Iteration V - Using Signals

```
===========================================================================
```

Due Date: March 10, 2004, 11:59:59 PM

Useful man pages

```
=================
```

Needed: sigaction, *setpgid, sched_yield, kill(2), killpg, signal(7)
FYI: sigemptyset, sigaddset, sigprocmask, sigsuspend

To get a list of signals: type "kill -l"

Important signals to know about:

```
    (1) SIGKILL
    (2) SIGSTOP
    (3) SIGCONT
    (4) SIGTSTP
    (5) SIGINT
```

A description of each signal can be found in the signal(7) man page.

NOTE:

```
======
```

It is mandatory that you read the "sigaction" and setpgid man pages *completely*. Every answer I give will assume that you have read and understood these man pages. If you do not understand them, then I would suggest that you ask early.

Important: Process Groups - setpgid(), getpgid()

```
================================================
```

This is an important concept in UNIX. When a process is created, it is assigned to a group (usually the same group as its parent). This group is assigned an ID that is equivelant to the PID of the group leader, which was the first process to belong to the group. If a signal is sent to the Group ID (equivelant to the PID of the group leader) than the *whole* group receives the signal.

This impacts our program because we are operating under another shell program (the one you logged in to). This shell lauches our program as a group leader to a new group. As our program creates children, these children are assigned to the same group as our program.

When the user presses Ctrl-Z, this signal is intercepted by the shell (not our program), and sent to our program. Since our program is the group leader, that means that the signal is sent to *every* one of our children.

While this behavior is normal, it is not the behavior that we would like. To prevent the propagation of the signal, we need to assign each child to its own group (using setpgid()). We can then act more like a shell.

In iteration 6, we will be using these groups to assist us in propagating signals properly, so please understand the concept thouroughly.

Objects I made

==============

Parser	-	Parses the input into tokens
Executor	-	Uses the tokens to create a Process
Process	-	Object that symbolizes a process in the Operating System
Signal	-	Object that symbolizes a signal in the Operating System
ProcControl	-	List container of Processes

Overview

Now our project can execute an outside binary that we tell it to, with the arguments that we tell it to use, with little limitation. While we can run multiple program at once, we have little control over them. This control is what will be introduced in this iteration - through signals.

A signal is a method of inter-process communication. By using signals, we can suspend a process and then force the process to continue when we are ready.

We will be adding onto the previous iteration. The following functions will be added in this iteration:

* Using signals to control process activity

Description:

You are to extend your previous program by adding the ability to do the following:

1. All operations done in the previous iterations must work. This means that all tests performed on previous iterations need to continue to work.

2. Handle, at a minimum, the following signals:

    A. SIGTSTP (Ctrl-Z) - When this signal is received, your program should suspend the current foreground process using the STOP signal.

    If there is no process in the foreground, nothing should happen. Though it is permissible to show another prompt (i.e. pretend the user pressed enter).

    When a process is suspended, the program should reshow the prompt, so that another program can be run.

    B. SIGINT (Ctrl-C) - When this signal is received, your program kill the current foregound process using the KILL signal. If there is no current foreground process then the signal should do nothing. it is permissible to pretend that the user pressed enter.

    C. SIGSEGV - When this signal is received, then your program should output the fact that a Segmentation fault has occurred, then exit. While this handler is useful, it may interfere with debugging tools (such as gdb). Therefore, before submitting your program, it is asked that you comment out the code for this handler (just in case we need it later).

```
    D.  SIGCHLD - This signal is used to tell the parent process (your
        program) about the current status of its children.  In this
        program we use this signal to determine whether a child has been
        killed, has exited normally, or has been suspended or continued.

        Each time one of these event occurs, the user should be told
        about this fact.  See the sigaction man page for the SIGCHLD
        si_codes.

3.  We will be adding three new tokens:

    A.  bg -    When this is typed in, the last process that was
        suspended should be continued in the background.  That is, the
        process should be sent the CONT signal.  This will cause the
        suspended process to continue in the background.  While this
        process is runnig in the background, the user should still be
        able to run another command through the prompt.

    B.  fg -    When this is typed in, the last process that was
        suspended or backgrounded (which ever happened first) should be
        brought to the foreground.  That is, the process should be given
        focus and should be able to receive keyboard input.

    C.  jobs -  When this is typed in, a list of the current processes
        should be displayed to the screen.  The current state of the
        process (whether it is running or suspended) need not be shown,
        though it may be added if wanted.

    NOTE:   Be sure to try the sample executable to verify that
    thebehavior of your program is similiar as that is the minimum
    expected.

Glossary
=========

Foreground Process -    The process that is focued and able to receive
                        keyboard input.  There can only be one process
                        that is in the foreground

Background Process -    Any process that is running in the background.
                        These processes cannot receive keyboard input,
                        though it is possible that they will output to
                        the screen


SAMPLE OUTPUT
================

Script started on Sun Feb 15 21:57:20 2004
 ]0;sea@sabatour: /home/sea/Projects/TA-CS284/WS04/src/iteration5 src/iteration5> ./
sea:sabatour:src/iteration5:> ./TEST

COMMAND EXEC: ./TEST
Argument Table:
        Entry 0: ./TEST
Entry 1: NULL - End of Table

TEST 0
TEST 1
```

```
Suspended 30071 ./TEST
sea:sabatour:src/iteration5:> bg
Backgrounded 30071 ./TEST
[0] 30071 ./TEST

sea:sabatour:src/iteration5:> TEST 2
TEST 3
./TEST

COMMAND EXEC: ./TEST
Argument Table:
        Entry 0: ./TEST
Entry 1: NULL - End of Table

TEST 4

CHILD exited: PID 30071
sea:sabatour:src/iteration5:> TEST 0
TEST 1

sea:sabatour:src/iteration5:> TEST 2
TEST 3
TEST 4

CHILD exited: PID 30072
sea:sabatour:src/iteration5:> ./TEST

COMMAND EXEC: ./TEST
Argument Table:
        Entry 0: ./TEST
Entry 1: NULL - End of Table

TEST 0
sea:sabatour:src/iteration5:>
Suspended 30073 ./TEST
sea:sabatour:src/iteration5:> fg
Forground process 30073 ./TEST
TEST 1
TEST 2
TEST 3
TEST 4

CHILD exited: PID 30073
sea:sabatour:src/iteration5:> ./TEST

COMMAND EXEC: ./TEST
Argument Table:
        Entry 0: ./TEST
Entry 1: NULL - End of Table

TEST 0
TEST 1

Suspended 30074 ./TEST
sea:sabatour:src/iteration5:> jobs
[0] 30074 ./TEST

sea:sabatour:src/iteration5:> fg
Forground process 30074 ./TEST
```

```
TEST 2
TEST 3
TEST 4

CHILD exited: PID 30074
sea:sabatour:src/iteration5:> jobs

sea:sabatour:src/iteration5:>
Program exit.
src/iteration5> ^D  exit

Script done on Sun Feb 15 21:58:03 2004
```