Programming Assignment 1: Shell
Design Document

Benjamin Ross
bpross@ucsc.edu
CMPS 111, Spring 2012

1 Goals

The goal of this project is to implement a basic shell program for the MINIX3 operating system. Secondary goals are to introduce programming in the MINIX3 environment and to introduce system calls to the programmer.

2 Available Resources

The only available resources for this project are system calls; they inlcude: exit(), fork(), execvp(), wait(), close(), dup(), and pipe(). No starter code was given for this assignment and all other functions will be written by the programmer.

3 Design

The basic design is to implement a shell that performs basic actions, which are typical of a shell program. These actions are: exit, a command with no arguments, command with arguments, output redirection, input redirection and piping of output. The shell will be designed to handle at least 20 commands on a single line, with each command having up to 50 arguments. The total size of one line will not exceed 1024 characters. Any lines given to the shell that are longer than 1024 will be returned as a syntax error to the user and will not be run.

3.1 Design of each function

```
void parse(char *line,  char **args)
   check to see if line is terminated in \n
      switch to \0 if it is
   while NOT End of Line
      replace white space with null chars
      place char into *args
      increment line
    end args with \0



void execute(char **args)
   fork process
      -check to see if fork was successful
    make sure pid == 0
       run execvp and check to see if it returns errors
     wait for child process to stop
```

```
void execute_re_out(char **args, char *file)
   open outfile
   make sure file was opened correctly
   fork process and check to make sure it opened correctly
   close stdout and dup with outfile
   close outfile
   run execvp and check to see if it returns errors
   wait for child process to stop

void execute_re_in(char **args, char *file)
   open infile
   check to see if file was opened correctly
   fork process and check to see if it was successful
   close stdin and dup with infile
   close infile
   run execvp and check to see if it returns errors
   wait for child process to stop

void execute_re_pipe(char **args_1, char **args_2)
   open pipe file descriptors
   fork the child process and check to see if successful
   dup pipe input to stdin
   execute second command
   dup pipe output to stdout
   execute first command

void check_for_meta(char **args)
   for arguments in args
      if > found send to execute_re_out
      if < found send to execute_re_in
      if | found send to execute_re_pipe

void main
    while(1)
      print prompt
      use fgets to get input line
      send line to parse
      check if exit is argument 0
       send to check_for_meta
       if no meta execute
```

4 Testing

For testing various commands will be tried and executed with the shell. Testing should be done to make sure the shell does not exit when it is not supposed to and that it will handle all of the required inputs.