Recitation 2

Welcome back, everybody!

Today's Topics

- Signals
- Redirection and Pipes
- Process Groups

Signals

Signals

- Software interrupt - a notification sent to a process

- An example of inter-process communication
  - What is another example of inter-process communication?

- From the OS: some sort of exception has occurred
- From another process: used to coordinate activity or as notification

Activity: Responses to Signals

In each scenario, determine whether the signal was blocked, ignored, used default handling, or used a handler function (non-default)

1. In penn-shredder, I hit Ctrl+Z. In response, penn-shredder stopped, and now I'm back in the host shell.

Activity: Responses to Signals

In each scenario, determine whether the signal was blocked, ignored, used default handling, or used a handler function (non-default)

1. In penn-shredder, I hit Ctrl+Z. In response, penn-shredder stopped, and now I'm back in the host shell. **Default Action**
2. In penn-shell, I hit Ctrl+Z. In response, penn-shell reprompted.

Activity: Responses to Signals

In each scenario, determine whether the signal was blocked, ignored, used default handling, or used a handler function (non-default)

1. In penn-shredder, I hit Ctrl+Z. In response, penn-shredder stopped, and now I'm back in the host shell. **Default Action**
2. In penn-shell, I hit Ctrl+Z. In response, penn-shell reprompted. **Signal Handler Function**
3. The parent process takes terminal control from its child and gives it to itself. The parent process was not stopped by SIGTTIN or SIGTTOU, and continues as normal.

Activity: Responses to Signals

In each scenario, determine whether the signal was blocked, ignored, used default handling, or used a handler function (non-default)

1. In penn-shredder, I hit Ctrl+Z. In response, penn-shredder stopped, and now I'm back in the host shell. **Default Action**
2. In penn-shell, I hit Ctrl+Z. In response, penn-shell reprompted. **Signal Handler Function**
3. The parent process takes terminal control from its child and gives it to itself. The parent process was not stopped by SIGTTIN or SIGTTOU, and continues as normal. **Signal Ignored**
4. A process received a signal, and its signal handler called vec_push. While in the middle of executing vec_push, it received the same signal. Once the first call to vec_push returned, the signal handler ran vec_push again.

Activity: Responses to Signals

In each scenario, determine whether the signal was blocked, ignored, used default handling, or used a handler function (non-default)

1. In penn-shredder, I hit Ctrl+Z. In response, penn-shredder stopped, and now I'm back in the host shell. **Default Action**
2. In penn-shell, I hit Ctrl+Z. In response, penn-shell reprompted. **Signal Handler Function**
3. The parent process takes terminal control from its child and gives it to itself. The parent process was not stopped by SIGTTIN or SIGTTOU, and continues as normal. **Signal Ignored**
4. A process received a signal, and its signal handler called vec_push. While in the middle of executing vec_push, it received the same signal. Once the first call to vec_push returned, the signal handler ran vec_push again. **Signal Blocked, then Unblocked**

Redirections and Pipes

Pipelines

- Consider the following pipelined command:

```
# sleep 1 | sleep 20 | sleep 100
```

  How long does it take to finish?

Pipelines

- Consider the following pipelined command:
  ```
  # sleep 1 | sleep 20 | sleep 100
  ```

  How long does it take to finish?
  100 seconds

  Why?

Pipelines

- Consider the following pipelined command:

```
# sleep 1 | sleep 20 | sleep 100
```

How long does it take to finish?
100 seconds

Why?
Pipelined processes run in parallel

Let’s take a look at two_pipe_animation :)

- Can be found on the website’s schedule page, June 9th’s lecture

Let's take a look at two_pipe_animation :)

- Can be found on the website's schedule page, June 9th's lecture
- How might we achieve this elegant procedure?

Let's take a look at two_pipe_animation :)

- Can be found on the website's schedule page, June 9th's lecture

- How might we achieve this elegant procedure?
- Closing redundant file descriptors
  - What does the child not use?
  - What does the parent need to hold on to before forking the next child?

Process Groups

Activity: True or False

1. Only foreground jobs can write to stdout and stderr.

Activity: True or False

1. Only foreground jobs can write to stdout and stderr. **FALSE**
2. Only foreground jobs can receive the signals sent from CTRL+C and CTRL+Z.

Activity: True or False

1. Only foreground jobs can write to stdout and stderr. **FALSE**
2. Only foreground jobs can receive the signals sent from CTRL+C and CTRL+Z. **TRUE**
3. Only foreground jobs can receive SIGINT and SIGTSTP.

Activity: True or False

1. Only foreground jobs can write to stdout and stderr. **FALSE**
2. Only foreground jobs can receive the signals sent from CTRL+C and CTRL+Z. **TRUE**
3. Only foreground jobs can receive SIGINT and SIGTSTP. **FALSE**
4. You can only call fg on a background process that is stopped.

Activity: True or False

1. Only foreground jobs can write to stdout and stderr. **FALSE**
2. Only foreground jobs can receive the signals sent from CTRL+C and CTRL+Z. **TRUE**
3. Only foreground jobs can receive SIGINT and SIGTSTP. **FALSE**
4. You can only call fg on a background process that is stopped. **FALSE**
5. A parent process can call wait() or waitpid() on a child that is in a different process group.

Activity: True or False

1. Only foreground jobs can write to stdout and stderr. **FALSE**
2. Only foreground jobs can receive the signals sent from CTRL+C and CTRL+Z. **TRUE**
3. Only foreground jobs can receive SIGINT and SIGTSTP. **FALSE**
4. You can only call fg on a background process that is stopped. **FALSE**
5. A parent process can call wait() or waitpid() on a child that is in a different process group. **TRUE**

Process Groups and waitpid()

waitpid(pid_t *pid*, int **status*, int *options*)

Options:

- WNOHANG: do not wait for process to finish, but “collect” already finished or changed-state processes. Move on otherwise.
- WUNTRACED: also return if the child has stopped

Pid:

- -1: wait for all children
- Negative of pgid: wait for children of a specific process group
- Pid: wait for a child with specific process id

waitpid(pid_t *pid*, int **status*, int *options*)

Options:

- WNOHANG: do not wait for process to finish, but “collect” already finished or changed-state processes. Move on otherwise.
- WUNTRACED: also return if the child has stopped

Pid:

- -1: wait for all children
- Negative of pgid: wait for children of a specific process group
- Pid: wait for a child with specific process id

Be aware that STOP and CONTINUE are also detected by waitpid()!

Waiting in Project 2

- How does waitpid and its options come up?
- When should terminal control change?
- Will you encounter zombies? If so, how will you handle them?
- How do process groups factor in?

Waiting in Project 2

- How does waitpid and its options come up?
  - Depending on the job being in fg or bg, we need different waitpid args
- When should terminal control change?
  - When the role of “foreground process group” changes
- Will you encounter zombies? If so, how will you handle them?
  - When background jobs finish first, they will be zombies until reaped by parent calling waitpid()
- How do process groups factor in?
  - One command (job) with all its pipelined components will correspond to one process group

Shell OH?