REPORT

Assignment 2 : Building your own shell

Shashwat Johri

12041380

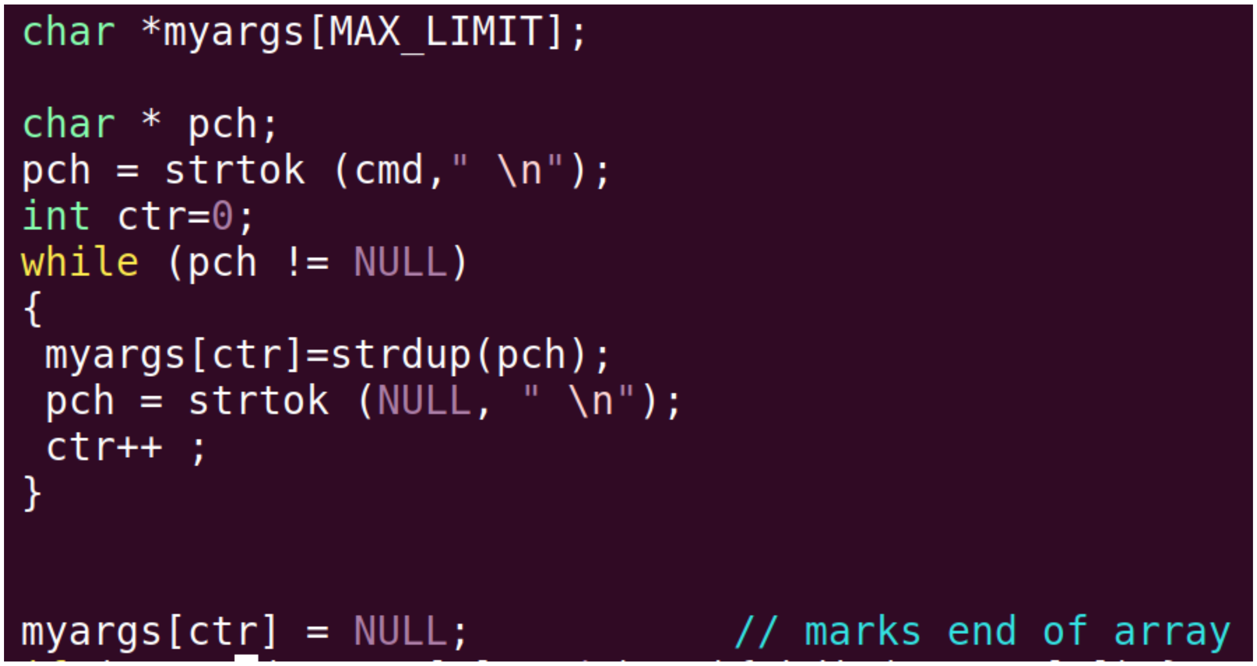
PART A:

The major parts of this-

Splitting the input string based on space as a delimiter:

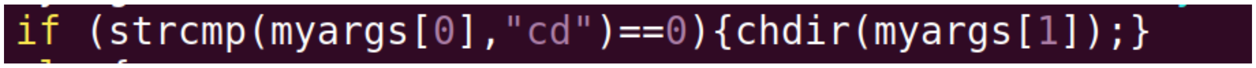
Done using strtok

The myargs[] must end with NULL to be put in execvp



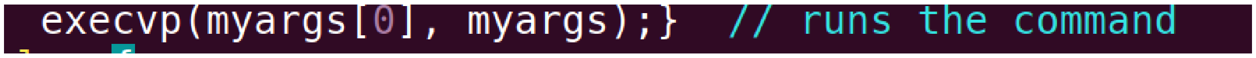
```c
char *myargs[MAX_LIMIT];

char * pch;
pch = strtok (cmd," \n");
int ctr=0;
while (pch != NULL)
{
 myargs[ctr]=strdup(pch);
 pch = strtok (NULL, " \n");
 ctr++ ;
}


myargs[ctr] = NULL;           // marks end of array
```

Checking for cd separately and implementing is using chdir:



```c
if (strcmp(myargs[0],"cd")==0){chdir(myargs[1]);}
```

For the rest do execvp:



```c
 execvp(myargs[0], myargs);}  // runs the command
```

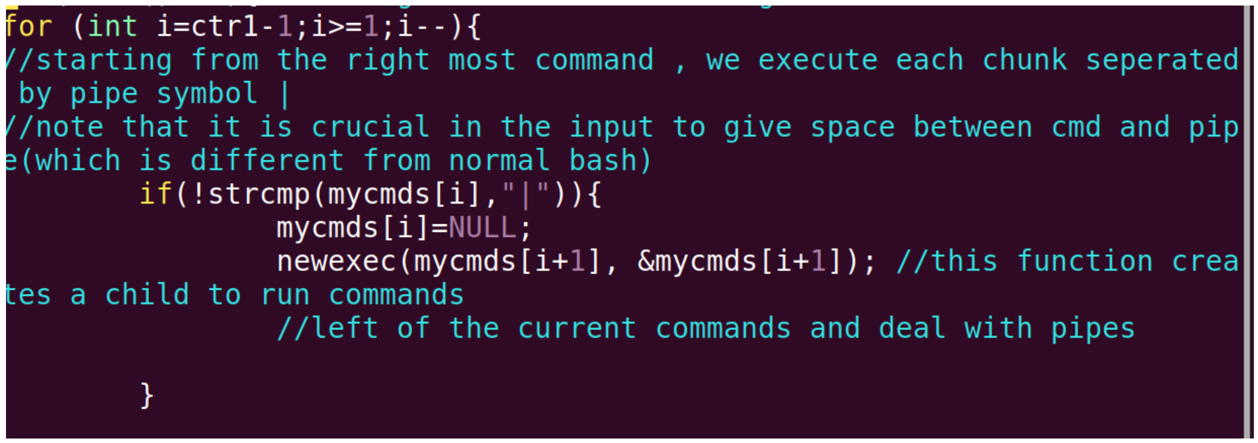
Now for the bash-like experience we envelop all of this in a while loop and fork everytime we run a command .

PART B-1 :

We use strtok to make array of the following manner-

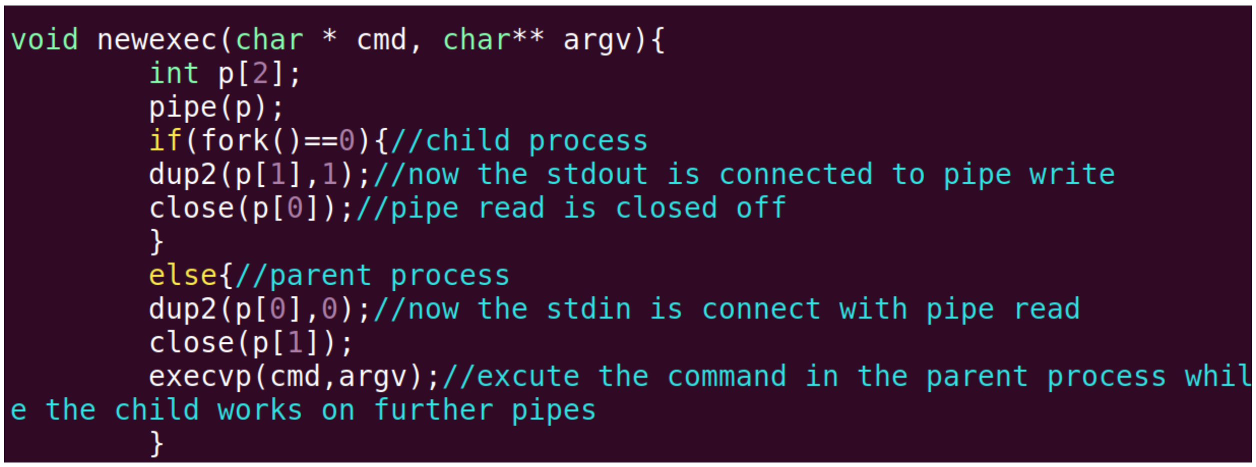
{“ls”,”|”,”grep”, “.c”,NULL}

Now starting from the right we separate out the command chunks and pass them to newexec() function



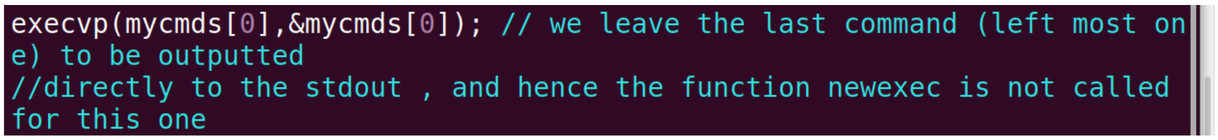
```
for (int i=ctr1-1;i>=1;i--){
//starting from the right most command , we execute each chunk seperated
 by pipe symbol |
//note that it is crucial in the input to give space between cmd and pip
e(which is different from normal bash)
        if(!strcmp(mycmds[i],"|")){
                mycmds[i]=NULL;
                newexec(mycmds[i+1], &mycmds[i+1]); //this function crea
tes a child to run commands
                //left of the current commands and deal with pipes

        }
```

The newexec function takes the cmd chunk and makes a child that is responsible for carrying out further commands to the left of this one the parent does execvp. The stdout and stdin are connected with pipes



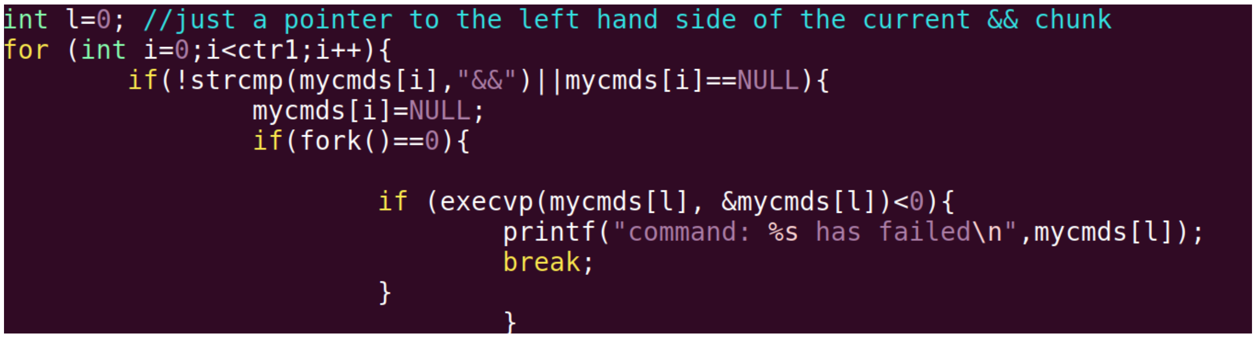
```
void newexec(char * cmd, char** argv){
        int p[2];
        pipe(p);
        if(fork()==0){//child process
        dup2(p[1],1);//now the stdout is connected to pipe write
        close(p[0]);//pipe read is closed off
        }
        else{//parent process
        dup2(p[0],0);//now the stdin is connect with pipe read
        close(p[1]);
        execvp(cmd,argv);//excute the command in the parent process whil
e the child works on further pipes
        }
```

We allow the last (to the left) command to output directly ot the stdout, hence the overall output comes on to the terminal



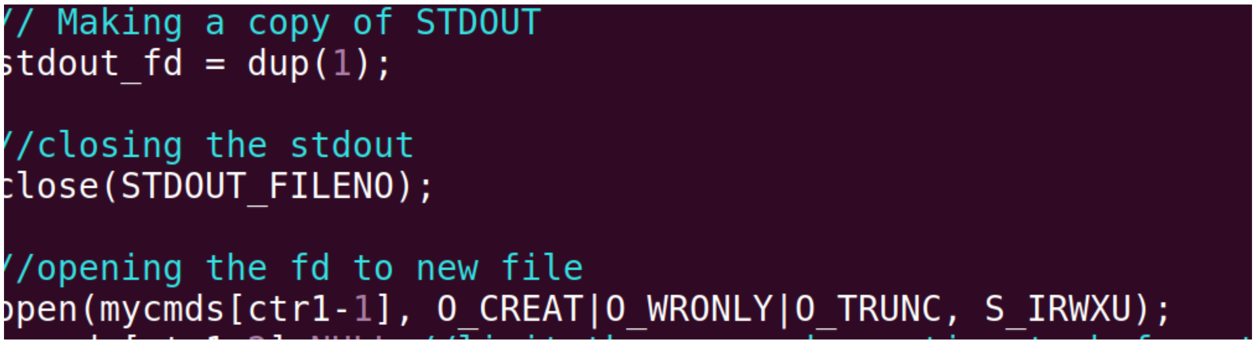
```
execvp(mycmds[0],&mycmds[0]); // we leave the last command (left most on
e) to be outputted
//directly to the stdout , and hence the function newexec is not called
for this one
```

PART B-2:
We use strtok to make array of the following manner-
{"ls","&&","ps",NULL}
Now starting from the left we execvp the chunk of cmds unless the execvp returns a negative value, in that case we stop the execution and wait for the next command.



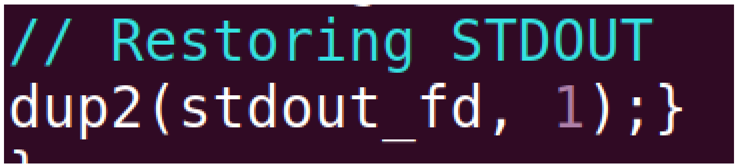
```
int l=0; //just a pointer to the left hand side of the current && chunk
for (int i=0;i<ctr1;i++){
        if(!strcmp(mycmds[i],"&&")||mycmds[i]==NULL){
                mycmds[i]=NULL;
                if(fork()==0){

                        if (execvp(mycmds[l], &mycmds[l])<0){
                                printf("command: %s has failed\n",mycmds[l]);
                                break;
                        }
                                }
```

PART C:
For this part we close the stdout FD and open the FD to the filename after >



```
// Making a copy of STDOUT
stdout_fd = dup(1);

//closing the stdout
close(STDOUT_FILENO);

//opening the fd to new file
open(mycmds[ctr1-1], O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
```

After the command is done we reopen the STDOUT fd



```
// Restoring STDOUT
dup2(stdout_fd, 1);}
```