Computer Architecture and Operating Systems
MONSOON SEMESTER of 2019

INSTRUCTOR: DR. Sambuddho Chakravarty

Creating Own Linux Shell

Step 1: Take and parse inputs from the user using the fgets() function.

This is done using a character array that takes space separated input as an individual element in an array. Include the relevant libraries.

```
char inputval[21] ={0x0};
              char *ptr =inputval;
              char *args[21]={NULL};
              printf("\n -> ");
              fgets(inputval,21,stdin);
              // printf("%s\n",  );
              strcpy(history[counterval],inputval);
              counterval+=1;
              for (int i=0; i<sizeof(args) && *ptr; ptr++)
              {
                   if (*ptr ==' ') continue;
                   if (*ptr == '\n') break;
                   for (args[i++] = ptr; *ptr && *ptr != ' ' &&
*ptr != '\n';     ptr++);
                   *ptr = '\0';
              }
```

Step 2: Create a child process using the fork() system call in which the whole shell will run.

```
System("clear");
	pid_t child=fork();
	if(child<0)
	{
		perror("Fork Error. PID<0.");
		exit(-1);
	}
	if (child!=0)
	{
		// printf("I'm the parent with pid %d\n", getpid() );
		wait(NULL);
	}
	if (child==0)
	{ // Code //
	}
```

Step 3: Within this child, every command forks a new child, executes the command and then exits, thereby killing the child process. During execution, the wait() function is set to NULL, hence the child won't be killed unless the execution completes.

```
if (strcmp(inputval, "ls")==0)
                {
                    pid_t newchild=fork();
                    if (newchild == 0)
                    {
                        execv("ls",args);
                        // exit(0);
                    }
                    wait(NULL);

                    // printf("input is %d", *inputval);
                }
```

Step 4: Repeat the forking for all commands. Use system() systemcall for internal commands.

Step 5: For external commands, create new C files and write the implementation of the respective commands. Call this compiled C file in the main shell file using the execv() systemcall.

```
///// rm.c //////
int main(int argc,char *filedval[])
{
	// printf(fileval[1]);
	printf("I'm RM command!\n");
	// for (int i=0; i<21; i++){
	//    printf("%s\n", argv[i]);
	// }
	if(strcmp(filedval[1],"-I")==0)
	{
		printf("Do You Want to Delete %s\n",filedval[2]);
		char choice[5];
		scanf("%c",&choice);
		remove(filedval[2]);


	}
	else
		{
			printf("Not Deleted");
		}


	return 0;
}
```

```
////// mainshell.c ///////
if (strcmp(inputval, "rm")==0)
                {
                        pid_t newchild=fork();
                        if (newchild == 0)
                        {
                                execv("rm",args);
                                // exit(0);
                        }
                        wait(NULL);

                        // printf("input is %d", *inputval);
                }
```

Step 6: Finally create a Makefile that compiles the main shell C file and executes it.

```
makeshell: hello.c

        gcc -o mainshell hello.c

        ./mainshell
```

Step 7: The shell is ready!