Project 1
Implement a Unix Shell with History Feature

Objectives:

1. Get a taste of how a simple shell works.
2. Solidify understanding of systems calls, such as **`fork`**, **`read`**, **`wait`**, **`execvp`**, and etc.
3. Understand signal handling mechanisms under Linux/Unix.

Descriptions:

This project needs you to modify a C program named **`SimShell.c`** which serves as a shell interface that accepts user commands and then executes each command in a separate process. The program provides the basic operations of a command line shell as described below. The shell interface provides the user a prompt after which the next command is entered. The examples below illustrate the prompt **`command>`** and the user's commands.

`command>ls` /* this will display the files and directories under current directory*/
`command>cat hello.cpp` /* this will display content of the file hello.cpp */
`command>ctrl-c` /* ctrl-c will display the 10 most recent commands you have run*/
`command>cat hello.cpp &` /* run the command in background */

The program provides the basic operations of a command line shell, as outlined below. The setup() function reads in user's next command (up to 80 characters), and then parses it into separate tokens that are used to fill the argument vector for the command to be executed. If the command is to be run in the background, it will end with '&', and setup() will update the parameter **`background`** accordingly. This shell program terminates when user enters ctrl-d and setup() then invokes exit().

So, the first part of this project is to implement the setup(). Then, modify main() so that upon returning from setup(), a child process is forked and executes the command specified by user. The child process will call execvp() to execute the command. The parent process will decide to wait for the child process or not upon the value of parameter **background**.

```
#include <stdio.h>
#include <unistd.h>

#define MAX_LINE 80 /* 80 chars per line, per command, should be
enough. */

/*
 * setup() reads in the next command line, separating it into distinct
 * tokens using whitespace as delimiters. setup() sets the args
 * parameter as a null-terminated string.
 */

void setup(char inputBuffer[], char *args[],int *background)
{
   // you need to implement the function
}


int main(void)
{
    char inputBuffer[MAX_LINE]; // buffer to hold the command entered
    int background;     //equals 1 if a command is followed by '&'
    char *args[MAX_LINE/2 + 1];  /* command line (of 80) has max of 40
                                    arguments */

    while (1){     //Program terminates normally inside setup
        background = 0;
        printf(" COMMAND->\n");
        //setup() calls exit() when ctrl-d is entered
        setup(inputBuffer,args,&background);    //get next command

        /* the steps are:
        (1) fork a child process using fork()
        (2) the child process will invoke execvp()
        (3) the parent waits or returns to the setup() function,
            depending on the value of background . */
    }
}
```

You are also required to provide a history feature for this shell program. Assume the shell program can hold up to 10 most recent commands. When the user hits ctrl-c, the shell program will display the (up to 10) most recent commands before quitting the system. Ctrl-c is the SIGINT signal by which the system notifies the process that the SIGINT event has occurs. So, you need to handle this signal, in order to print out the most recent commands and exit the system. Below is a piece of program to demonstrate how to deal with a signal.

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

#define BUFFER_SIZE 50

static char buffer[BUFFER_SIZE];

/* the signal handler function */
void handle_SIGINT() {
   write(STDOUT_FILENO,buffer,strlen(buffer));
   exit(0);
}

int main(int argc, char *argv[])
{
    /* set up the signal handler */
    struct sigaction handler;
    handler.sa_handler = handle_SIGINT;
    sigaction(SIGINT, &handler, NULL);

    strcpy(buffer,"Caught <ctrl><c>\n");

    /* wait for <control> <C> */
    while (1);

    return 0;
}
```

To summarize, you are required to implement a shell program that reads in a user's commands and then executes the commands. When the user hits ctrl-d, the shell program should be terminated. When the user hits ctrl-c, the shell program shows the most recent commands in the buffer with the max size of 10 before termination.

Suggested Methodology:

Step 1: implement the setup().
Step 2: modify main() to use the child process to execute the command.
Step 3: add the history feature.

Useful Techniques:

1. If you want to know the specification of a system call such as "fork", you can always use the command: **man fork**

2. When you add the history feature for holding the recent commands, you can use a circular buffer to do that.

Submission:

1. All source code files
2. A readme file that briefly describes each file, and how to run the program
3. A Makefile file
4. Use **tar** to pack all files above into a package named **project1.tar**
5. Due date: 10/14/2008, Tuesday, 1:30PM
6. Submission command: **/home/fac/zhuy/fall08/SubmitHW340 p1 project1.tar**

Grading Policy:

1. If you do not have the files specified above, it will result in a score of zero.
2. If the program cannot be compiled and run, it will result in a score of zero.
3. Collaboration will result in a score of zero for all the students involved.
4. Total points are 20.