Operating System
SHELL

Assignment 3 Design Documentation

PARAS MEHAN
2018062

Pseudo-Code of shell.c

```
int main()
{
	while(1)
	{
		print "₹" to STDOUT
		A <- String Input from STDIN
		if( A == "exit")
		{
			Terminate while loop.
		}
		else
		{
			Fork a child process and call func(A) inside child
process and wait for the child to exit.
		}
	}
	return 1;
}
void func(char * A)
{
	if( A == "" )
		exit(0);
	pip <- 0;
	if( '|' in A)
	{
		pip <- 1;
		(A,B) <- (Split A on 2 Parts, Before are after the 1st '|', let
A = part before 1st pipe, and B be the part after the first pipe);
```

```
}

char *tmp[1000],*args[1000];
int argc <- 0;
tmp <- ( Split A on spaces, For Ex A = "B C D" , tmp = ['A','B','C'])
for i in [0,1,2,....len(tmp)]:
{
      if(tmp[i][0]=='1' and tmp[i][1]=='>')
      {
            // This case correspondes to 1>filename
            close(0);
            filename <- extract(tmp[i]);
            creat(filename,0666);
      }
      else if(tmp[i]=="2>&1")
      {
            //This case correspondes to 2>&1
            close(2);
            dup(1);
      }
      else if(tmp[i]==">>")
      {
            //This case correspondes to >>
            filename <- tmp[i+1];
            i <- i+1;
            close(1);
            open(filename, O_WRONLY | O_APPEND | O_CREAT);
      }
      else if(tmp[i][0]=='>')
      {
            //This case correspondes to >
            filename <- tmp[i+1];
            i <- i+1;
            close(1);
            creat(filename,0666);
      }
      else if(tmp[i][0]=='<')
      {
            //This case correspondes to >
            filename <- tmp[i+1];
            i <- i+1;
            close(1);
```

```
                open(filename, O_RDONLY);
            }
            else
            {
                args[argc] <- tmp[i];
                argc+=1;
            }
        }
        if(pip==0)
            execvp(args[0],args);
        else
        {
            //The case where pipe is present.
            int fd[2];
            pipe(fd);
            int pid = fork();
            if(pid == 0)
            {
                close(fd[0]);
                close(1);
                dup(fd[1]);
                close(fd[1]);

                // We execute the command inside the child process.
                execvp(args[0],args);
            }
            else
            {

                close(fd[1]);
                close(0);
                dup(fd[0]);
                close(fd[0]);

                // We continue to parse the rest of the string (B, string
after the 1st '|')
                func(B);
            }
        }
}
```

Example Input Explanation

Input : “$ /bin/ls | /usr/bin/sort | /usr/bin/uniq”

- In the main **shell** process, the shell takes input from the user and Saves it in a string “**A**”.
- The shell process forks a new child process and calls a function with the name “**func**(**A**)” inside the child process(We will call it “**Head**” process).
- **func** determines whether there is “|” present in **A** or not.
- Since “|” is present in **A**, it breaks A into 2 parts one before “|” and one after(**B**). **A** now becomes “/bin/ls” and **B** = “ /usr/bin/sort | /usr/bin/uniq”.
- Then **func** splits into spaces and saves it in a temporary array(**tmp**).
- Then it traverses through the array and checks whether “>” or “>>” or “1>filename” or “2>filename” or “2>&1” is present in the temporary array.
- As string A does not have these, it adds the elements of temporary array into **args** (**args**={“/bin/ls”,”NULL”}).
- Now since the pipe was present in A, it creates a pipe using **pipe**() command and forks a new child process (we will call it **child1**).
- Then **func1** changes the file descriptor of child1 process (in the if(pid==0) case) and calls **execvp**(args[0],args) inside **child1**. Hence **child1** executes “/bin/ls”.
- Then in the else case(pid>0), **func** changes the file descriptors (according to pipe) and recursively calls **func**(**B**).
- Now Head process again executes **func** with the new string “/usr/bin/sort | /usr/bin/uniq”.
- The same process repeats, with **A** = “/usr/bin/sort | /usr/bin/uniq”.
- “|” is again present in **A**, it again splits it on 1st pipe.
- **A** = “/usr/bin/sort” and **B**=”/usr/bin/uniq”.
- Then A is split on spaces and saved in **tmp** array. **tmp**=[“/usr/bin/sort”].
- Then it traverses through the **tmp** array and checks whether “>” or “>>” or “1>filename” or “2>filename” or “2>&1” is present in the temporary array.
- Then elements of **tmp** are added to **args** array.
- Since “|” was present, the Head process again forks a new child process (**child2**) and **child2** executes “/use/bin/sort”, after **func** changes the file descriptors.
- After changing the file descriptors, the **Head** process again recursively calls func with **A** = “/usr/bin/uniq”.

- This time no “|” is present, hence after following the same steps, we get **args**=[“/usr/bin/uniq”].
- Note that in all the steps, only the file-descriptor 0(STDIN) of the Head process is changing, file-descriptor 1 is still STDOUT of the **shell** process.
- The **Head** executes the command “/usr/bin/uniq”, by calling execvp() and prints the output on STDOUT.

Note.

- “₹” is used in place of “$”.
- The following assumptions are taken while taking input:
    - While Using “>” / output redirection, there is a space before and after “>”.
    - While Using “<” / input redirection, there is a space before and after “<”.
    - While Using “1>filename” - output redirection, there is a no space between ‘1’,’>’,’filename’.
    - While Using “2>&1”, there is no space between ‘2’,’>’,’&’,’1’.