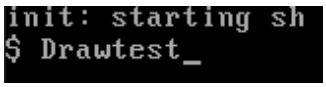
CS344: OPERATING SYSTEMS LAB

Group 13

Adarsh Kumar, 190101002 | **Upender Dahiya, 190101096**
Tanishq Katare, 190101093 | **Keshav Chourasiya, 190101045**

Task 1

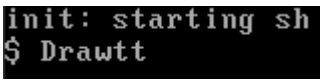
Output for Caret Navigation:



Pressing **RIGHT** arrow does nothing as we are currently at the end

Pressing **LEFT** arrow and adding ‘x’ in between, text appropriately shifted

Pressing **RIGHT** arrow and adding ‘x’ in between, text appropriately shifted unlike the 1st case



Using **BACKSPACE** in between should also be handled, which is what shown here

Output of Shell History:

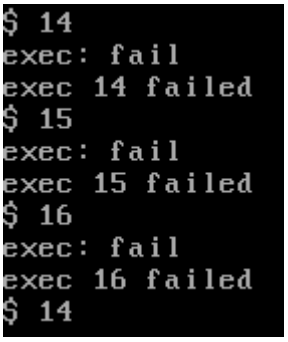
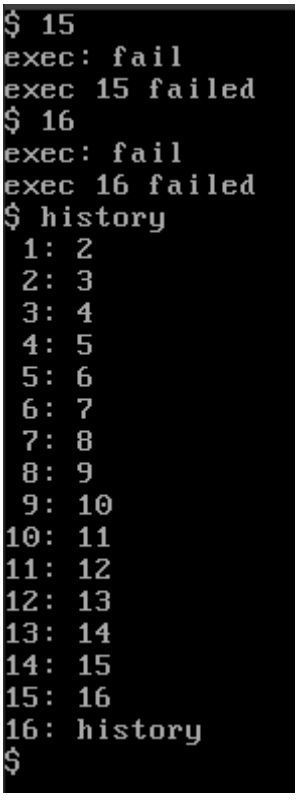


Image on left shows that, firstly we have executed some random 16 commands. These commands are now saved into the history buffer, the last command ‘14’ shown here is achieved using the **UP/DOWN** arrow keys



The image on right shows that history buffer can **maximum** store only **16 commands** hence the last history command upon running forces the command ‘1’ to be taken out from the buffer due to the limitation in size. Here shown are the currently stored 16 commands.

Files changed or created:
For task 1.1 - console.c, console.h
For task 1.2 - console.c, console.h, sh.c, and all files in pt.4 below

1. **console.c-** This file is responsible for every input and output text that is shown in console, to add the caret navigation we have to manipulate prewritten functions in it and also add some new functions for achieving the navigation.

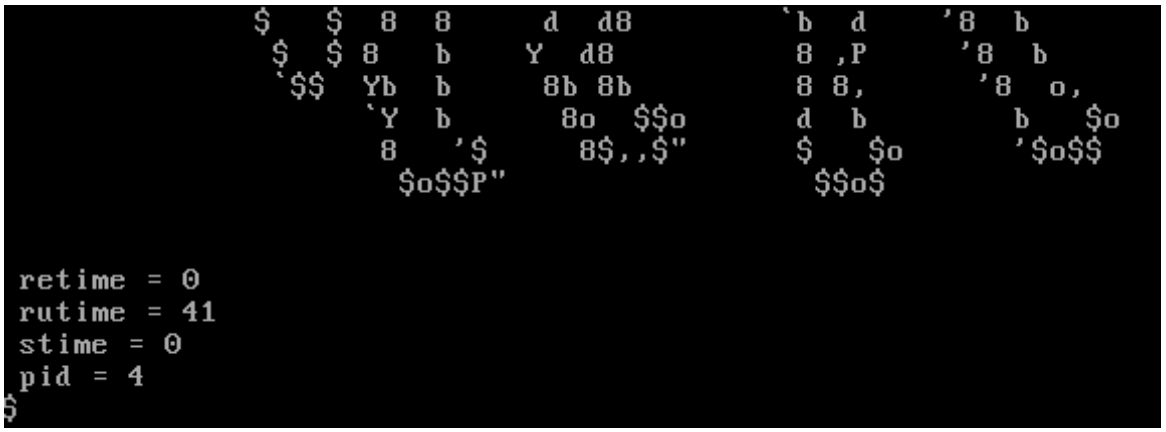
2. **console.h-** As said there are some new functions that has to be added, so this header files contains all the function declaration and also some macros used in console.c file
3. **sh.c-** This file has the 'history' command declaration and the system call that is needed to be invoked after hitting enter in console after giving history as the command.
4. **syscall.c, syscall.h, sysproc.c, user.h, usys.S-** These files are the same files that we changed in last assignment also, all the changes are done for creating a function definition of any system call, so for any new system call few changes have to be made, all these changes are done in these files only. The changes that one need to apply are :
    a. **user.h**: It contains the system call definitions of xv6. Have to add function declaration here for the new system calls.
    b. **usys.S**: It contains a list of system calls exported by the kernel. In this a global macro is defined called SYSCALL that inturn runs a small assembly code, which moves the mapped integer value to eax register and invokes an interrupt.
    c. **syscall.h**: It contains a mapping from system call name to system call number.
    d. **syscall.c**: It contains helper functions to connect to correct system call number to the actual system call implementations.
    e. **sysproc.c**: It contains the actual implementations of process related system calls.

Explanation of functions in console.c: (All references to functions and variables are taken from the attached codes provided with submission file)

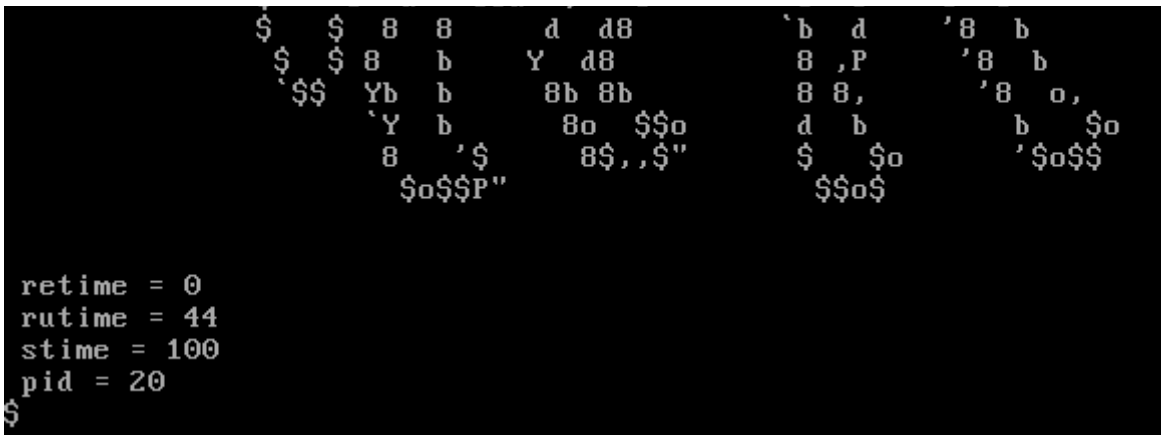
1. **void clearLineOnConsole(void)** :- Clears the current line on the console screen.
2. **void copyLineToBuf0(void)** :- Copies currently displayed line to Buf0, a temporary buffer created to store the command and store length on structure created.
3. **void eraseCurrentCommand()** :- Erase all content of current command.
4. **void printBufToConsole(char *bufToPrintOnScreen, uint length)** :- Prints the provided buffer as argument on console.
5. **void copyBufferToInputBuf(char *bufToSaveInInput, uint length)** :- Copying the given buffer to input.buf. (input is a defined structure that encapsulates some details of any command along with the command itself like position of start and end of command and caret location).
6. **void savingToHistoryTable()** :- Copying current command in input.buf to saved history table. Similar to input structure, another structure called historyBufferArray is defines that binds all the required nitty gritties for implementing history table.
7. **int history(char *buffer, int historyId)** :- Function that gets called by the sys_history system call. It checks for any erroneous cases and returns different error codes accordingly and stores the command to history table and returns 0 otherwise.
8. **void shiftbufright()** :- Shifting input.buf one position to right and printing back to console
9. **void shiftbufleft()** :- Shifting input.buf one positon to left and printing back to console, used when BACKSPACE is pressed in between position of the command
10. **static void cgaputc(int c)** :- Prints each character to the console.(Existed function)
11. **void consputc(int c)** :- Some functions used here are cgaputc() i.e. used for printing on QEMU console and uartputc() i.e. used for printing on Unix console. This function also updates the Unix console when something on QEMU console is typed.(Existed function)
12. **void consoleintr(int (*getc)(void))** :- Handles interrupts in consoles.(Existed function)

Task 2

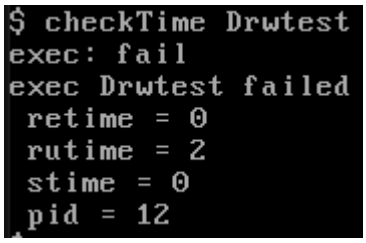
Output of user program that shows clock tick details of another user program when executed.



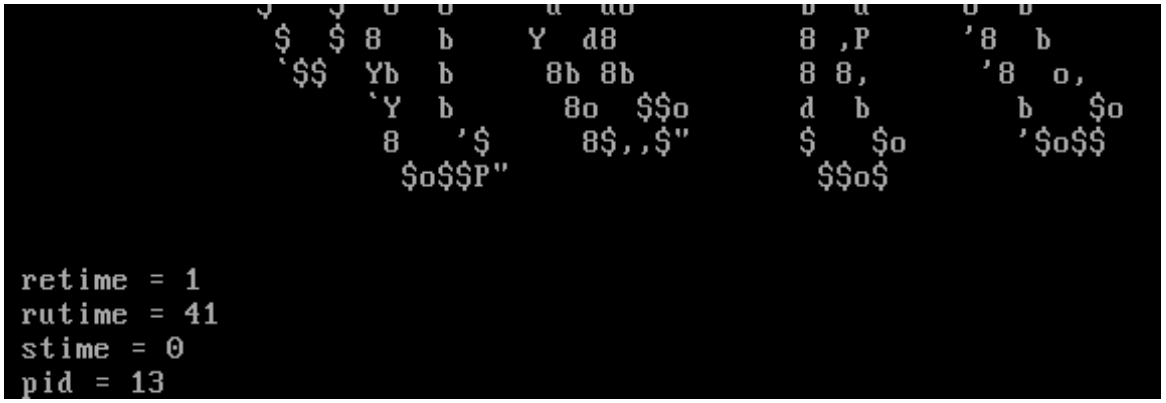
The user program that is executed here is the program that prints ascii image, implemented in last assignment. Upon executing we can see the it took 41 clock ticks for running the process. And also the extended wait2() function also provides the PID which is also shown here.



Now adding a line sleep(100); in the Drawtest.c file would temporarily suspend the program for exactly 100 clock ticks, therefore upon executing we can see the reflected stime as 100 here in the image.



Since Drwtest is not an user program hence the output shows execution failed



Here in this case, retime that shows clock ticks during which the process waited is 1

Files changed or created:

1. **proc.c :-** This contains the definition of the function wait2() and updateTimes.
2. **proc.h :-** The first step was to extend the current proc structure and add ctime(process creation time), stime(process SLEEPING time), retime(process READY(RUNNABLE) time), rutime(process RUNNING time) of a process. This is what done here.
3. **trap.c :-** Invoking the updateTimes() function is done in this file
4. **checkTime.c :-** Created a test program which utilises the wait2 system call by creating a checkTime like command for the same. This command executes the provided user program given as arguments normally, the only thing different is that it uses wait2() instead of normal wait(). At end it displays clock time ticks and pid.
5. **Makefile :-** Added checkTime as the user program list
6. **syscall.c, syscall.h, user.h, usys.S, sysproc.c :-** waitx() system call was added modifying these files, changes that need to be done are similar to that of the previous task.

Explanation of some functions:

1. **int wait2(int *retime, int *rutime, int *stime)** :- To extract the time ticks information from the kernel new function wait2() is used to extend wait(), that takes retime, rutime and stime integer pointer as the arguments
2. **void updateTimes()** :- When a new process gets created the kernel code should update all the time ticks that is achieved using updateTimes() function that increments the ticks on the basis of state of the process.
3. **int fork(void)** :- Creates a new process copying current process as the parent. Caller must set state of returned process to RUNNABLE.