**Codes of shellA.c**

/\* Bobo Shi \*/

/\*

type 'gcc shellA.c -o shA' to compile

type './shA' to run program.

Test cases are in testA.txt file.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <string.h>

#define MAX\_LINE 80 /\* 80 chars per line, per command, should be enough. \*/

#define HIS\_SIZE 5

/\*\*

\* 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)

{

int length, /\* # of characters in the command line \*/

i, /\* loop index for accessing inputBuffer array \*/

start, /\* index where beginning of next command parameter is \*/

ct; /\* index of where to place the next parameter into args[] \*/

ct = 0;

/\* read what the user enters on the command line \*/

length = read(STDIN\_FILENO, inputBuffer, MAX\_LINE);

start = -1;

if (length == 0)

exit(0); /\* ^d was entered, end of user command stream \*/

if (length < 0){

perror("error reading the command");

exit(-1); /\* terminate with error code of -1 \*/

}

/\* examine every character in the inputBuffer \*/

for (i = 0; i < length; i++) {

switch (inputBuffer[i]){

case ' ':

case '\t' : /\* argument separators \*/

if(start != -1){

args[ct] = &inputBuffer[start]; /\* set up pointer \*/

ct++;

}

inputBuffer[i] = '\0'; /\* add a null char; make a C string \*/

start = -1;

break;

case '\n': /\* should be the final char examined \*/

if (start != -1){

args[ct] = &inputBuffer[start];

ct++;

}

inputBuffer[i] = '\0';

args[ct] = NULL; /\* no more arguments to this command \*/

break;

case '&':

\*background = 1;

inputBuffer[i] = '\0';

break;

default : /\* some other character \*/

if (start == -1)

start = i;

}

}

args[ct] = NULL; /\* just in case the input line was > 80 \*/

}

/\*void add\_command\_to\_history( const char \*command )

{

}\*/

struct history{

char buffer[MAX\_LINE];

char \*args[MAX\_LINE/2+1];

int count;

int background;

};

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 \*/

int i,j,rr\_flag; //bobo add

pid\_t pid;

int status;

char \*runargs[MAX\_LINE/2+1];

int run\_bg;

struct history his[HIS\_SIZE+1];

int command\_count=0;

char temp[MAX\_LINE];

int rr\_num, rr\_i;

//printf("sizeof= %d", sizeof(his[0].args));

for(i=0; i<HIS\_SIZE+1; i++){

for(j=0; j<MAX\_LINE/2+1; j++){

// printf("%d %d\n", i, j);

his[i].args[j]=NULL;

}

his[i].count=0;

}

while (1){ /\* Program terminates normally inside setup \*/

for(i=0; i<MAX\_LINE/2+1; i++){

runargs[i]=NULL;

}

run\_bg=0;

background = 0;

printf("SystemsIIShell->");

fflush(0);

setup(inputBuffer, args, &background); /\* get next command \*/

if(args[0]==NULL) continue;

// if it is 'rr' or 'r num'

if (strcmp(args[0], "rr") == 0){

if (his[HIS\_SIZE].count<1) {

printf("No recent command\n");

break;

}

else{

j=0;

while(his[HIS\_SIZE].args[j]){

if(his[HIS\_SIZE].args[j]){

strcpy(temp,his[HIS\_SIZE].args[j]);

runargs[j]=malloc(sizeof(temp));

strcpy(runargs[j],temp);

printf("%s ",runargs[j]);

}

j++;

}

printf("\n");

run\_bg=his[HIS\_SIZE].background;

}

}

else if (strcmp(args[0], "r")==0){

//rr\_num = atoi(args[1]);

//if(rr\_num = atoi(args[1]))

if(args[1]){

if(rr\_num= atoi(args[1])){

rr\_flag=0;

for (i=0; i<HIS\_SIZE+1; i++){

if(his[i].count==rr\_num){

j=0;

while(his[i].args[j]){

if(his[i].args[j]){

strcpy(temp,his[i].args[j]);

runargs[j]=malloc(sizeof(temp));

strcpy(runargs[j],temp);

// runargs[j]=his[i].args[j];

printf("%s ",runargs[j]);

}

j++;

}

printf("\n");

run\_bg=his[i].background;

rr\_flag=1;

break;

}

}

if (rr\_flag==0){

printf("the num you indicate is not in history\n");

j=0;

while(args[j]){

if(args[j])

runargs[j]=args[j];

j++;

}

run\_bg=background;

}

}

else {

printf("%s is not a num!\n", args[1]);

j=0;

while(args[j]){

if(args[j])

runargs[j]=args[j];

j++;

}

run\_bg=background;

}

}

else{

printf("no num!\n");

j=0;

while(args[j]){

if(args[j])

runargs[j]=args[j];

j++;

}

run\_bg=background;

}

}

else{

j=0;

while(args[j]){

if(args[j])

runargs[j]=args[j];

j++;

}

run\_bg=background;

}

if(strcmp(runargs[0],"h") == 0 || strcmp(runargs[0],"history")==0){

if (command\_count<5)

for (i=HIS\_SIZE-command\_count+1; i<HIS\_SIZE+1; i++){

if ( his[i].count > 0 && his[i].buffer != NULL && his[i].args[0] != NULL){

printf("%7d ", his[i].count);

j=0;

while(his[i].args[j]){

if(his[i].args[j])

printf(" %s", his[i].args[j]);

j++;

}

if(his[i].background==1)

printf(" &");

printf("\n");

}

}

else{

for (i=1; i<HIS\_SIZE+1; i++)

{

printf("%7d ", his[i].count);

j=0;

while(his[i].args[j]){

if(his[i].args[j]){

printf(" %s", his[i].args[j]);

}

j++;

}

if(his[i].background==1)

printf(" &");

printf("\n");

}

}

}

else{

pid = fork(); /\* fork a child \*/

if (pid < 0){ /\* error occurred \*/

fprintf(stderr, "Fork Failed\n");

return 1;

}

else if (pid == 0){ /\* child process\*/

if (execvp (\*runargs, runargs) < 0){ /\* if wrong command \*/

printf("\*\*\* ERROR: exec failed\n");

exit(1);

}

}

else { /\* parent process \*/

if (run\_bg==0) /\* wait if '&' \*/

while (wait(&status) != pid);

}

}

// shift value in his one by one

command\_count++;

for (i=0; i<HIS\_SIZE; i++){

if(command\_count+i > HIS\_SIZE+1){

strcpy(his[i].buffer,"");

j=0;

while(his[i].args[j]){

if(his[i].args[j]){

free(his[i].args[j]);

his[i].args[j]=NULL;

}

j++;

}

his[i].background=0;

}

if(command\_count+i > HIS\_SIZE){

j=0;

while(his[i+1].args[j]){

if(his[i+1].args[j]){

strcpy(temp,his[i+1].args[j]);

his[i].args[j] = malloc(sizeof(temp));

strcpy(his[i].args[j],temp);

}

j++;

}

strcpy(his[i].buffer,his[i+1].buffer);

his[i].background=his[i+1].background;

his[i].count=his[i+1].count;

}

}

//store command to his[HIS\_SIZE]

if(command\_count>1){

i=0;

while(his[HIS\_SIZE].args[i]){

//printf("free last: his[HIS\_SIZE].args[%d]=%s\n",i,his[HIS\_SIZE].args[i]);

if(his[HIS\_SIZE].args[i]){

free(his[HIS\_SIZE].args[i]);

his[HIS\_SIZE].args[i]=NULL;

}

i++;

}

his[HIS\_SIZE].background=0;

strcpy(his[HIS\_SIZE].buffer,"");

}

i=0;

while(runargs[i]){

if(runargs[i]){

strcpy(temp,runargs[i]);

his[HIS\_SIZE].args[i]=malloc(sizeof(temp));

strcpy(his[HIS\_SIZE].args[i],temp);

}

i++;

}

his[HIS\_SIZE].count=command\_count;

his[HIS\_SIZE].background=run\_bg;

i=0;

while(runargs[i]){

if(runargs[i]){

// free(runargs[i]);

runargs[i]=NULL;

}

i++;

}

run\_bg=0;

}

}

**Test cases for shellA.c**

//Bobo Shi

//test cases for shellA.c in Lab2

//"gcc -g shellA.c -o shA" to compile

//"./shA" to execute

testcases:

SystemsIIShell->ls

a.out shA shellA.c shellA-old2.c shellB.c shell.c testB.txt

history.dat shB shellA.c~ shellA-old.c shellB.c~ testA.txt

SystemsIIShell->grep bobo shellA.c

int i,j,rr\_flag; //bobo add

SystemsIIShell->h

1 ls

2 grep bobo shellA.c

SystemsIIShell->pwd &

SystemsIIShell->/home/0/shib/Lab2

whoami

shib

SystemsIIShell->history

1 ls

2 grep bobo shellA.c

3 h

4 pwd &

5 whoami

SystemsIIShell->r 1 /\* 1st in history is ls \*/

ls

a.out shA shellA.c shellA-old2.c shellB.c shell.c testB.txt

history.dat shB shellA.c~ shellA-old.c shellB.c~ testA.txt

SystemsIIShell->r 5 /\* 5th in history is whoami \*/

whoami

shib

SystemsIIShell->head shellA.c

/\* Bobo Shi \*/

/\*

type 'gcc shellA.c -o shA' to compile

type './shA' to run program.

Test cases are in testA.txt file.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

SystemsIIShell->ls -lh

total 160K

-rwx------ 1 shib class 13K Jun 27 15:17 a.out

-rw------- 1 shib class 60 Jun 27 15:17 history.dat

-rwx------ 1 shib class 17K Jun 27 16:57 shA

-rwx------ 1 shib class 18K Jun 27 12:37 shB

-rw------- 1 shib class 7.6K Jun 27 16:56 shellA.c

-rw------- 1 shib class 7.1K Jun 27 00:24 shellA.c~

-rw------- 1 shib class 7.1K Jun 27 16:56 shellA-old2.c

-rw------- 1 shib class 6.8K Jun 26 16:25 shellA-old.c

-rw------- 1 shib class 8.3K Jun 27 00:24 shellB.c

-rw------- 1 shib class 8.3K Jun 26 23:51 shellB.c~

-rw------- 1 shib class 3.5K Jun 24 13:12 shell.c

-rw------- 1 shib class 3.5K Jun 27 12:40 testA.txt

-rw------- 1 shib class 3.3K Jun 27 12:40 testB.txt

SystemsIIShell->echo bobo is smart

bobo is smart

SystemsIIShell->rr /\* rr: run most recent \*/

echo bobo is smart

bobo is smart

SystemsIIShell->history

8 whoami

9 head shellA.c

10 ls -lh

11 echo bobo is smart

12 echo bobo is smart

SystemsIIShell->r 10

ls -lh

total 160K

-rwx------ 1 shib class 13K Jun 27 15:17 a.out

-rw------- 1 shib class 60 Jun 27 15:17 history.dat

-rwx------ 1 shib class 17K Jun 27 16:57 shA

-rwx------ 1 shib class 18K Jun 27 12:37 shB

-rw------- 1 shib class 7.6K Jun 27 16:56 shellA.c

-rw------- 1 shib class 7.1K Jun 27 00:24 shellA.c~

-rw------- 1 shib class 7.1K Jun 27 16:56 shellA-old2.c

-rw------- 1 shib class 6.8K Jun 26 16:25 shellA-old.c

-rw------- 1 shib class 8.3K Jun 27 00:24 shellB.c

-rw------- 1 shib class 8.3K Jun 26 23:51 shellB.c~

-rw------- 1 shib class 3.5K Jun 24 13:12 shell.c

-rw------- 1 shib class 3.5K Jun 27 12:40 testA.txt

-rw------- 1 shib class 3.3K Jun 27 12:40 testB.txt

SystemsIIShell->h

10 ls -lh

11 echo bobo is smart

12 echo bobo is smart

13 history

14 ls -lh

SystemsIIShell->r 13 /\* r 13: history. This tests history command in history \*/

history

11 echo bobo is smart

12 echo bobo is smart

13 history

14 ls -lh

15 h

SystemsIIShell->ps -l

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

0 R 33971 1028 31760 0 80 0 - 27026 - pts/18 00:00:00 ps

0 S 33971 4372 4371 0 80 0 - 27746 rt\_sig pts/18 00:00:00 tcsh

0 S 33971 31760 4372 0 80 0 - 1014 wait pts/18 00:00:00 shA

SystemsIIShell->./shA

SystemsIIShell->ls

a.out shA shellA.c shellA-old2.c shellB.c shell.c testB.txt

history.dat shB shellA.c~ shellA-old.c shellB.c~ testA.txt

SystemsIIShell->badcommand /\* try wrong command, should not be executed \*/

\*\*\* ERROR: exec failed

SystemsIIShell->history

1 ls

2 badcommand

SystemsIIShell->date

Thu Jun 27 16:59:40 EDT 2013

/\* here I press 'Ctrl+d' to go back into my first 'shA' shell \*/

SystemsIIShell->SystemsIIShell->

SystemsIIShell->h

14 ls -lh

15 h

16 history

17 ps -l

18 ./shA

SystemsIIShell->which gcc

/usr/bin/gcc

SystemsIIShell->du -lh

168K .

SystemsIIShell->h

17 ps -l

18 ./shA

19 h

20 which gcc

21 du -lh

// begin to test some bad commands

SystemsIIShell->r 2 /\*test r num, num is out of history \*/

the num you indicate is not in history

\*\*\* ERROR: exec failed

SystemsIIShell->r g /\* test r num, num is not a number \*/

g is not a num!

\*\*\* ERROR: exec failed

SystemsIIShell->r /\* test r with no rum following \*/

no num!

\*\*\* ERROR: exec failed

SystemsIIShell->history

21 du -lh

22 h

23 r 2

24 r g

25 r

SystemsIIShell->echo have a good day !

have a good day !

//press 'Ctrl+d' to exit shA

**Codes of shellB.c**

/\* Bobo Shi \*/

/\*

type 'gcc shellB.c -o shB' to compile

type './shB' to run program.

Test cases are in testB.txt file.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <string.h>

#define MAX\_LINE 80 /\* 80 chars per line, per command, should be enough. \*/

#define HIS\_SIZE 5

/\*\*

\* 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)

{

int length, /\* # of characters in the command line \*/

i, /\* loop index for accessing inputBuffer array \*/

start, /\* index where beginning of next command parameter is \*/

ct; /\* index of where to place the next parameter into args[] \*/

ct = 0;

/\* read what the user enters on the command line \*/

length = read(STDIN\_FILENO, inputBuffer, MAX\_LINE);

start = -1;

if (length == 0)

exit(0); /\* ^d was entered, end of user command stream \*/

if (length < 0){

perror("error reading the command");

exit(-1); /\* terminate with error code of -1 \*/

}

/\* examine every character in the inputBuffer \*/

for (i = 0; i < length; i++) {

switch (inputBuffer[i]){

case ' ':

case '\t' : /\* argument separators \*/

if(start != -1){

args[ct] = &inputBuffer[start]; /\* set up pointer \*/

ct++;

}

inputBuffer[i] = '\0'; /\* add a null char; make a C string \*/

start = -1;

break;

case '\n': /\* should be the final char examined \*/

if (start != -1){

args[ct] = &inputBuffer[start];

ct++;

}

inputBuffer[i] = '\0';

args[ct] = NULL; /\* no more arguments to this command \*/

break;

case '&':

\*background = 1;

inputBuffer[i] = '\0';

break;

default : /\* some other character \*/

if (start == -1)

start = i;

}

}

args[ct] = NULL; /\* just in case the input line was > 80 \*/

}

/\*void add\_command\_to\_history( const char \*command )

{

}\*/

struct history{

char buffer[MAX\_LINE];

char \*args[MAX\_LINE/2+1];

int count;

int background;

};

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 \*/

int i,j,k,rr\_flag; //bobo add

pid\_t pid;

int status;

char \*runargs[MAX\_LINE/2+1];

int run\_bg;

struct history his[HIS\_SIZE+1];

int command\_count=0;

char temp[MAX\_LINE];

int rr\_num, his\_num;

FILE \*hisfile;

int his\_background, his\_count;

int i\_temp;

//check if "history.dat" file exist.

//if yes, read it.

for(i=0; i<HIS\_SIZE+1; i++){

for(j=0; j<MAX\_LINE/2+1; j++){

// printf("%d %d\n", i, j);

his[i].args[j]=NULL;

}

his[i].count=0;

}

for(i=0; i<MAX\_LINE/2+1; i++){

runargs[i]=NULL;

}

run\_bg=0;

hisfile = fopen("history.dat","r");

his\_num=0;

if (hisfile != NULL){

if(fscanf(hisfile,"%d", &his\_num) <=0 ) {

printf("wrong history.dat file");

exit(0);

}

for (i=0; i<his\_num; i++){

if(fscanf(hisfile,"%d %d", &his\_background, &his\_count) <=0)

break;

else{

for (j=0; j<his\_count; j++){

fscanf(hisfile,"%s", temp);

his[HIS\_SIZE-his\_num+i+1].args[j]=malloc(sizeof(temp));

strcpy(his[HIS\_SIZE-his\_num+i+1].args[j],temp);

}

his[HIS\_SIZE-his\_num+i+1].background=his\_background;

his[HIS\_SIZE-his\_num+i+1].count=i+1;

}

}

}

if(hisfile) fclose(hisfile);

//printf("sizeof= %d", sizeof(his[0].args));

while (1){ /\* Program terminates normally inside setup \*/

background = 0;

printf("SystemsIIShell->");

fflush(0);

setup(inputBuffer, args, &background); /\* get next command \*/

if(args[0]==NULL) continue;//

// if it is 'rr' or 'r num'

if (strcmp(args[0], "rr") == 0){

if (his[HIS\_SIZE].count<1) {

printf("No recent command\n");

break;

}

else{

j=0;

while(his[HIS\_SIZE].args[j]){

if(his[HIS\_SIZE].args[j]){

strcpy(temp,his[HIS\_SIZE].args[j]);

runargs[j]=malloc(sizeof(temp));

strcpy(runargs[j],temp);

printf("%s ", runargs[j]);

}

j++;

}

printf("\n");

run\_bg=his[HIS\_SIZE].background;

}

}

else if (strcmp(args[0], "r")==0){

if(args[1]){

if(rr\_num=atoi(args[1]))

{

rr\_flag=0;

for (i=0; i<HIS\_SIZE+1; i++){

if(his[i].count==rr\_num){

j=0;

while(his[i].args[j]){

if(his[i].args[j]){

strcpy(temp,his[i].args[j]);

runargs[j]=malloc(sizeof(temp));

strcpy(runargs[j],temp);

printf("%s ",runargs[j]);

}

j++;

}

printf("\n");

run\_bg=his[i].background;

rr\_flag=1;

break;

}

}

if (rr\_flag==0){

printf("the num you indicate is not in history\n");

j=0;

while(args[j]){

if(args[j])

runargs[j]=args[j];

j++;

}

run\_bg=background;

}

}

else {

printf("%s is not a num!\n", args[1]);

j=0;

while(args[j]){

if(args[j])

runargs[j]=args[j];

j++;

}

run\_bg=background;

}

}

else{

printf("no num!\n");

j=0;

while(args[j]){

if(args[j])

runargs[j]=args[j];

j++;

}

run\_bg=background;

}

}

else{

j=0;

while(args[j]){

if(args[j])

runargs[j]=args[j];

j++;

}

run\_bg=background;

}

if(strcmp(runargs[0],"h") == 0 || strcmp(runargs[0],"history")==0){

if (command\_count+his\_num<6)

for (i=HIS\_SIZE-command\_count-his\_num+1; i<HIS\_SIZE+1; i++){

if ( his[i].count > 0 && his[i].buffer != NULL && his[i].args[0] != NULL){

printf("%7d ", his[i].count);

j=0;

while(his[i].args[j]){

if(his[i].args[j])

printf(" %s", his[i].args[j]);

j++;

}

if(his[i].background==1)

printf(" &");

printf("\n");

}

}

else{

for (i=1; i<HIS\_SIZE+1; i++)

{

printf("%7d ", his[i].count);

j=0;

while(his[i].args[j]){

if(his[i].args[j]){

printf(" %s", his[i].args[j]);

}

j++;

}

if(his[i].background==1)

printf(" &");

printf("\n");

}

}

}

else{

pid = fork(); /\* fork a child \*/

if (pid < 0){ /\* error occurred \*/

fprintf(stderr, "Fork Failed\n");

return 1;

}

else if (pid == 0){ /\* child process\*/

if (execvp (\*runargs, runargs) < 0){ /\* if wrong command \*/

printf("\*\*\* ERROR: exec failed\n");

exit(1);

}

}

else { /\* parent process \*/

if (run\_bg==0) /\* wait if '&' \*/

while (wait(&status) != pid);

}

}

// shift value in his one by one

command\_count++;

for (i=0; i<HIS\_SIZE; i++){

if(command\_count+i+his\_num > HIS\_SIZE+1){

strcpy(his[i].buffer,"");

j=0;

while(his[i].args[j]){

if(his[i].args[j]){

free(his[i].args[j]);

his[i].args[j]=NULL;

}

j++;

}

}

if(command\_count+i+his\_num > HIS\_SIZE){

j=0;

while(his[i+1].args[j]){

if(his[i+1].args[j]){

strcpy(temp,his[i+1].args[j]);

his[i].args[j] = malloc(sizeof(temp));

strcpy(his[i].args[j],temp);

}

j++;

}

strcpy(his[i].buffer,his[i+1].buffer);

his[i].background=his[i+1].background;

his[i].count=his[i+1].count;

}

}

//store command to his[HIS\_SIZE]

if(command\_count+his\_num>1){

i=0;

while(his[HIS\_SIZE].args[i]){

//printf("free last: his[HIS\_SIZE].args[%d]=%s\n",i,his[HIS\_SIZE].args[i]);

if(his[HIS\_SIZE].args[i]){

free(his[HIS\_SIZE].args[i]);

his[HIS\_SIZE].args[i]=NULL;

}

i++;

}

strcpy(his[HIS\_SIZE].buffer,"");

}

i=0;

while(runargs[i]){

if(runargs[i]){

strcpy(temp,runargs[i]);

his[HIS\_SIZE].args[i]=malloc(sizeof(temp));

strcpy(his[HIS\_SIZE].args[i],temp);

}

i++;

}

his[HIS\_SIZE].count=command\_count+his\_num;

his[HIS\_SIZE].background=background;

i=0;

while(runargs[i]){

runargs[i]=NULL;

i++;

}

run\_bg=0;

hisfile = fopen("history.dat","w");

if (his\_num+command\_count<6) j=his\_num+command\_count;

else j=6;

fprintf(hisfile,"%d\n", j);

for (i=0; i<j; i++){

i\_temp=0;

while(his[6-j+i].args[i\_temp]){

i\_temp++;

}

fprintf(hisfile,"%d %d\n", his[6-j+i].background, i\_temp);

for (k=0; k<i\_temp; k++){

fprintf(hisfile,"%s\n", his[6-j+i].args[k]);

}

}

fclose(hisfile);

}

}

**Test cases for shellB.c**

//Bobo Shi

//test cases for shellB.c in Lab2

//the history buffer is called 'history.dat'

//when there is no 'history.dat' file in current directory, shellB.c will creat a new one

//The format of history.dat is

//his\_num(number of commands)

//background i(number of args for 1st command)

//args[0]

//args[1]

//....

//background i(number of args for 2nd command)

//args[0]

//....

//background i(number of args for his\_num command)

//args[0]

//....

//....

//if 'history.dat' exists in current directory, then read it to his which contains the history of command

//type 'gcc -g shellB.c -o shB' to compile

//type './shB' to execute shell

//This the first time to run. There is no 'history.dat' file which contains the histotry of commands

testcase:

SystemsIIShell->ls

a.out shB shellA.c~ shellA-old.c shellB.c~ shell.c testB.txt

shA shellA.c shellA-old2.c shellB.c shellB-old.c testA.txt

SystemsIIShell->grep bobo shellB.c

int i,j,k,rr\_flag; //bobo add

SystemsIIShell->who am i

shib pts/18 2013-06-27 15:51 (dhcp-128-146-2-53.osuwireless.ohio-state.edu)

SystemsIIShell->h

1 ls

2 grep bobo shellB.c

3 who am i

SystemsIIShell->r 1

ls

a.out shA shellA.c shellA-old2.c shellB.c shellB-old.c testA.txt

history.dat shB shellA.c~ shellA-old.c shellB.c~ shell.c testB.txt

SystemsIIShell->pwd &

/home/0/shib/Lab2

SystemsIIShell->r 3

who am i

shib pts/18 2013-06-27 15:51 (dhcp-128-146-2-53.osuwireless.ohio-state.edu)

SystemsIIShell->rr

who am i

shib pts/18 2013-06-27 15:51 (dhcp-128-146-2-53.osuwireless.ohio-state.edu)

SystemsIIShell->h

4 h

5 ls

6 pwd &

7 who am i

8 who am i

SystemsIIShell->./shB

SystemsIIShell->h

2 ls

3 pwd &

4 who am i

5 who am i

6 h

SystemsIIShell->date

Thu Jun 27 17:31:27 EDT 2013

SystemsIIShell->badcommand

\*\*\* ERROR: exec failed

SystemsIIShell->history

5 who am i

6 h

7 h

8 date

9 badcommand

SystemsIIShell->rr

history

6 h

7 h

8 date

9 badcommand

10 history

SystemsIIShell->ps

PID TTY TIME CMD

4372 pts/18 00:00:00 tcsh

27939 pts/18 00:00:00 shB

28856 pts/18 00:00:00 shB

28866 pts/18 00:00:00 ps

SystemsIIShell->SystemsIIShell->

SystemsIIShell->echo 1

1

SystemsIIShell->echo 2

2

SystemsIIShell->echo 3

3

SystemsIIShell->echo 4

4

SystemsIIShell->echo 5

5

SystemsIIShell->echo 6

6

/\* here I press 'Ctrl+d' to exit the shB shell \*/

//type './shB' to start shB shell again. 'history.dat' file exits

SystemsIIShell->h

2 echo 2

3 echo 3

4 echo 4

5 echo 5

6 echo 6

SystemsIIShell->r 5

echo 5

5

SystemsIIShell->date

Thu Jun 27 17:33:23 EDT 2013

SystemsIIShell->rr

date

Thu Jun 27 17:33:28 EDT 2013

SystemsIIShell->ls

a.out shA shellA.c shellA-old2.c shellB.c shellB-old.c testA.txt

// 'Ctrl+d' to exit

SystemsIIShell->/home/0/shib/Lab2

// './shB' to start shell again

% ./shB

SystemsIIShell->rr

ls

a.out shA shellA.c shellA-old2.c shellB.c shellB-old.c testA.txt

history.dat shB shellA.c~ shellA-old.c shellB.c~ shell.c testB.txt

SystemsIIShell->history

3 echo 5

4 date

5 date

6 ls

7 ls

//'Ctrl+d' to exit

// begin to test some bad commands

// './shB' to begin shell

SystemsIIShell->h

2 date

3 date

4 ls

5 ls

6 history

SystemsIIShell->r 1 /\* test r num, where num is out of history \*/

the num you indicate is not in history

\*\*\* ERROR: exec failed

SystemsIIShell->r notnum /\* test r num, where 'num' is not a num \*/

notnum is not a num!

\*\*\* ERROR: exec failed

SystemsIIShell->r /\* test r without a num following \*/

no num!

\*\*\* ERROR: exec failed

// 'Ctrl+d' to exit