CSCI 4210 — Operating Systems
CSCI 6140 — Computer Operating Systems
Homework 3 (document version 1.2)
Inter-Process Communication (IPC) using Pipes in C

Overview

- This homework is due by 11:59:59 PM on **Tuesday, October 13, 2015**. Homeworks are to be submitted electronically.
- This homework will count as 4% of your final course grade.
- This homework is to be completed **individually**. Do not share your code with anyone else.
- You **must** use C for this homework assignment.
- Your program **must** successfully compile and run on Ubuntu v14.04.3 LTS or v15.04.
- Your program **must** successfully compile via `gcc` with no warning messages when using the `-Wall` compiler option.

Homework Specifications

In this third homework, you will use C to implement a fork-based calculator program. The goal is to work with pipes and processes, i.e., inter-process communication (IPC).

Overall, your program will read an input file that specifies a set of calculations to be made by using a Scheme/LISP-like format. More specifically, your program will call `fork()` to create child processes to perform each calculation, thus forming a process tree that represents the parsed mathematical expression.

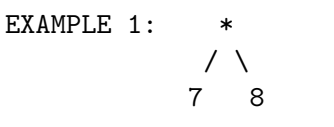
The input file is specified on the command-line as the first argument and consists of one expression to be evaluated. For this homework, you must support addition, subtraction, multiplication, and division, adhering to the standard mathematical order of operations. Note that each of these operations must have at least two operands (display an error if this is not the case).

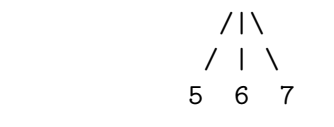
Example expressions are shown below.

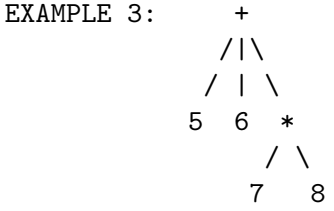
```
# EXAMPLE 1: 7 * 8
#
(* 7 8)

# EXAMPLE 2: 5 + 6 + 7
#
(+ 5 6 7)
```

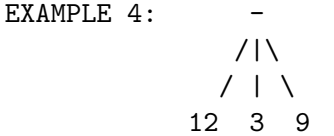
```
# EXAMPLE 3: 5 + 6 + 7 * 8
#
(+ 5 6 (* 7 8))

# EXAMPLE 4: 12 - 3 - 9
#
(- 12 3 9)

# EXAMPLE 5: 18 - 14 / 2
#
(- 18 (/ 14 2))
```

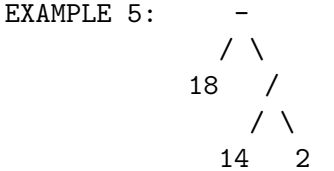
The five examples shown above have corresonding "parse trees" shown below.











For the above parse tree diagrams, your program must use `fork()` to create child processes that match these trees. Here, the edges between nodes are pipes in which each child process conveys the result of its intermediate calculation.

A simple algorithm here is to fork a child process for each operand encountered. In the `(* 7 8)` example, the parent process calls `fork()` twice, i.e., for the `7` and `8` operands.

In the `(- 18 (/ 14 2))` example, the parent process also calls `fork()` twice, this time for the `18` and `(/ 14 2)` operands. The second child process then calls `fork()` twice, i.e., for the `14` and `2` operands.

Note that you can assume all values given and all values calculated will be integers. Therefore, use integer division (i.e., truncate any digits after the decimal point).

Required Output

When you execute your program, each parent process must display a message when it parses an operator (i.e., starts an operation). Further, each child process must display a message when it sends an intermediate result back to its parent via a pipe.

The top-level parent process must display the final answer for the given expression.

For the example `(+ 5 6 (* 7 8))` expression, your program must display the following (though note that process IDs will likely be different and the order of some lines of output could be different, too):

```
PROCESS 31091: Starting '+' operation
PROCESS 31092: Sending '5' on pipe to parent
PROCESS 31093: Sending '6' on pipe to parent
PROCESS 31094: Starting '*' operation
PROCESS 31095: Sending '7' on pipe to parent
PROCESS 31096: Sending '8' on pipe to parent
PROCESS 31094: Sending '56' on pipe to parent
PROCESS 31091: Final answer is '67'
```

As with previous homeworks, do not include any other debugging statements, though you should be using `#ifdef DEBUG_MODE` to organize your debugging code. See the `main-with-debug.c` example on the course website.

Handling Errors

Your program must ensure that the correct number of command-line arguments are included. If not, display an error message and usage information as follows:

```
ERROR: Invalid arguments
USAGE: ./a.out <input-file>
```

If system calls fail, use `perror()` to display the appropriate error message, then exit the program by returning `EXIT_FAILURE`.

If given an invalid expression, display an error message using the format shown below. For example, given `(* 3)`, display the following:

```
PROCESS 6431: Starting '*' operation
PROCESS 6431: ERROR: not enough operands
```

As another example error, given `(QRST 5 6)`, display the following:

```
PROCESS 7330: ERROR: unknown 'QRST' operator
```

In both of the above cases, your program should exit.

Upon return from `wait()`, if the child process terminated due to a signal, display the following error message:

```
PARENT: child <child-pid> terminated abnormally
```

If the child process terminated normally, but its exit status was not `0`, display the following error message:

```
PARENT: child <child-pid> terminated with nonzero exit status <exit-status>
```

Submission Instructions

To submit your homework, please create a single compressed and zipped file (using `tar` and `gzip`). Please include only source and documentation files (i.e., do not include executables or binary files!).

To package up your submission, use `tar` and `gzip` to create a compressed `tar` file using your RCS userid, as in `goldsd.tar.gz`, that contains your source files (e.g., `main.c` and `file2.c`); include a `readme.txt` file only if necessary.

Here's an example showing how to create this file:

```
bash$ tar cvf goldsd.tar main.c file2.c readme.txt
main.c
file2.c
readme.txt
bash$ gzip -9 goldsd.tar
```

Submit the resulting `goldsd.tar.gz` file via the corresponding homework submission link available in LMS (`http://lms9.rpi.edu`). The link is in the Assignments section.