CS 344
Operating Systems Lab
Assignment - 2A
Report

Team Members:

1. Braj Nandan Kalya - 190101029
2. Anurag Singhal - 190123010
3. Pallav Pandey - 190123044
4. Alokeveer Mondal - 190123003

How to use the patch file:

- Keep xv6 cloned repo and the submitted **patchfile.patch** in the same directory. (Preferably a new folder in desktop to avoid errors)
    - (Link to clone xv6: ***git clone git://github.com/mit-pdos/xv6-public.git***)
- Open this parent directory (that contains patch and xv6) in the terminal and run the following command-
    - ***patch -s -p0 < patchfile.patch***
- The xv6-public folder will now have all the required changes.

Task 1.1 Caret navigation

```
case LEFT_ARROW:
  if (input.e != input.w) {
    input.e--;
    consputc(c);
  }
  break;
case RIGHT_ARROW:
  if (input.e < input.rightmost) {
    consputc(input.buf[input.e % INPUT_BUF]);
    input.e++;
  }
  else if (input.e == input.rightmost){
    consputc(' ');
    consputc(LEFT_ARROW);
  }
  break;
```

consoleintr() is called whenever there is an interrupt.
We have handled Left Arrow and Right Arrow cases in the switch statement in **consoleintr().**

```
switch(c) {
  case '\n':
    pos += 80 - pos%80;
    break;
  case BACKSPACE:
    if(pos > 0) --pos;
    break;
  case LEFT_ARROW:
    if(pos > 0) --pos;
    break;
  default:
    crt[pos++] = (c&0xff) | 0x0700;  // black on white
}
```

pos variable stores the position of the cursor that is displayed on the qemu console. Cursor is brought to the beginning of the next line by adjusting the value of *pos* variable in the case of '\n'.

```
void
consputc(int c)
{
  if(panicked){
    cli();
    for(;;)
      ;
  }

  if(c == BACKSPACE){
    uartputc('\b'); uartputc(' '); uartputc('\b');
  } else if(c == LEFT_ARROW){
    uartputc('\b');
  } else {
    uartputc(c);
  }

  cgaputc(c);
}
```

consputc() is used to print characters to the qemu console and as well as linux termial. **consputc()** further calls **cgaputc()** which is used to print to the qemu kernel specifically. We have handled Left Arrow and Right Arrow cases in **cgaputc() and consputc().**

When the cursor is not present at the end of line, typing any character will require the suffix to be shifted right by 1 unit and backspace will require the suffix to be shifted left by 1 unit. Following functions are used for the same:

```
void shiftbufleft() {
  uint n = input.rightmost - input.e;
  uint i;
  consputc(LEFT_ARROW);
  input.e--;
  for (i = 0; i < n; i++) {
    char c = input.buf[(input.e + i + 1) % INPUT_BUF];
    input.buf[(input.e + i) % INPUT_BUF] = c;
    consputc(c);
  }
  input.rightmost--;
  consputc(' '); // delete the last char in line
  for (i = 0; i <= n; i++) {
    consputc(LEFT_ARROW); // shift the caret back to the left
  }
}
```

Shift input.buf one byte to the left, and repaint the chars on-screen. Used only when punching in BACKSPACE and the caret isn't at the start of the line.

```
void shiftbufright() {
  uint n = input.rightmost - input.e;
  int i;
  for (i = 0; i < n; i++) {

    char c = charsToBeMoved[i];
    input.buf[(input.e + i) % INPUT_BUF] = c;
    consputc(c);
  }
  // reset charsToBeMoved for future use
  memset(charsToBeMoved, '\0', INPUT_BUF);
  // return the caret to its correct position
  for (i = 0; i < n; i++) {
    consputc(LEFT_ARROW);
  }
}
```

Shift input.buf one byte to the right, and repaint the chars on-screen. Used only when punching in new keys and the caret isn't at the end of the line.

```
default:
  if(c != 0 && input.e-input.r < INPUT_BUF){
    c = (c == '\r') ? '\n' : c;
    if(c == '\n' || c == C('D') || input.rightmost == input.r + INPUT_BUF - 1){
      input.buf[input.rightmost++ % INPUT_BUF] = c;
      consputc(c);
      saveCommandInHistory();
      input.w = input.e = input.rightmost;
      wakeup(&input.r);
    }
    else {
      if (input.rightmost > input.e) { // caret isn't at the end of the line
        copyCharsToBeMoved();
        input.buf[input.e++ % INPUT_BUF] = c;
        input.rightmost++;
        consputc(c);
        shiftbufright();
      }
      else {
        input.buf[input.e++ % INPUT_BUF] = c;
        input.rightmost = input.e - input.rightmost == 1 ? input.e : input.rightmost;
        consputc(c);
      }
    }
  }
  break;
```

Typing in general has been handled in the default case inside **consoleintr()** as shown above. End line at any moment is resulting in moving to the next line no matter where the caret is and it's ascii value is entered at the end of the buffer.
When editing from the middle of a buffer text is shifted accordingly.

Task 1.2 Shell History Ring

```
struct {
  char bufferArr[MAX_HISTORY][INPUT_BUF];
  uint lengthsArr[MAX_HISTORY];
  uint lastCommandIndex;
  int numOfCommmandsInMem;
  int currentHistory;
} historyBufferArray;
```

// holds the actual command strings
// this will hold the length of each command string
// the index of the last command entered to history
// number of history commands in mem
// this will hold the current history view (the oldest will be MAX_HISTORY-1)

Following are some functions implemented to use command history (historyBufferArray.bufferArr) efficiently:

```
void
earaseCurrentLineOnScreen(void){
    uint numToEarase = input.rightmost - input.r;
    uint i,j;
    for(j = 0;j<input.rightmost-input.e;j++){
      consputc(RIGHT_ARROW);
    }
    for (i = 0; i < numToEarase; i++) {
      consputc(BACKSPACE);
    }
}
```

// stores the number of characters to erase

// shifts the cursor to the rightmost index

// erases all the characters

```
void
copyCharsToBeMovedToOldBuf(void){
    lengthOfOldBuf = input.rightmost - input.r;
    uint i;
    for (i = 0; i < lengthOfOldBuf; i++) {
        oldBuf[i] = input.buf[(input.r+i)%INPUT_BUF];
    }
}
```

// total number of char.

// storing characters on display in the old buffer.

```
void
earaseContentOnInputBuf(){
  input.rightmost = input.r;
  input.e = input.r;
}
```

// make rightmost as the starting index.
// make current as the starting index.

```
void
copyBufferToInputBuf(char * bufToSaveInInput, uint length){
  uint i;
  for (i = 0; i < length; i++) {
    input.buf[(input.r+i)%INPUT_BUF] = bufToSaveInInput[i];
  }
  input.e = input.r+length;
  input.rightmost = input.e;
}
```

This method will copy the given buf to input.buf and it will set the input.e and input.rightmost assumes input.r=input.w=input.rightmost=input.e

```
void
copyBufferToScreen(char * bufToPrintOnScreen, uint length){
  uint i;
  for (i = 0; i < length; i++) {
    consputc(bufToPrintOnScreen[i]);
  }
}
```

```
case UP_ARROW:
  if (historyBufferArray.currentHistory < historyBufferArray.numOfCommmandsInMem-1 ){
    earaseCurrentLineOnScreen();
    if (historyBufferArray.currentHistory == -1)
        copyCharsToBeMovedToOldBuf();
    earaseContentOnInputBuf();
    historyBufferArray.currentHistory++;
    tempIndex = (historyBufferArray.lastCommandIndex + historyBufferArray.currentHistory) %MAX_HISTORY;
    copyBufferToScreen(historyBufferArray.bufferArr[ tempIndex]  , historyBufferArray.lengthsArr[tempIndex]);
    copyBufferToInputBuf(historyBufferArray.bufferArr[ tempIndex]  , historyBufferArray.lengthsArr[tempIndex]);
  }
  break;
```

Up Arrow interrupt functionality added in **consoleintr()**. We will look into the history if any previous command is present, we will print that to the terminal.

```
case DOWN_ARROW:
  switch(historyBufferArray.currentHistory){
    case -1:
      //does nothing
      break;
    case 0: //get string from old buf
      earaseCurrentLineOnScreen();
      copyBufferToInputBuf(oldBuf, lengthOfOldBuf);
      copyBufferToScreen(oldBuf, lengthOfOldBuf);
      historyBufferArray.currentHistory--;
      break;
    default:
      earaseCurrentLineOnScreen();
      historyBufferArray.currentHistory--;
      tempIndex = (historyBufferArray.lastCommandIndex + historyBufferArray.currentHistory)%MAX_HISTORY;
      copyBufferToScreen(historyBufferArray.bufferArr[ tempIndex]  , historyBufferArray.lengthsArr[tempIndex]);
      copyBufferToInputBuf(historyBufferArray.bufferArr[ tempIndex]  , historyBufferArray.lengthsArr[tempIndex]);
      break;
  }
  break;
```

Down Arrow interrupt functionality added in **consoleintr()**. We will look into the history if any further command is present, we will print that to the terminal.

```
void
saveCommandInHistory(){
  historyBufferArray.currentHistory= -1;
  if (historyBufferArray.numOfCommmandsInMem < MAX_HISTORY)
    historyBufferArray.numOfCommmandsInMem++;
  uint l = input.rightmost-input.r -1;
  historyBufferArray.lastCommandIndex = (historyBufferArray.lastCommandIndex - 1)%MAX_HISTORY;
  historyBufferArray.lengthsArr[historyBufferArray.lastCommandIndex] = l;
  uint i;
  for (i = 0; i < l; i++) {
    historyBufferArray.bufferArr[historyBufferArray.lastCommandIndex][i] =  input.buf[(input.r+i)%INPUT_BUF];
  }

}
```

This function saves the currently written command to history. We keep on saving commands in a cyclic way by using modular operations to avoid crossing **MAX_HISTORY**.

```
int history(char *buffer, int historyId) {
  if (historyId < 0 || historyId > MAX_HISTORY - 1)
    return -2;
  if (historyId >= historyBufferArray.numOfCommmandsInMem )
    return -1;
  memset(buffer, '\0', INPUT_BUF);
  int tempIndex = (historyBufferArray.lastCommandIndex + historyId) % MAX_HISTORY;
  memmove(buffer, historyBufferArray.bufferArr[tempIndex], historyBufferArray.lengthsArr[tempIndex]);
  return 0;
}
```

This is the function that gets called by the system call **sys_history** and writes the requested command history in the buffer.

```
void history1() {
  int i, count = 0;
  for (i = 0; i < MAX_HISTORY; i++) {
    if (history(cmdFromHistory, MAX_HISTORY-i-1) == 0) { //this is the sys call
      count++;
      printf(1, " %d: %s\n", count, cmdFromHistory);
    }
  }
}
```

This function is implemented in sh.c history1() gives the system call sys_history(), which will in turn call history() present in console.c

```
// Read and run input commands.
while(getcmd(buf, sizeof(buf)) >= 0){
  if(buf[0] == 'c' && buf[1] == 'd' && buf[2] == ' '){
    // Chdir must be called by the parent, not the child.
    buf[strlen(buf)-1] = 0;  // chop \n
    if(chdir(buf+3) < 0)
      printf(2, "cannot cd %s\n", buf+3);
    continue;
  }
  if(buf[0] == 'h' && buf[1] == 'i' && buf[2] == 's' && buf[3] == 't' && buf[4] == 'o' && buf[5] == 'r' && buf[6] == 'y' && buf[7] == '\n') {
    history1();
    continue;
  }
```

A second ‘if’ condition is added inside the while loop present in main of sh.c
This is a shell command that will call history1() present in the same file, which in turn results in the printing of command history on the qemu console.

Necessary modifications were made to handle the system call generated by history1() function. The procedure was the same as used in the previous lab. The following files were modified for the same:

- usys.S
- syscall.h
- syscall.c
- sysproc.c
- user.h

Task 2: Statistics

```
38  struct proc {
39    uint sz;                     // Size of process memory (bytes)
40    pde_t* pgdir;                // Page table
41    char *kstack;                // Bottom of kernel stack for this process
42    enum procstate state;        // Process state
43    int pid;                     // Process ID
44    struct proc *parent;         // Parent process
45    struct trapframe *tf;        // Trap frame for current syscall
46    struct context *context;     // swtch() here to run process
47    void *chan;                  // If non-zero, sleeping on chan
48    int killed;                  // If non-zero, have been killed
49    struct file *ofile[NOFILE];  // Open files
50    struct inode *cwd;           // Current directory
51    char name[16];               // Process name (debugging)
52
53    uint ctime;                   // Process creation time
54    int stime;                   //process SLEEPING time
55    int retime;                  //process READY(RUNNABLE) time
56    int rutime;                  //process RUNNING time
57  };
```

Proc struct was extended by adding the four fields namely ctime, stime ,retime, runtime.

1. ctime : process creation time
2. retime : The aggregated number of clock ticks during which the process waited
3. rutime : The aggregated number of clock ticks during which the process was running
4. stime : The aggregated number of clock ticks during which the process was waiting for I/O (was not able to run).

```
537 void updatestatistics() {
538   struct proc *p;
539   acquire(&ptable.lock);
540   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
541     switch(p->state) {
542       case SLEEPING:
543         p->stime++;
544         break;
545       case RUNNABLE:
546         p->retime++;
547         break;
548       case RUNNING:
549         p->rutime++;
550         break;
551       default:
552         ;
553     }
554   }
555   release(&ptable.lock);
556 }
557
```

This function whenever called will update all the parameters of all the current running processes which were newly defined in proc struct.

```
37  trap(struct trapframe *tf)
38  {
39    if(tf->trapno == T_SYSCALL){
40      if(myproc()->killed)
41        exit();
42      myproc()->tf = tf;
43      syscall();
44      if(myproc()->killed)
45        exit();
46      return;
47    }
48
49    switch(tf->trapno){
50    case T_IRQ0 + IRQ_TIMER:
51      if(cpuid() == 0){
52        acquire(&tickslock);
53        ticks++;
54        updatestatistics();
55        wakeup(&ticks);
56        release(&tickslock);
57      }
58      lapiceoi();
59      break;
```

Evaluation metrics (retime, runtime, stime) for the current process is updated with every clock cycle by calling updatestatistics() method at line 54 in trap.c file.

```
107 int sys_wait2(void) {
108     int res;
109     int retime = 0;
110     int rutime = 0;
111     int stime = 0;
112     argint(0, &retime);
113     argint(1, &rutime);
114     argint(2, &stime);
115     res = sys_wait();
116     *(int*)retime = myproc()->retime;
117     *(int*)rutime = myproc()->rutime;
118     *(int*)stime = myproc()->stime;
119     return res;
120   }
121
```

Implementation of sys_wait2 in sysproc.c using already defined sys_wait() to get the process ID.

Updates retime, rutime, stime for the current process to show in console.

```
194     else {
195       pid=wait2(&retime, &rutime, &stime);
196       printf(1, "pid:%d retime:%d rutime:%d stime:%d\n", pid, retime, rutime, stime);
197     }
198   }
199   exit();
```

wait2() will be called and information will be printed on the qemu console after each command. Implemented inside main function of sh.c

Necessary modifications were made to handle the system call generated by wait2() function. The procedure was the same as used in the previous lab. The following files were modified for the same:

- usys.S
- syscall.h
- syscall.c
- sysproc.c
- user.h
- defs.h