal handler.
    - Called when process receives signal of type **sig**.
    - Referred to as "installing" the signal handler.
    - Executing handler is called "catching" or "handling" the signal.
    - When the handler executes its return statement, control passes back to instruction in the control flow of the process that was interrupted by receipt of the signal.

Handling Signals (1)

- **Things to remember**
  - Pending signals are not queued.
    - For each signal type, just have single bit indicating whether or not signal is pending.
    - Even if multiple processes have sent this signal.
  - A newly arrived signal is blocked while the handler of the signal is running.
  - Sometimes system calls such as **read()** are not restarted automatically after they are interrupted by the delivery of a signal.
    - They return prematurely to the calling application with an error condition. **(errno == EINTR)**

Handling Signals (2)

- What is the problem of the following code?

```
#define N (10)

pid_t pid[N];
int ccount = 0;

void handler (int sig) {
   pid_t id = wait(NULL);
   ccount--;
   printf ("Received signal %d from pid %d\n", sig, id);
}

int main(void) {
   int i;
   ccount = N;
   signal (SIGCHLD, handler);
   for (i = 0; i < N; i++) {
      if ((pid[i] = fork()) == 0) {
         exit(0); /* child */
      }
   }

   while (ccount > 0)
      sleep (5);

    return 0;
}
```

Exercise #2

- Deal with non-queueing signals

```
#define N (10)

pid_t pid[N];
int ccount = 0;

void handler (int sig) {
   pid_t id = wait(NULL);
   ccount--;
   printf ("Received signal %d from pid %d\n", sig, id);
}

int main(void) {
   int i;
   ccount = N;
   signal (SIGCHLD, handler);
   for (i = 0; i < N; i++) {
      if ((pid[i] = fork()) == 0) {
         exit(0); /* child */
      }
   }

   while (ccount > 0)
      sleep (5);

    return 0;
}
```

Tip: pid_t waitpid(pid[i], NULL, WNOHANG)
pid == 0 if child is still running

Exercise #3

- **React to externally generated events**

- **Make zombie process**
  - When the process get ctrl+c signal from keyboard, it just prints "beep" to the monitor 5 times with 1-second interval (use sleep)
  - Print "I'm Alive!" to the monitor after 5-times beep