Command: /bin/ls | /usr/bin/sort | /usr/bin/uniq

Output:

a.out
assembly64.pdf
assembly-ref
child.c
creds.txt
C Refresher
(C)(x86) ExpCodes
DBMS Original Lectures
Entangled State Prep and Teleportation Protocols.ipynb
Introduction to Computational molecular biology.pdf
Linux Kernel Development, 3rd Edition.pdf
multipipe.c
NFSMW2005
OS assignments
output.txt
pintos
pipe.c
pyCodes
shell.c
SPEC
The Bloch Sphere.ipynb
TimeTable.png

Explaination & Pseudocode:

When the above command is entered, this shell forks a child process using FORK syscall after taking the input string. The child process handles the command's execution while the parent process waits until this child process exits.

```
pid=fork()
if(pid = child-process)
        //rest of the processing//
else if(pid = parent-process)
        //wait for child to finish
else
        //report fork failure
```

The rest of the processing involves the following steps:-

1) Search for any redirection operation in the input string and apply if found.
   Pseudocode:

```
Iterate through the input string str:
        if ( '2>&1' in str):
                duplicate fd 2(stderr) to fd 1(stdout).
```

Remove this portion from the input string
if (‘1>filename’ in str):
close(stdout)
fd=open(filename)
Duplicate fd to 1(stdout)
Remove this portion from the input string
if (‘2>filename’ in str):
close(stderr)
fd=open(filename)
Duplicate fd to 2(stderr)
Remove this portion from the input string

2) Parse the processed input string and split into list of tokens
Pseudocode:
Iterate through the processed input string:
If cur=’ ‘ or cur=’\n’
Skip
If cur=’|’ //pipe
Add ‘|’ to the list
If cur=’<’
Add ‘<’ to the list
If cur=’>’
Add ‘>’ to the list
If cur=’>>’
Add ‘>’ to the list
while(cur not in [‘\0’, ’ ‘, ’\n’, ’|’, ’>’, ’<’])
Add cur word to the list
After adding all the args add a NULL
3) Iterate through the generated list of tokens, search for commands and operators and handle them accordingly. This is the portion where actual execution of commands takes place.
Pseudocode:
Iterate through the list of tokens:
Initialize an empty list:args
Add tokens upto the next NULL to the list:args

op=the_next_token
if(op==’|’) //pipe
Declare a pipe: pipe(p[2]);
Fork a child process
In the child:
close(1) //close stdout
dup(p[1]) //duplicate p[1] to fd=1

```
                close(p[0])     //close read side of pipe:p[0]
                Execute the commands using args collected above.
            In the parent:
                close(0);       //close stdin
                dup(p[0]);      //duplicate p[0] to fd=0
                close(p[1]);    //close write side of pipe:p[1]

        if(op=='>')     //redirect stdout to a file in trunk mode
            Open the file specified in file descriptor fd in trunk mode
            Fork a child process
            In the child:
                close(1)        //close stdout
                dup(fd)         //duplicate the output file in fd to 1(stdout)
                Execute the commands using args collected above.
            In the parent:
                wait(NULL)      //wait for the child to exit

        if(op=='>>')    //redirect stdout to a file in append mode
            Open the file specified in file descriptor fd in append mode
            Fork a child process
            In the child:
                close(1)        //close stdout
                dup(fd)         //duplicate the output file in fd to 1(stdout)
                Execute the commands using args collected above.
            In the parent:
                wait(NULL)      //wait for the child to exit
        if(op=='<')     //read a file and redirect to stdin
            Open the file specified in file descriptor fd in read mode
            Fork a child process
            In the child:
                close(0)        //close stdin
                dup(fd)         //duplicate the output file in fd to 0(stdin)
                Execute the commands using args collected above.
            In the parent:
                wait(NULL)      //wait for the child to exit

        if( last command in the input) //#
            Just execute the command using args collected in above loop.
```

Specifically referring to the given command:

There is no redirection command apart from the pipes so step 1 does not affect the input the string and nothing is executed in step 1. Step 2 splits the input string into tokens and stores in the list called cmd. Now step 3 takes over. In the first iteration, all the arguments corresponding

to /bin/ls are stored into the args list. Then a pipe is found so a pipe object is initialized and in newly created child process stdout is closed to duplicate the write side of the pipe object to fd=1 followed by closing the read side of the same pipe object and executing the command specified by the args. The parent process during this time closes stdin and duplicates the read side of the pipe object followed by closing of the write side. This same step is repeated in the second iteration but this time for /usr/bin/sort. In the third and final iteration since the end of the input string is reached, just the args fetched for /usr/bin/uniq are used and the command is directly executed in the last if statement in the pseudocode above marked as '//#'.