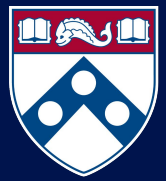
CIS 5480
Recitation 2

Thursday, February 13 2025



Penn Engineering
University of Pennsylvania

Agenda

- Job Control
- Signal Handling

Job Control

We group processes together in a process group to send on signal across all processes.

Processes in a pipeline form a process group.

fork() creates a process group to which the parent and child belong to.

Job Control

Processes have a group id (PGID) and is usually the PID of the process that creates the group.

That PID is reserved, even after that process is terminated, until the whole group is terminated.

Job Control

Bash Demo

Job Control

int setpgid(pid_t pid, pid_t pgid)

setpgid(2) sets the pgid of process pid to pgid.

What happens if pid is zero?

What happens if pgid is zero?

Job Control

int setpgid(pid_t pid, pid_t pgid)

setpgid(2) sets the PGID of process pid to pgid.

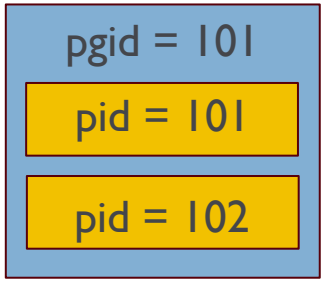
What happens if pid is zero?

pid is set to the PID of the calling process (i.e the process changes its PGID to pgid).

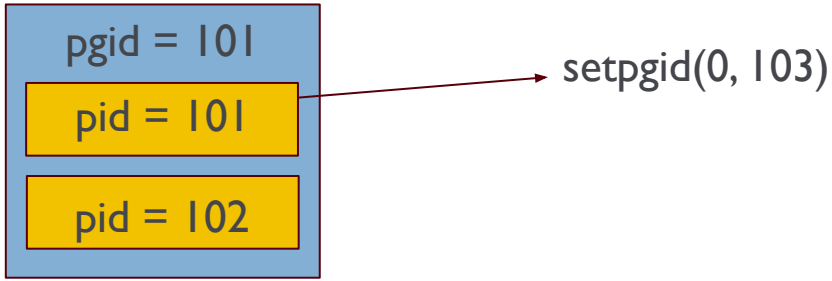
What happens if pgid is zero?

pgid is set to the PGID of the process pid.

Job Control

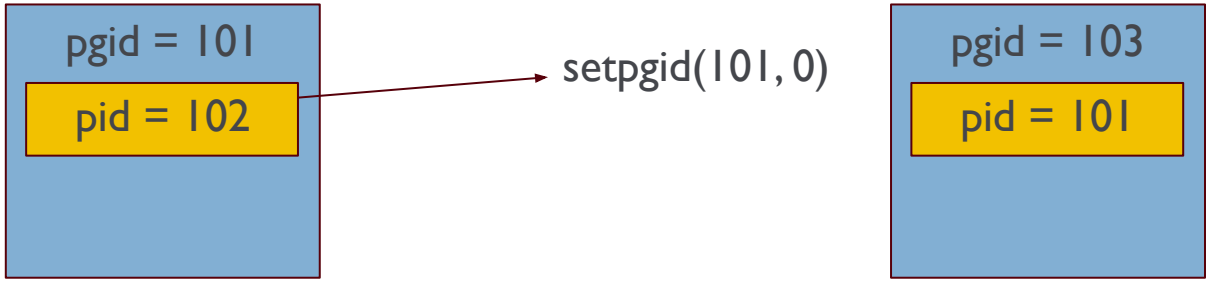


Job Control

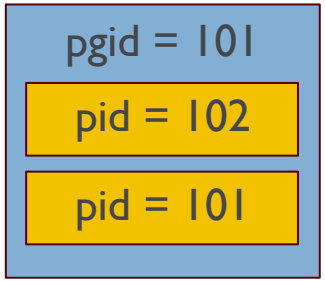


Job Control

Job Control



Job Control



Job Control

int setpgid(pid_t pid, pid_t pgid)

setpgid(2) sets the PGID of process pid to pgid.

What does setpgid(0, 0) do?

Job Control

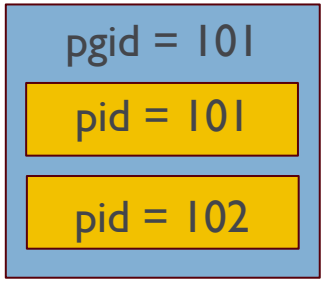
int setpgid(pid_t pid, pid_t pgid)

setpgid(2) sets the pgid of process pid to pgid.

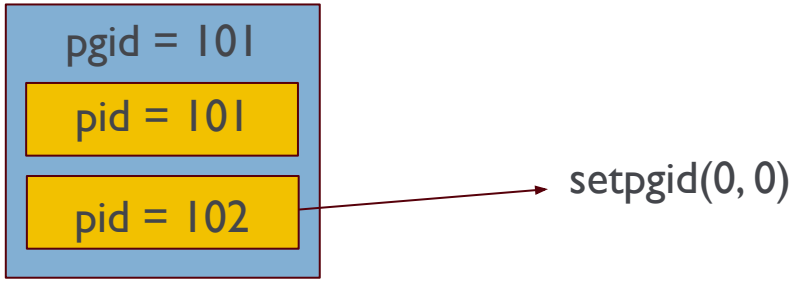
What does setpgid(0, 0) do?

Sets the PGID of the calling process to its PID.

Job Control



Job Control



Job Control

Job Control

pid_t getpgid(pid_t pid)

getpgid(3p) returns the PGID of process pid. It returns -1 if it fails.

What happens if pid is zero?

Job Control

pid_t getpgid(pid_t pid)

getpgid(3p) returns the pgid of process pid. It returns -1 if it fails.

What happens if pid is zero?

pid is set to the PID of the calling process (i.e. returns the PGID of the calling process).

Job Control

With all functions that use pid, you can use pgid.

However, you must use the negative of the pgid!

Bash Demo

Signal Handling

- Signals are software-generated interrupts that notify processes of events.
- Similar to hardware interrupts but handled at the process level.
- Used for inter-process communication and exception handling.

Signal Handling

- **Common Signals:**
  - `SIGINT` (Interrupt from keyboard (CTRL-C) , default: terminate)
  - `SIGKILL` (Forcefully terminate process, cannot be ignored)
  - `SIGTERM` (Terminate process gracefully)
  - `SIGCHLD` (Child process status change)
  - `SIGSEGV` (Segmentation fault, default: core dump)
  - `SIGALRM` (Alarm signal, triggers after a timer expires)
- **Default Actions:**
  - Terminate (`SIGINT`, `SIGKILL`, `SIGTERM`)
  - Ignore (`SIGCHLD`, `SIGALRM` by default in some cases)
  - Core dump (`SIGSEGV`)

Writing a Signal Handler

- `sigaction()` allows defining custom handlers.

Syntax:
struct sigaction sa;
sa.sa_handler = my_handler;
sa.sa_flags = SA_RESTART;

- sigaction(SIGINT, &sa, NULL);
- Special values:
  - `SIG_IGN` (Ignore signal)
  - `SIG_DFL` (Restore default behavior)
  -

Writing a Signal Handler

- A signal handler is a function that takes an integer signal number as a parameter.

Example:

```
void my_handler(int signum) {
    printf("Caught signal %d\n", signum);

}
```

- Important:
  - Keep handlers simple.
  - Avoid non-reentrant functions (e.g., `printf`, `malloc`).
  - Why?

Q1

Question:
Consider the following pseudocode for handling `SIGINT`. What will happen when the user presses `Ctrl+C` three times?

```
void handler(int signum) {
    printf("SIGINT received! Ignoring...\n");
}

int main() {
    set_signal_handler(SIGINT, handler);
    while (1) {
        sleep(1);
    }
    return 0;
}
```

Q1

How can we modify the program to exit after receiving SIGINT three times?

```
void handler(int signum) {
    printf("SIGINT received! Ignoring...\n");
}

int main() {
    set_signal_handler(SIGINT, handler);
    while (1) {
        sleep(1);
    }
    return 0;
}
```

Signal Sets

- A signal set (`sigset_t`) represents a collection of signals.
- Functions for managing signal sets:
  - `sigemptyset(&set);` → Initializes an empty set.
  - `sigfillset(&set);` → Initializes a set with all signals.
  - `sigaddset(&set, SIGINT);` → Adds `SIGINT` to the set.
  - `sigdelset(&set, SIGTERM);` → Removes `SIGTERM` from the set.
  - `sigismember(&set, SIGKILL);` → Checks if `SIGKILL` is in the set.
- Used in functions like `sigprocmask()` and `sigsuspend()` to control signal behavior.

Blocking and Ignoring Signals

- Signals can be blocked using `sigprocmask()`.

Example:

```
sigset_t set;
sigemptyset(&set);
sigaddset(&set, SIGINT);
```

- sigprocmask(SIG_BLOCK, &set, NULL);
- Difference between ignoring (`SIG_IGN`) and blocking:
  - Ignored signals are discarded.
  - Blocked signals are deferred until unblocked.

Blocking and Ignoring Signals

Consider the following pseudocode where a process installs a signal handler for `SIGINT` and blocks `SIGTERM` for the first 5 seconds.

```
void handler(int signum) {
    printf("Received signal: %d\n", signum);
}

int main() {
    block_signal(SIGTERM);
    set_signal_handler(SIGINT, handler);

    sleep(5);

    unblock_signal(SIGTERM);
    while (1);
}
```

1. **What happens if `SIGINT` is sent before 5 seconds?**
2. **What happens if `SIGTERM` is sent before 5 seconds?**
3. **What happens if `SIGTERM` is sent after 5 seconds?**

Signals: Summary

- Signals provide a mechanism for process communication and exception handling.
- Use `sigaction()` to define custom handlers.
- Use `sigset_t` to manage multiple signals.
- `kill()` sends signals to processes.
- `waitpid()` helps manage child processes without blocking.
- `alarm(n)` sends SIGALRM after n seconds
- Block and ignore signals strategically to avoid unintended behavior.