Implementation of head command

Implemented in XV6-RISCV in DOCKER (in Mac OS - M1)

Code logic:

File: head.c
All the variables are declared. If no filename is given, it is read as a standard input and passed to

/xv6-riscv/user/head.c

```c
int main(int argc, char *argv[]) {
      int i,n;
      int fd = 0;
      int fdk = 0;
      int flines = 14;
      char *file;
      char a;
      int f = 0;
      char buffer[512];
      file = "";

      if (argc <= 1) {
        head_f(0, "", flines);
        while((n = read(0, buffer, sizeof(buffer))) > 0 )
      {
         head(0, buffer, flines);
      }
        if (n < 0) {
          printf("Error: Unable to read file(Kernel mode)\n");
      }
          exit(0);
        }
```

the function and the system call. An appropriate error message is printed on failure.

/xv6-riscv/user/head.c C

```c
        }
        else {
              for (i = 1; i < argc; i++) {
                  a = *argv[i];
                  if(a != '-'){
                      if ((fd = open(argv[i], 0)) < 0){
                        printf("Error: Cannot Open File %s\n", argv[i]);
                        exit(1);
                       }
                  }
                  if (a == '-') {
                      argv[i]++;
                      flines = atoi(argv[i]++);
                      f = 1;
                  }
```

If '-' number of lines is given, it is stored in a variable *flines* and a flag f is set.
The arguments are passed to the function *head_f* where the user mode implementation is coded. The file is read and the content is passed to the *head()* system call and an appropriate error message is displayed if any. If multiple files are given, each file content is displayed preceded with the file name.

/xv6-riscv/user/head.c

```c
                if(f == 0){
                    printf("\nFile: %s\n", argv[i]);
                    head_f(fd, file, flines);
                    fdk = open(argv[i], 0);
                    while((n = read(fdk, buffer, sizeof(buffer))) > 0 )
                    {
                        printf("\nFile: %s\n", argv[i]);
                        head(n, buffer, flines);
                    }
                    if (n < 0) {
                        printf("Error: Unable to read file(kernel mode)\n");
                        close(fdk);
                    }
                }
                f = 0;
        }
        close(fd);
        exit(0);
    }
```

Each character of the file is read and printed until a line is completed.

/xv6-riscv/user/head.c

```c
#include "kernel/types.h"
#include "kernel/stat.h"
#include "user.h"

char buf[512];

int head_f(int fd, char *name, int line)
{
    int i, n, l_no;
    l_no = 0;
    printf("Head command is getting executed in user mode\n");
    while((n = read(fd, buf, sizeof(buf))) > 0 )
    {
      for(i=0;i<=n ;i++)
      {
        if(buf[i]!='\n'){
          printf("%c",buf[i]);
      }
```

l_no variable contains the total line number iterated. If it is equal to the provided line number, it returns back to the main block else the next line is read. If line number is not given, it is defaulted to 14 in main block which is passed to this function.

/xv6-riscv/user/head.c

```
        }
        else if (l_no == (line-1)){
              printf("\n");
              return 0;
        }
        else{
            printf("\n");
            l_no++;
        }
      }
    }
    if(n < 0){
      printf("head: unable to read(user mode)\n");
      exit(1);
    }
    return 0;
  }
```

File: sysproc.c

/xv6-riscv/kernel/sysproc.c C

```
uint64
sys_head(void)
{
  int n,i;
  int l=0;
  char buffer[512];
  int line;
  char convStr[2];
  argint(0, &n);
  argstr(1, buffer, sizeof(buffer));
  argint(2, &line); //lines to print - default 14
  printf("Head command is getting executed in kernel mode\n");
  for(i=0;i<=n ;i++)
    {
      if(buffer[i]!='\n'){
        convStr[0] = buffer[i];
        convStr[1]='\0';
        printf("%s", convStr);
      }
```

The arguments are read into respective variables. The file content is iterated over and each character is printed until the end of the line. As '%c' is not allowed, char is stored in a string and printed.

```
21        else if (l == (line-1)){
22          printf("\n");
23          return 0;
24        }
25        else{
26          printf("\n");
27          l++;
28        }
29      }
30    return 0;
31  }
```

The total lines iterated is stored in a variable, once it reaches the given line number it is returned back to the main program. If the line number is not given, it is defaulted to 14.

Other files modified:

/xv6-riscv/kernel/syscall.h

```
 6  #define SYS_read    5
 7  #define SYS_kill    6
 8  #define SYS_exec    7
 9  #define SYS_fstat   8
10  #define SYS_chdir   9
11  #define SYS_dup    10
12  #define SYS_getpid 11
13  #define SYS_sbrk   12
14  #define SYS_sleep  13
15  #define SYS_uptime 14
16  #define SYS_open   15
17  #define SYS_write  16
18  #define SYS_mknod  17
19  #define SYS_unlink 18
20  #define SYS_link   19
21  #define SYS_mkdir  20
22  #define SYS_close  21
23  #define SYS_uniq   22
24  #define SYS_head   23
```

Syscall.h

/xv6-riscv/kernel/syscall.c

```
 96  extern uint64 sys_uptime(void);
 97  extern uint64 sys_open(void);
 98  extern uint64 sys_write(void);
 99  extern uint64 sys_mknod(void);
100  extern uint64 sys_unlink(void);
101  extern uint64 sys_link(void);
102  extern uint64 sys_mkdir(void);
103  extern uint64 sys_close(void);
104  extern uint64 sys_uniq(void);
105  extern uint64 sys_head(void);
106
```

/xv6-riscv/kernel/syscall.c

```
121  [SYS_sbrk]    sys_sbrk,
122  [SYS_sleep]   sys_sleep,
123  [SYS_uptime]  sys_uptime,
124  [SYS_open]    sys_open,
125  [SYS_write]   sys_write,
126  [SYS_mknod]   sys_mknod,
127  [SYS_unlink]  sys_unlink,
128  [SYS_link]    sys_link,
129  [SYS_mkdir]   sys_mkdir,
130  [SYS_close]   sys_close,
131  [SYS_uniq]    sys_uniq,
132  [SYS_head]    sys_head,
133  };
134
```

syscall.c

usys.pl

/xv6-riscv/user/usys.pl

```
23  entry("write");
24  entry("close");
25  entry("kill");
26  entry("exec");
27  entry("open");
28  entry("mknod");
29  entry("unlink");
30  entry("fstat");
31  entry("link");
32  entry("mkdir");
33  entry("chdir");
34  entry("dup");
35  entry("getpid");
36  entry("sbrk");
37  entry("sleep");
38  entry("uptime");
39  entry("uniq");
40  entry("head");
```

user.h

/xv6-riscv/user/user.h

```
18  int mkdir(const char*);
19  int chdir(const char*);
20  int dup(int);
21  int getpid(void);
22  char* sbrk(int);
23  int sleep(int);
24  int uptime(void);
25  int uniq(int n, char *buffer, char var_icd);
26  int head(int n, char *buffer, int lines);
```

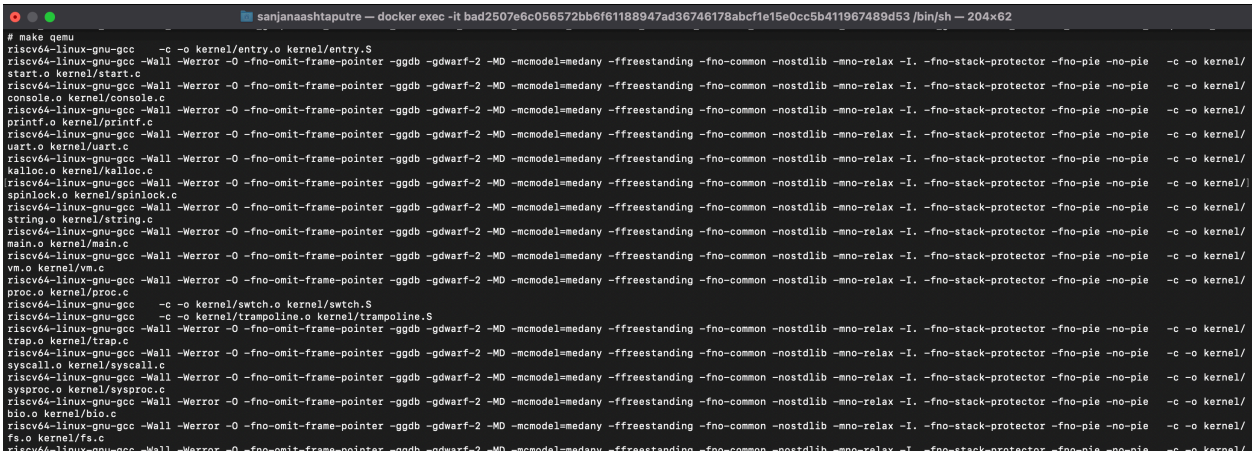
Input file and the output:

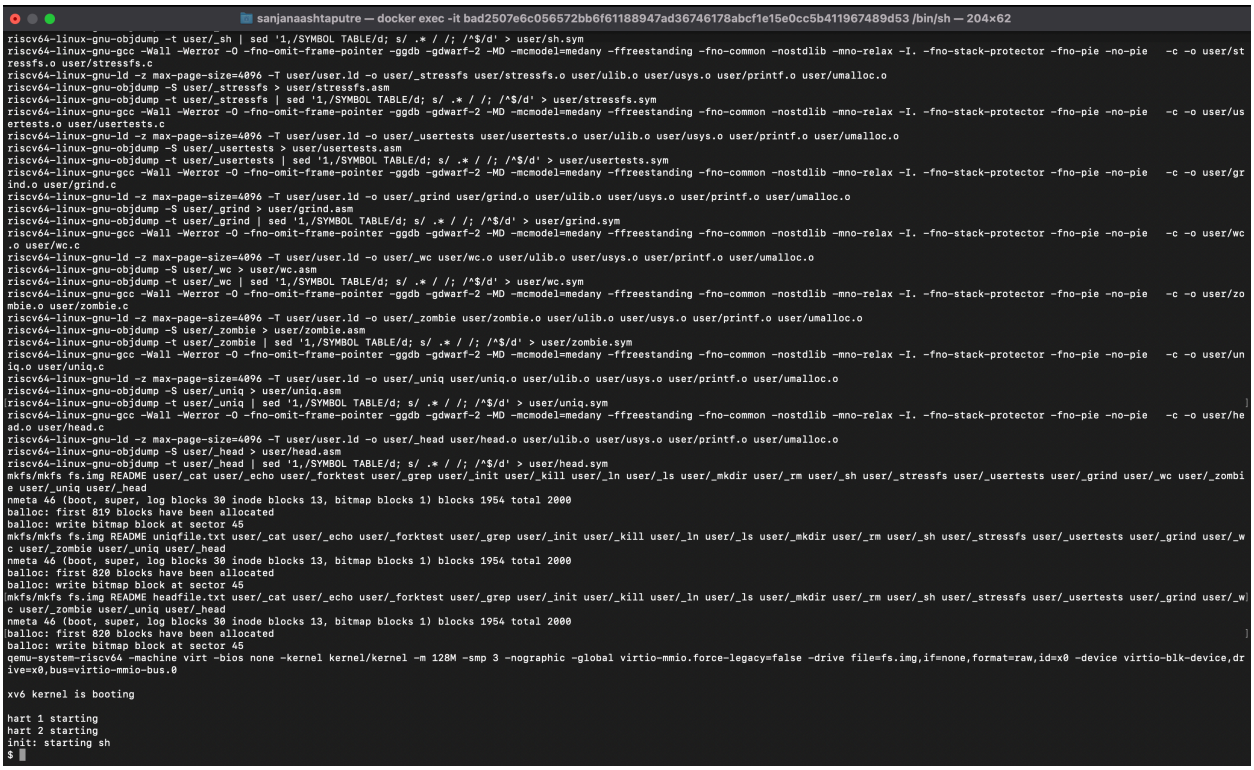
Make clean

sanjanaashtaputre — docker exec -it bad2507e6c056572bb6f61188947ad36746178abcf1e15...

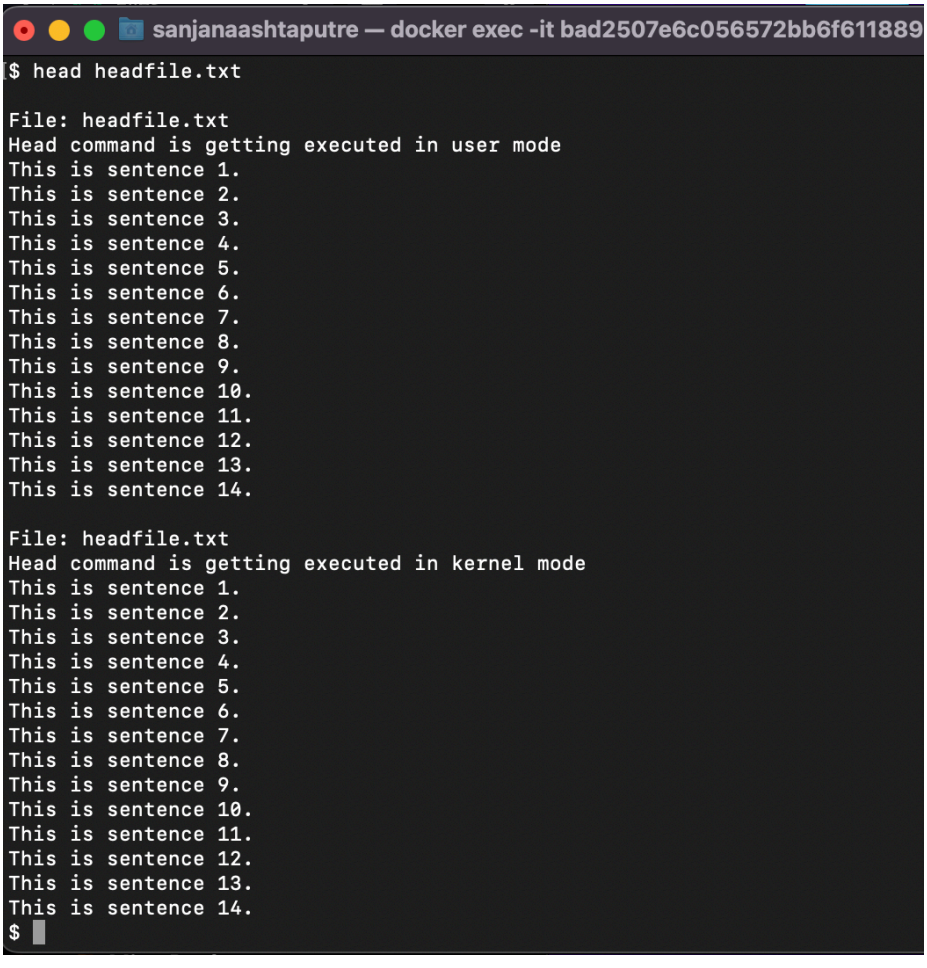
```
# make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*/*.o */*.d */*.asm */*.sym \
user/initcode user/initcode.out kernel/kernel fs.img \
mkfs/mkfs .gdbinit \
        user/usys.S \
user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user/_ln user/_ls user/_mkdir
user/_rm user/_sh user/_stressfs user/_usertests user/_grind user/_wc user/_zombie user/_uniq user/
_head
#
```

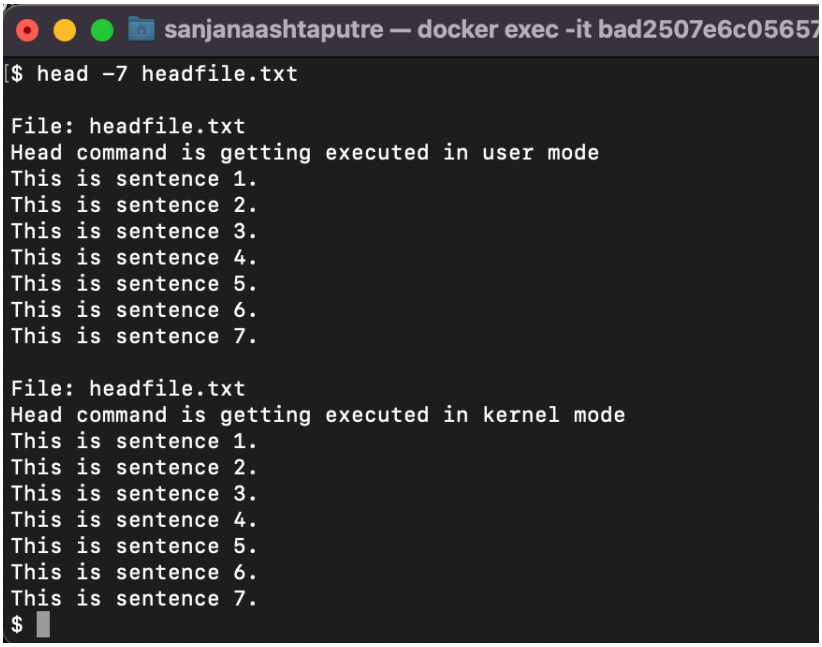
Make qemu





1. head headfile.txt



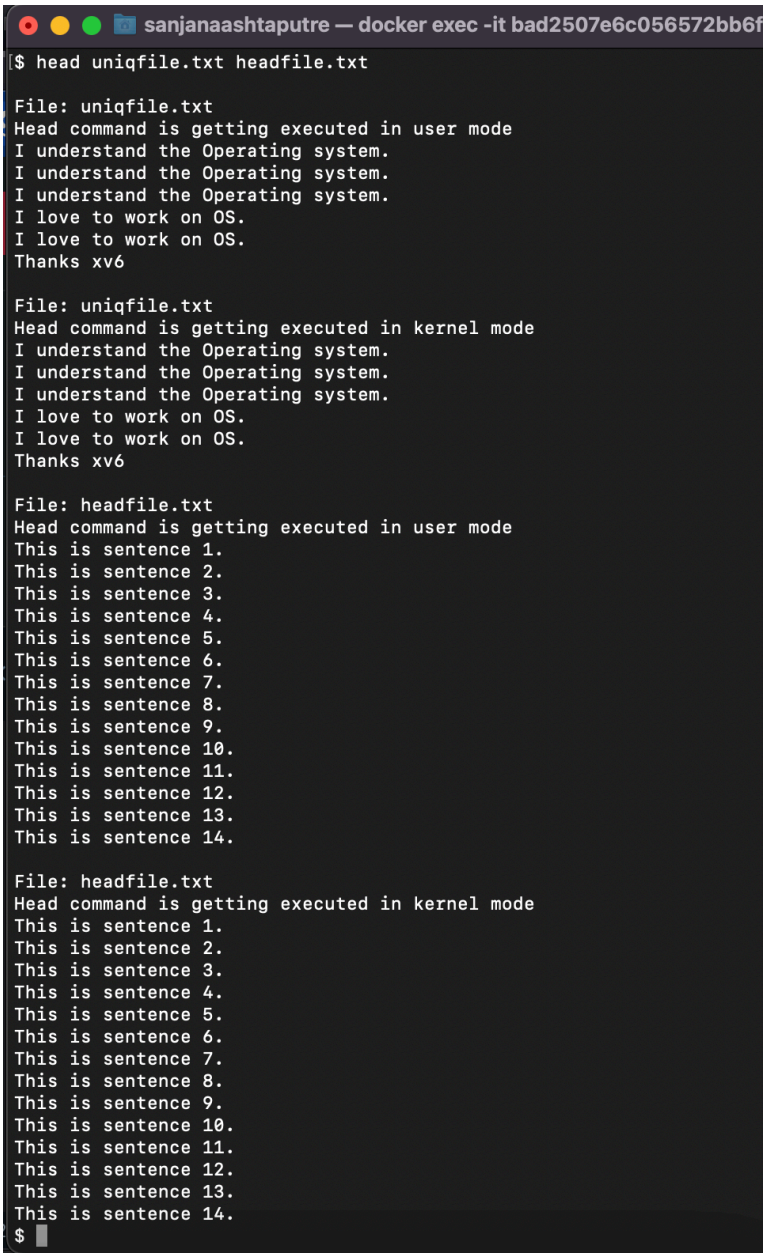


2. **head -7 headfile.txt**

Top 7 lines of the files are printed

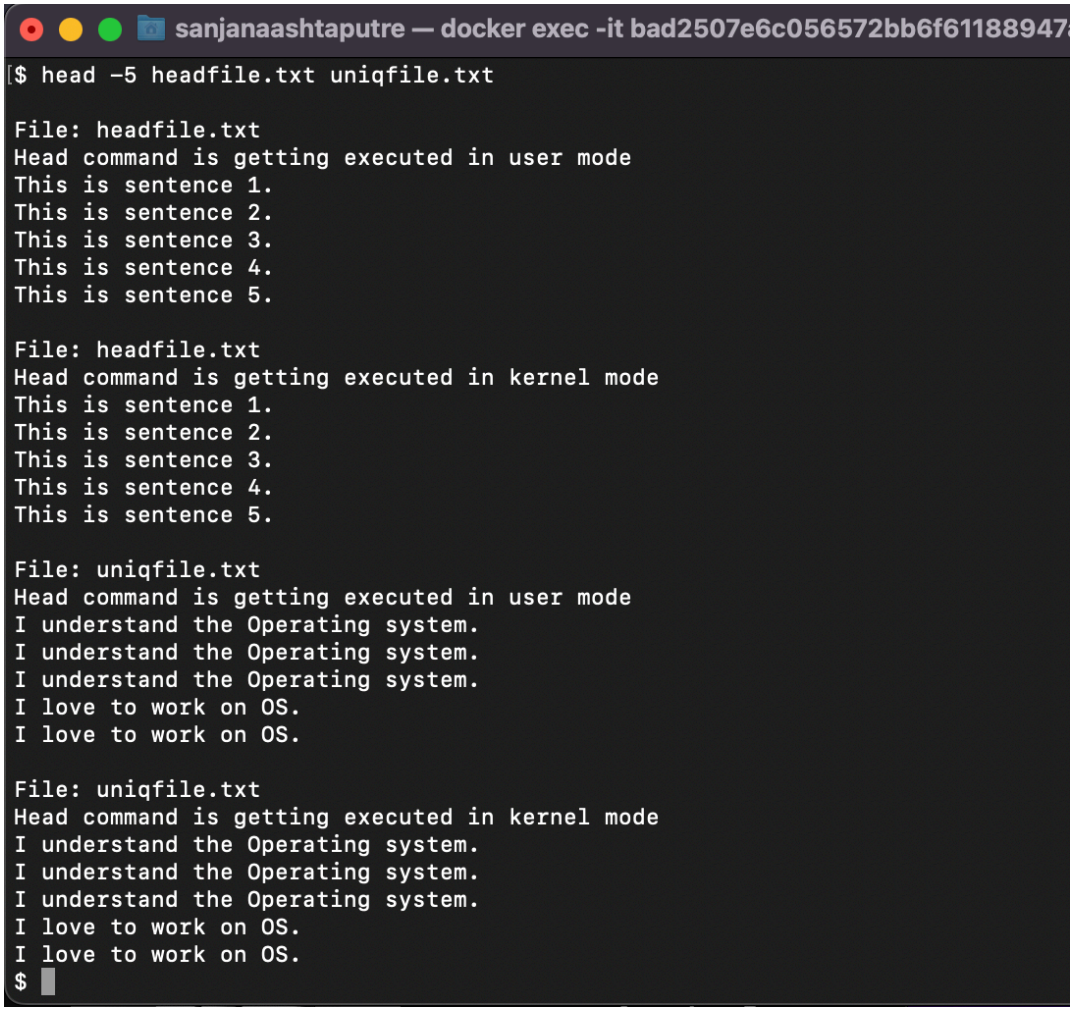
3. **head uniqfile.txt head file.txt**

When more than one file is given, both the contents unto 14 (default) lines are printed preceded with the file names.



4. **head -5 uniqfile.txt head file.txt**

When more than one file is given, both the contents unto 5 lines are printed preceded with the file names.



Input **headfile.txt**