CIS 5480
Recitation 03

Friday, September 18, 2026

Penn Engineering
University of Pennsylvania

Agenda

- Processes vs Pipelines vs Jobs
- Process Groups (pgids)
  - setpgid()
  - Sending signals to process groups
- Terminal Control
  - tcsetpgrp()
  - tcgetpgrp()
- Signal Handling vs Signal Sending
- Shell Lifecycle
- Demo

Process vs Pipelines vs Job

Process

- an executing program (PID)

Pipeline

- commands connected by pipes (e.g., a | b | c)

Job (shell concept)

- "a command line that the user typed"
  - may contain 1+ processes
  - has a state: foreground / background / stopped
    - Think about how you can implement this in PennOS!

Process Groups (PGIDs)

- Processes have a group id (PGID) and is usually the PID of the process that creates the group.
- During fork() the child inherits the PGID of the parent by default.

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

Process Groups (setpgid)

int setpgid(pid_t pid, pid_t pgid)

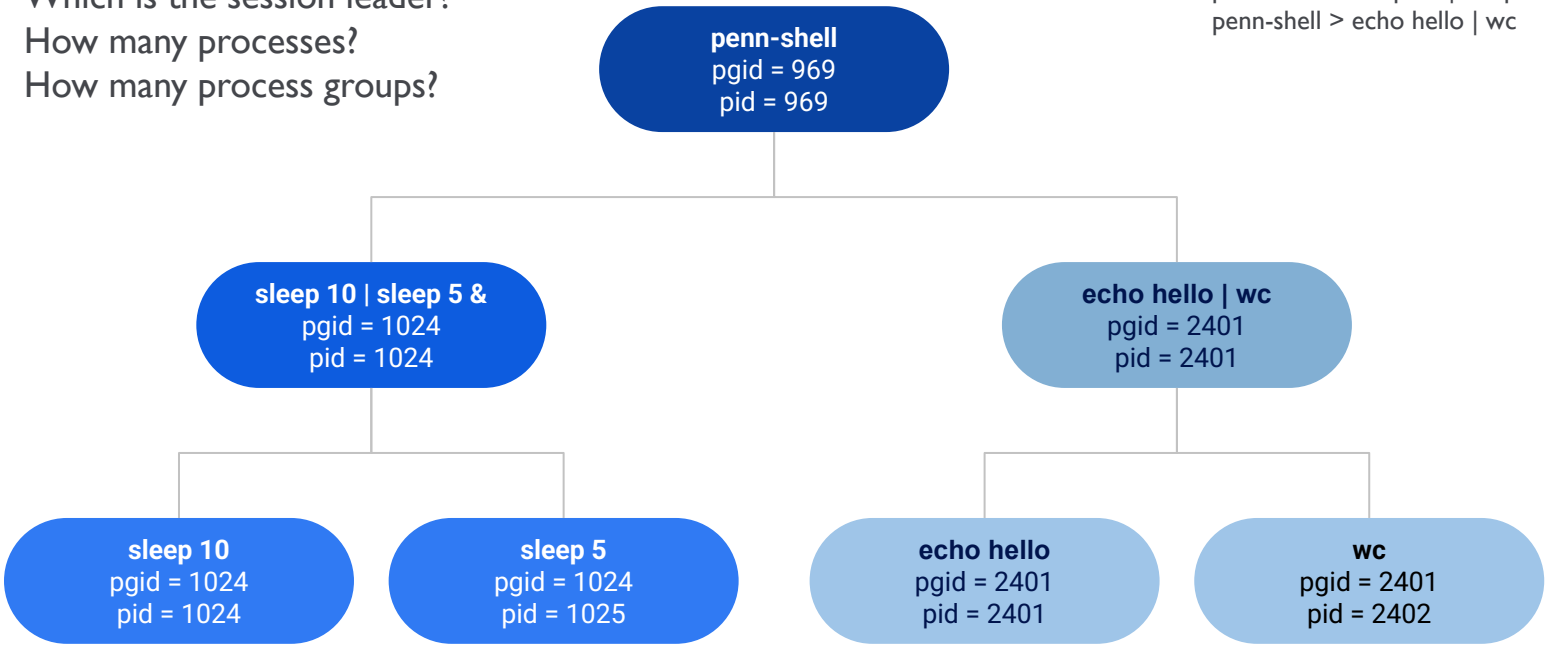
The PGID of the process, pid, is set to pgid

Call	**Meaning**
setpgid(0, X)	Change **my own** PGID to X
setpgid(P, 0)	Put P into its own PID's group
setpgid(0, 0)	Create a **new process group** with me as leader

Process Groups

1. Which is the session leader?
2. How many processes?
3. How many process groups?

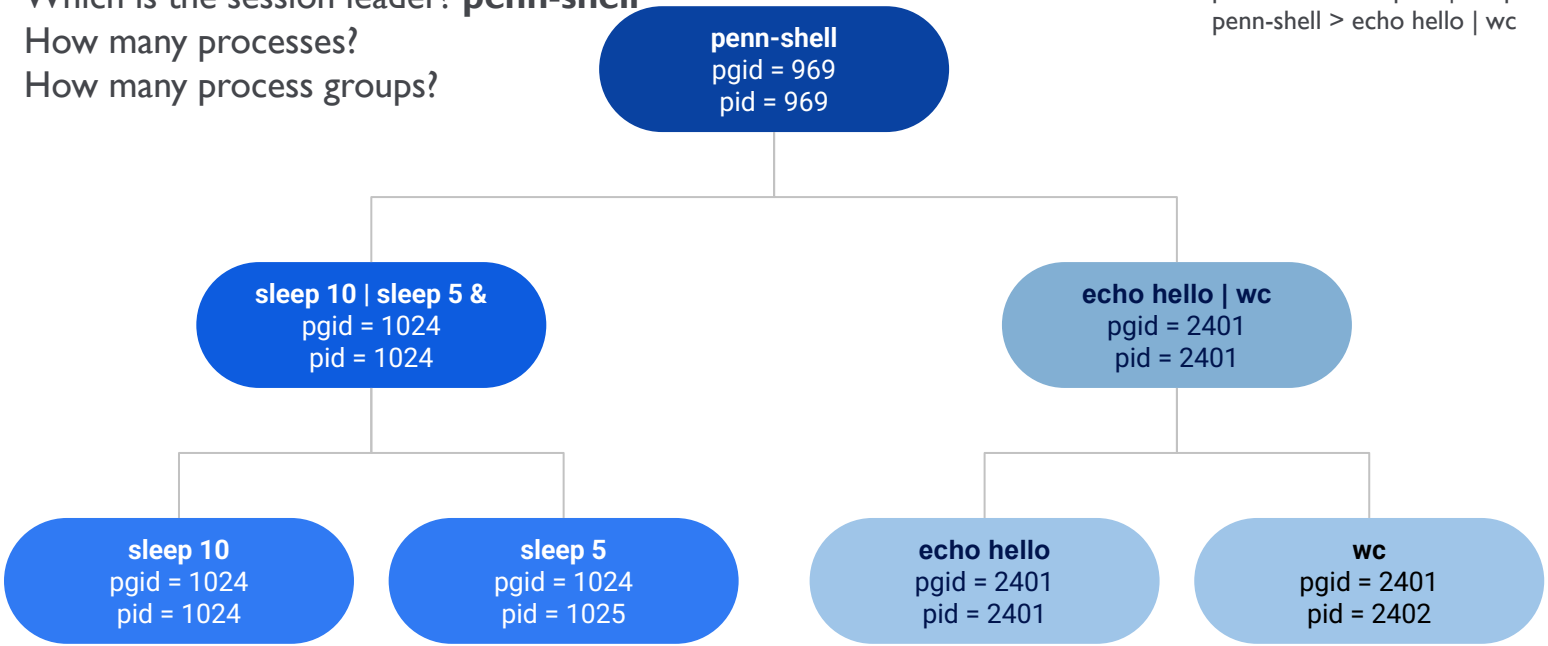
penn-shell > sleep 10 | sleep 5 &
penn-shell > echo hello | wc



Process Groups

1. Which is the session leader? **penn-shell**
2. How many processes?
3. How many process groups?

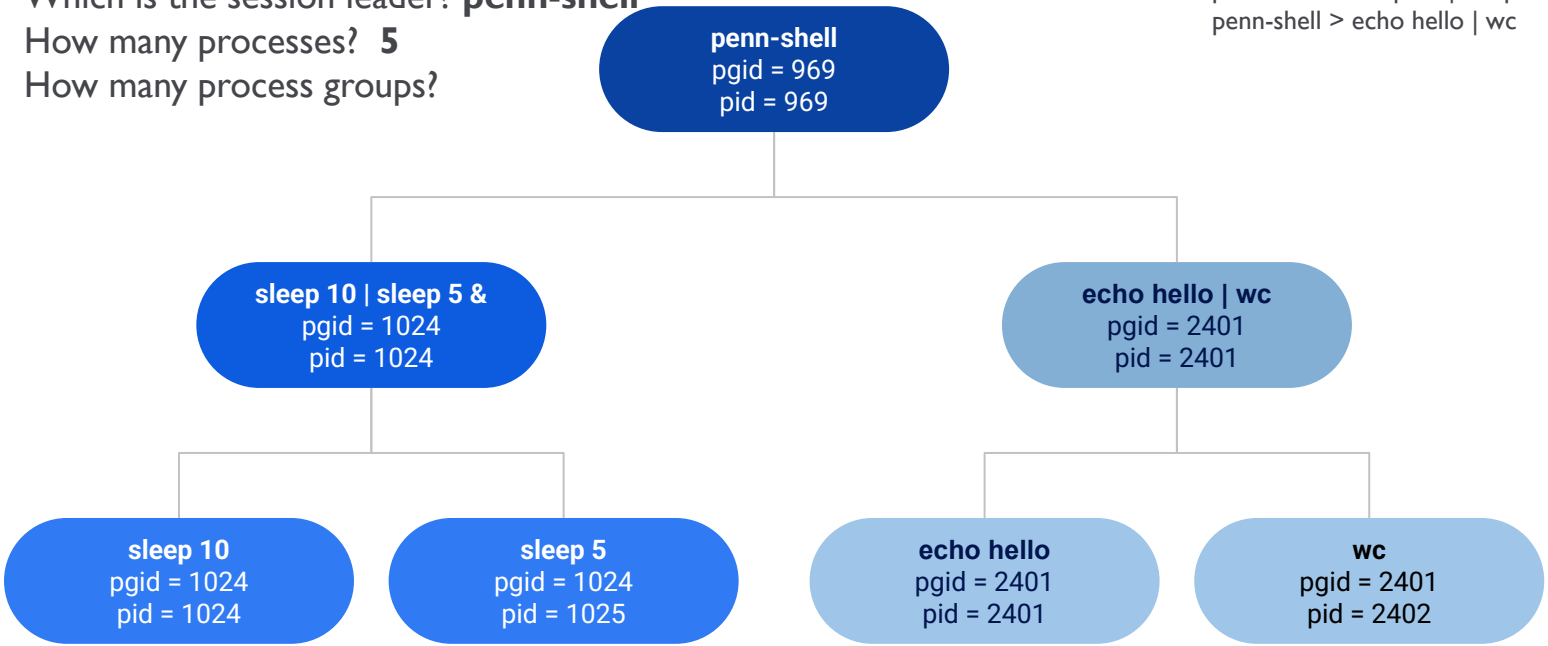
penn-shell > sleep 10 | sleep 5 &
penn-shell > echo hello | wc



Process Groups

1. Which is the session leader? **penn-shell**
2. How many processes? **5**
3. How many process groups?

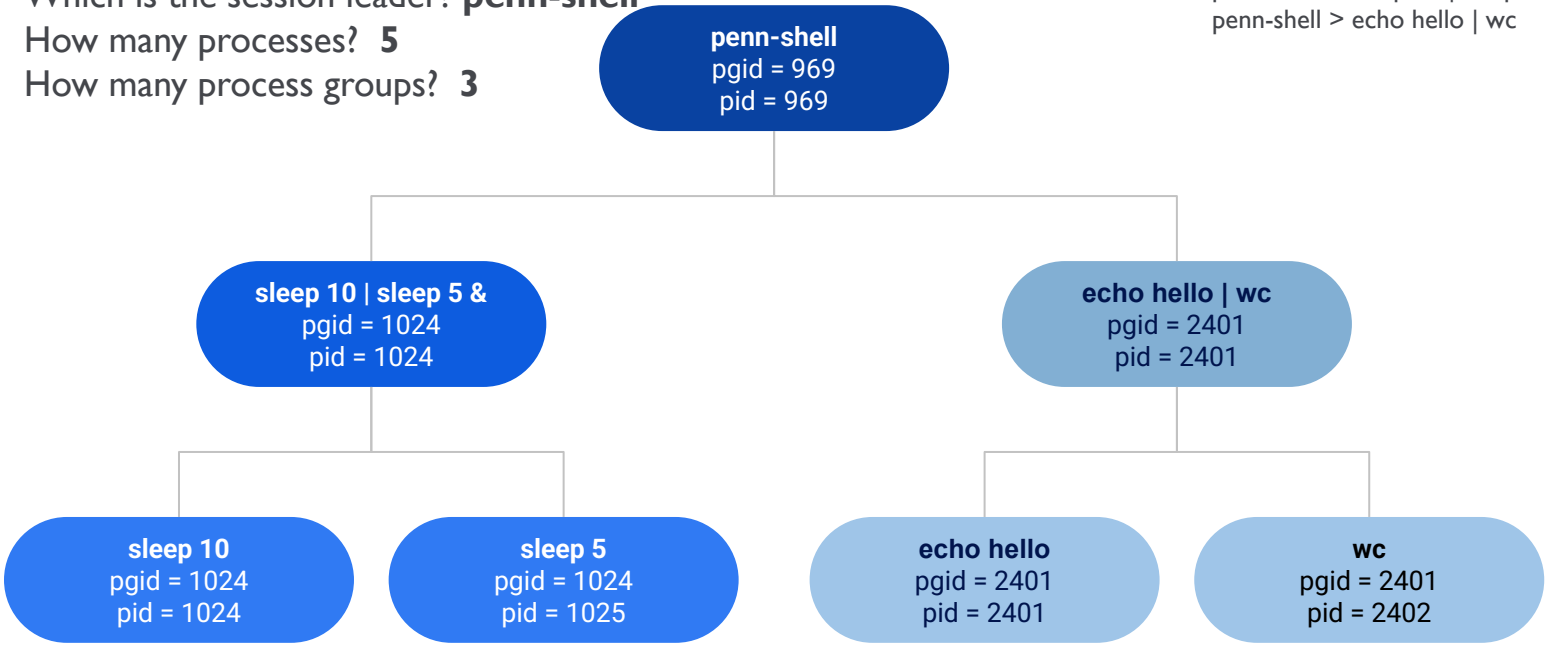
penn-shell > sleep 10 | sleep 5 &
penn-shell > echo hello | wc



Process Groups

1. Which is the session leader? **penn-shell**
2. How many processes? **5**
3. How many process groups? **3**

penn-shell > sleep 10 | sleep 5 &
penn-shell > echo hello | wc



Sending Signals and Waiting on Process Grps

- Instead of using a pid to refer to a singular process, you can pass in -PGID to kill() and waitpid()
  - **int kill(pid_t -pgid, int signal)**
- Doing so for kill() will send the signal to all processes in the group
  - **pid_t waitpid(pid_t -pgid, int *status, int options)**
- Doing so for waitpid() will wait for any process in the group

getpgid()

pid_t getpgid(pid_t pid)

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

What happens if pid is zero?

getpgid()

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).

Terminal Control

- Only one process group should have terminal control at a time
- Only the controlling group can read from and write to terminal, and receive terminal signals (`SIGINT` from Ctrl-C & `SIGTSTP` from Ctrl-Z)
  - Background processes do not have access to the terminal and can NOT read from it
  - Background processes CAN still write to a terminal

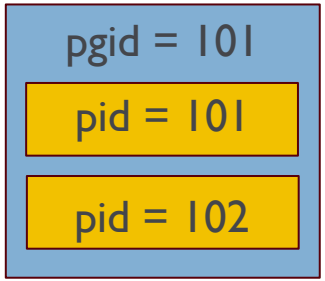
Terminal Control

- Which process group has terminal control at this point?
  - `pid_t tcgetpgrp(int fd);`
- What happens when another group (e.g., in the background) tries to access the terminal?
  - OS sends a `SIGTTIN` signal
  - Default action: stop the program

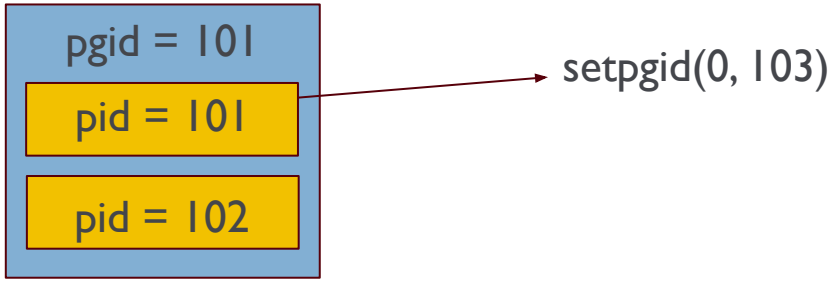
Terminal Control

- How can another process group take over terminal control?
  - **`int tcsetpgrp(int fd, pid_t pgrp);`**
  - `fd is a file descriptor associated with the controlling terminal (STDIN_FILENO)`
  - `Sets the process group specified by pgrp to be the foreground process group`
- What if we need to read from stdin from the background?
  - Call `tcsetpgrp()` and receive a `SIGTTOU` signal (default: stop the program)
  - We can configure it to block or ignore this signal

Job Control

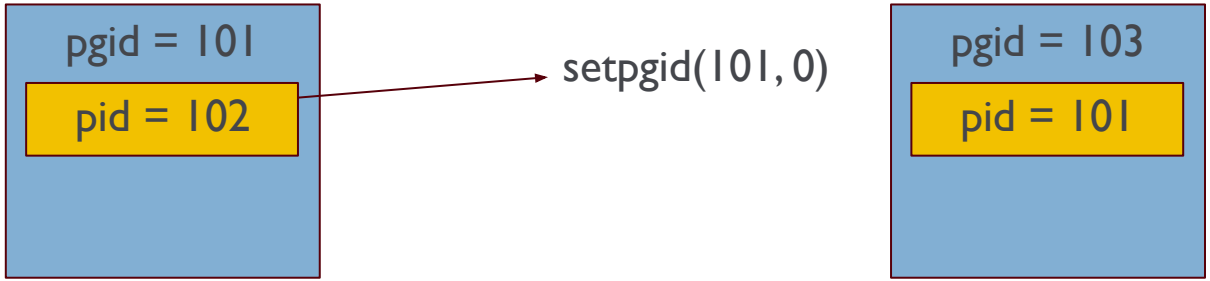


Job Control

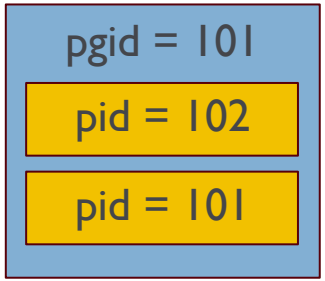


Job Control

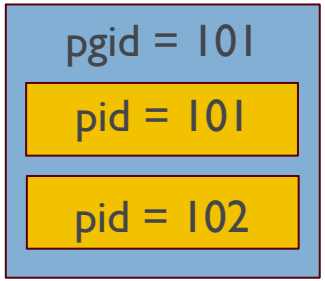
Job Control



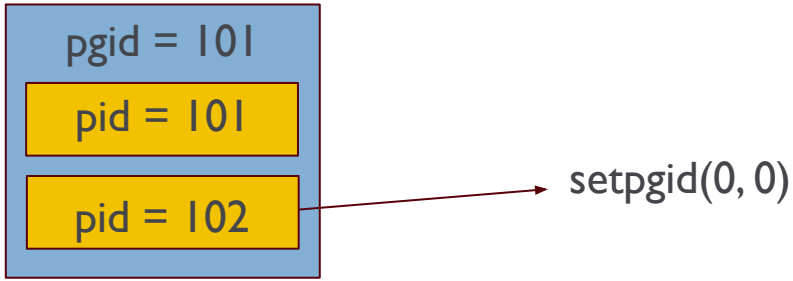
Job Control



Job Control



Job Control



Job Control

Worksheet #1.1 Foreground

**pls ignore that question/answer screenshots say that this is Q4. These ss' are from a previous semester. In fa26, this is Q1 on the recitation worksheet.

4 It's Terminal (Control)

4.1 Sessions and Terminals

The following output is given from a run of the `ps` command. Use the details to answer the questions below.

Note: TPGID is the PGID for the current foreground process on a given terminal.

USER	PPID	PID	PGRP	TPGID	SID	TTY	CMD
kyrdo	35943	35950	35950	68461	35950	pts/0	-bash
kyrdo	53690	53695	53695	68468	53695	pts/2	-bash
kyrdo	35950	68406	68405	68461	35950	pts/0	sleep 200
kyrdo	35950	68461	68461	68461	35950	pts/0	cat
kyrdo	35950	68462	68461	68461	35950	pts/0	grep How
kyrdo	35950	68463	68461	68461	35950	pts/0	uniq
kyrdo	53695	68464	68464	68468	53695	pts/2	sleep 300
kyrdo	53695	68465	68464	68468	53695	pts/2	sleep 200
kyrdo	53695	68468	68468	68468	53695	pts/2	ps

Foreground. For each foreground job, identify the following: 1) the commands that make up the job and their associated process ID, 2) the process group of the job, 3) the job's parent process and 4) what terminal and session the process is connected to.

Worksheet #1.1 Foreground Solution

**pls ignore that question/answer screenshots say that this is Q4. These ss' are from a previous semester. In fa26, this is Q1 on the recitation worksheet.

4 It's Terminal (Control)

4.1 Sessions and Terminals

The following output is given from a run of the **ps** command. Use the details to answer the questions below.

Note: TPGID is the PGID for the current foreground process on a given terminal.

foreground job = all processes whose PGRP = TPGID

USER	PPID	PID	PGRP	TPGID	SID	TTY	CMD
kyrdo	35943	35950	35950	68461	35950	pts/0	-bash
kyrdo	53690	53695	53695	68468	53695	pts/2	-bash
kyrdo	35950	68406	68405	68461	35950	pts/0	sleep 200
kyrdo	35950	68461	68461	68461	35950	pts/0	cat
kyrdo	35950	68462	68461	68461	35950	pts/0	grep How
kyrdo	35950	68463	68461	68461	35950	pts/0	uniq
kyrdo	53695	68464	68464	68468	53695	pts/2	sleep 300
kyrdo	53695	68465	68464	68468	53695	pts/2	sleep 200
kyrdo	53695	68468	68468	68468	53695	pts/2	ps

Foreground. For each foreground job, identify the following: 1) the commands that make up the job and their associated process ID, 2) the process group of the job, 3) the job's parent process and 4) what terminal and session the process is connected to.

Command	PIDs
cat	68461
grep How	68462
uniq	68463

job prc grp: 68461

parent process:

all children have PPID = 35950

so bash, pid 35950

TTY = pts/0

SID = 35950

Command	PID
ps	68468

pgrp: 68468

parent: bash (53695)

TTY: pts/2

session: 53695

Penn Engineering

Worksheet #1.1 Background

**pls ignore that question/answer screenshots say that this is Q4. These ss' are from a previous semester. In fa26, this is Q1 on the recitation worksheet.

Note: TPGID is the PGID for the current foreground process on a given terminal.

USER	PPID	PID	PGRP	TPGID	SID	TTY	CMD
kyrdo	35943	35950	35950	68461	35950	pts/0	-bash
kyrdo	53690	53695	53695	68468	53695	pts/2	-bash
kyrdo	35950	68406	68405	68461	35950	pts/0	sleep 200
kyrdo	35950	68461	68461	68461	35950	pts/0	cat
kyrdo	35950	68462	68461	68461	35950	pts/0	grep How
kyrdo	35950	68463	68461	68461	35950	pts/0	uniq
kyrdo	53695	68464	68464	68468	53695	pts/2	sleep 300
kyrdo	53695	68465	68464	68468	53695	pts/2	sleep 200
kyrdo	53695	68468	68468	68468	53695	pts/2	ps

Foreground. For each foreground job, identify the following: 1) the commands that make up the job and their associated process ID, 2) the process group of the job, 3) the job's parent process and 4) what terminal and session the process is connected to.

Background. For each background job, identify the following: 1) the commands that make up the job and their associated process ID, 2) the process group of the job, 3) the job's parent process and 4) what terminal and session the process is connected to.

Worksheet #1.1 Background Solution

**pls ignore that question/answer screenshots say that this is Q4. These ss' are from a previous semester. In fa26, this is Q1 on the recitation worksheet.

4 It's Terminal (Control)

4.1 Sessions and Terminals

The following output is given from a run of the **ps** command. Use the details to answer the questions below.

Note: TPGID is the PGID for the current foreground process on a given terminal.

USER	PPID	PID	PGRP	TPGID	SID	TTY	CMD
kyrdo	35943	35950	35950	68461	35950	pts/0	-bash
kyrdo	53690	53695	53695	68468	53695	pts/2	-bash
kyrdo	35950	68406	68405	68461	35950	pts/0	sleep 200
kyrdo	35950	68461	68461	68461	35950	pts/0	cat
kyrdo	35950	68462	68461	68461	35950	pts/0	grep How
kyrdo	35950	68463	68461	68461	35950	pts/0	uniq
kyrdo	53695	68464	68464	68468	53695	pts/2	sleep 300
kyrdo	53695	68465	68464	68468	53695	pts/2	sleep 200
kyrdo	53695	68468	68468	68468	53695	pts/2	ps

Foreground. For each foreground job, identify the following: 1) the commands that make up the job and their associated process ID, 2) the process group of the job, 3) the job's parent process and 4) what terminal and session the process is connected to.

pts/0

Command PID

sleep 200 68406

PGRP: 68405

PPID = 35950 → bash on pts/0

TTY = pts/0

SID = 35950

---

pts/2

Command PIDs

sleep 300 68464

sleep 200 68465

PGRP = 68464

PPID = 53695 → bash on pts/2

TTY = pts/2

SID = 53695

Penn Engineering

Shell Lifecycle

Imagine ls | wc is entered into the shell

1. Shell spawns
    a. The shell has its own pid and pgid
    b. Shell has its own signal handling installed
2. The command ls | wc is read by the shell
3. Define a process group id for every process in that job
    a. Every job maps to exactly one process group
4. Shell forks twice for its two children
    a. Parent and both children call setpgid(0, pgid)
        i. What if the shell moves terminal control to pgid before the child joins the group?

SUPER IMPORTANT FOR SHELL MILESTONE I

Shell Lifecycle Continued

5. If either of the children call exec, then their signal handling is reset. Otherwise, the signal handling is inherited from the parent (shell).
6. Give the job terminal control if it runs in the foreground
    a. tcsetpgrp(STDIN_FILENO, job_pgid)
7. Shell calls a waitpid on children that are in that process group
    b. waitpid(-pgid, …)
8. Signal kill(-pgid, SIGINT) sent to all processes with that pgid
    c. If job terminates then the shell gets terminal control again.

*****SUPER IMPORTANT FOR SHELL MILESTONE I*****

Worksheet #1.2

**pls ignore that question/answer screenshots say that this is Q4. These ss' are from a previous semester. In fa26, this is Q1 on the recitation worksheet.

4.2 Process Groups and Penn-Shell

Your `penn-shell` partner claims that you do not need to isolate a foreground job into its own process group and assign it terminal control as the parent process will wait for it to finish anyway before trying to do anything with the terminal. Do you agree with your partner? Why or why not?

Worksheet #1.2 Solution

Waiting and terminal control aren't the same!
The terminal delivers signals to the **foreground process group**, not to the parent shell.

- Waiting doesn't change who receives keyboard input, who receives Ctrl-C, Ctrl-Z, or who is allowed to read from/write to the terminal

1.2 Example Scenario

Let's say:

- Shell PID = 100
- Shell PGID = 100
- User runs sleep 200

If we don't create a new process group, then sleep stays in PGID 100. Shell and sleep are in same process group. Shell calls wait(). Now user calls Ctrl-C.

1.2 Example Scenario Continued

The terminal sends SIGINT to the **foreground process group.** That is PGID 100!

Shell gets SIGINT.
Sleep gets SIGINT.

Shell dies

Remember: wait() only blocks the shell in user space, but does nothing about signals, terminal ownership, or job control

Worksheet #2.1a

2. Race to the Terminal

2.1 Who Gets There First?

Your penn-shell partner has written the following code to launch a single foreground job. Assume the shell currently owns the terminal and `pid` will also be used as the new job's PGID.

```
pid_t pid = fork();

if (pid == 0) {
   execvp(argv[0], argv);
   perror("execvp");
   exit(EXIT_FAILURE);
} else {
   setpgid(pid, pid);
   tcsetpgrp(STDIN_FILENO, pid);

   waitpid(pid, NULL, 0);
}
```

Your partner tests the shell and notices that it usually works, but occasionally setpgid()/terminal setup fails or the child process gets stopped.

a. Recall that a race condition occurs when some behaviour depends on the order in which the processes are run. What race condition exists between the parent and child?

Worksheet #2.1a Solution

The shell forks a child, while the parent is responsible for moving that child into a new process group and handing that group terminal control. The result can therefore depend on whether the parent or child runs first. If the child execvp()s before the parent successfully calls setpgid(), the parent may no longer be able to change that child's process group.
If the child has been moved into its own PGID but that PGID has not yet received terminal control, the child is temporarily a background process. If it tries to read from the terminal, it can receive SIGTTIN and be stopped.

Worksheet #2.1b

b. Consider the following two possible schedules:

Schedule A: Parent executes setpgid() before the child executes execvp().

Schedule B: Child executes execvp() before the parent executes setpgid().

Explain what can happen in each case.

Worksheet #2.1b Solution

Schedule A: parent runs setpgid() first

- Child becomes part of PGID 200.
- PGID 200 receives terminal control.
- Child executes its program as the foreground job.

Schedule B: child execvp()s first

Now the parent may be too late to change the child's process group.

Child executes too early → Parent's setpgid() may fail → Job-control setup is incorrect

Worksheet #2.1c

2.1 Who Gets There First?

Your penn-shell partner has written the following code to launch a single foreground job. Assume the shell currently owns the terminal and `pid` will also be used as the new job's PGID.

```
pid_t pid = fork();

if (pid == 0) {
   execvp(argv[0], argv);
   perror("execvp");
   exit(EXIT_FAILURE);
} else {
   setpgid(pid, pid);
   tcsetpgrp(STDIN_FILENO, pid);

   waitpid(pid, NULL, 0);
}
```

Modify the code (above) so that establishing the child's process group does not depend on whether the parent or child runs first.

Worksheet #2.1c Solution

```c
pid_t pid = fork();

if (pid == 0) {
    setpgid(0, 0); // child puts itself in its own group

    execvp(argv[0], argv);
    perror("execvp");
    exit(EXIT_FAILURE);

} else {
    setpgid(pid, pid); // parent also puts child in its own group

    tcsetpgrp(STDIN_FILENO, pid);

    waitpid(pid, NULL, 0);
```

Worksheet #2.2a

2.2 Pipeline Launch

Your shell is launching a pipeline. All processes in the pipeline should belong to the same process group, with the first child acting as the process-group leader. Consider the following simplified code:

```
pid_t leader_pid = 0;

for (int i = 0; i < pipeline_length; ++i) {
   pid_t pid = fork();

   if (pid == 0) {
       if (i == 0) {
           // First child creates the process group
           setpgid(0, 0);
       } else {
           // Later children join the first child's group
           setpgid(0, leader_pid);
       }

       execvp(...);

   } else {
       if (i == 0) {
           leader_pid = pid;
       }
   }
}
```

Assume the first child receives PID 200, so the intended process group for the entire pipeline is PGID 200.

a. There is a race condition in this code snippet. What is it?

Worksheet #2.2a Solution

Knowing that the intended PGID is 200 does not mean that PGID 200 already exists. The parent can know leader_pid == 200 while child 200 has not yet actually created that process group.

Worksheet #2.2b

b. Modify the shell so that creation of PGID 200 does not depend solely on whether child 200 happens to run before child 201.

Worksheet #2.2b Solution

Do not make creation of PGID 200 depend solely on whether child 200 happens to run first.

```
1   pid_t leader_pid = 0;
2
3   for (int i = 0; i < pipeline_length; ++i) {
4       pid_t pid = fork();
5
6       if (pid == 0) {
7           if (i == 0) {
8               setpgid(0, 0);
9           } else {
10              setpgid(0, leader_pid);
11          }
12
13          execvp(...);
14
15      } else {
16          if (i == 0) {
17              leader_pid = pid;
18
19              // Parent also establishes PGID 200
20              setpgid(pid, pid);
21          }
22      }
23  }
```

Worksheet #3A

3: Who Gets the Signal?

Assume the following:
Shell PID = 100
Shell PGID = 100
Child PID = 200

For each scenario below:

1. Identify the foreground process group.
2. Identify which process(es) receive SIGINT if the user types Ctrl-C.
3. Identify which process(es) receive SIGTSTP if the user types Ctrl-Z.

Scenario A: No PGrp Setup

```
pid_t pid = fork();

if (pid == 0) {
    execvp(...);
} else {
    waitpid(pid, NULL, 0);
}
```

Worksheet #3A Solution

No process group setup.
Because no setpgid() occurs, the child inherits the shell's PGID.

FG Pgrp: 100
Ctrl+C and Ctrl+Z:
PGID 100

- Shell (PID 100)
- Child (PID 200)

Worksheet #3B

Scenario B: PGrp But No Terminal Handoff

```
if (pid == 0) {
    setpgid(0, 0);
    execvp(...);
}

setpgid(pid, pid);
waitpid(pid, NULL, 0);
```

Worksheet #3B Solution

Foreground PGID: 100
Ctrl+C: shell only
Ctrl+Z: shell only

Worksheet #3C

Scenario C: Foreground Job Setup

```
setpgid(pid, pid);
tcsetpgrp(STDIN_FILENO, pid);
waitpid(-pid, NULL, 0);
```

Worksheet #3C Solution

Foreground PGID: 200
Ctrl+C: child only
Ctrl+Z: child only

Worksheet #3D

Scenario D: Pipeline

```
Shell: PID 100, PGID 100

cat:   PID 200, PGID 200
grep:  PID 201, PGID 200
wc:    PID 202, PGID 200

Foreground PGID = 200
```

Worksheet #3D Solution

Foreground PGID: 200
Ctrl+C: cat (PID 200), grep (PID 201), wc (PID 202)
Ctrl+Z: cat (PID 200), grep (PID 201), wc (PID 202)

Penn Shell Demo