

Universidad de Concepción
Facultad de Ingeniería
Departamento de Ing. Informática y
Cs. de la Computación

Informe proyecto 1

Sistemas Operativos
Profesora Cecilia Hernández

Alumnos:

Nicolás Araya
Martina Cádiz

Fecha:

14/10/21

Introducción

El presente proyecto tiene como objetivo entender y conocer cómo trabajan algunas funciones a más bajo nivel, mediante la creación de nuevos comandos, el uso de fork y pipes se podrá extraer la noción del trabajo de una Shell de Linux. Este proyecto fue escrito en C++ usando el sistema operativo Linux.

Se busca desarrollar un intérprete de comandos escrito en C++ que cumpliera con ciertas funciones aplicando los conocimientos de llamadas a sistema y manejo de procesos.

El informe se escribirá en orden de aparición de las preguntas y se mostrarán capturas de consola para ejemplificar cada respuesta.

Primera parte

a) Prompt:

El intérprete de comandos cuenta con un prompt designado por el carácter '\$', también junto a este muestra la ruta actual.

```
/home/nicolas/Documentos/shell $ █
```

b) La shell lee un comando junto con sus argumentos. Internamente es parseada para separar comando de argumentos, no tiene límite de cantidad de argumentos.

```
vector<const char*> parsing(string s){
    vector<const char*> args;
    vector<string> commands;
    string aux = "";
    for(auto i : s){
        if(i != ' ' ) aux+=i;
        else{
            commands.push_back(aux);
            aux = "";
        }
    }
    commands.push_back(aux);

    for(int i = 0; i < commands.size(); i++){
        args.push_back(commands[i].c_str());
    }

    return args;
}
```

La función parsing devuelve un vector en el que cada elemento es una palabra

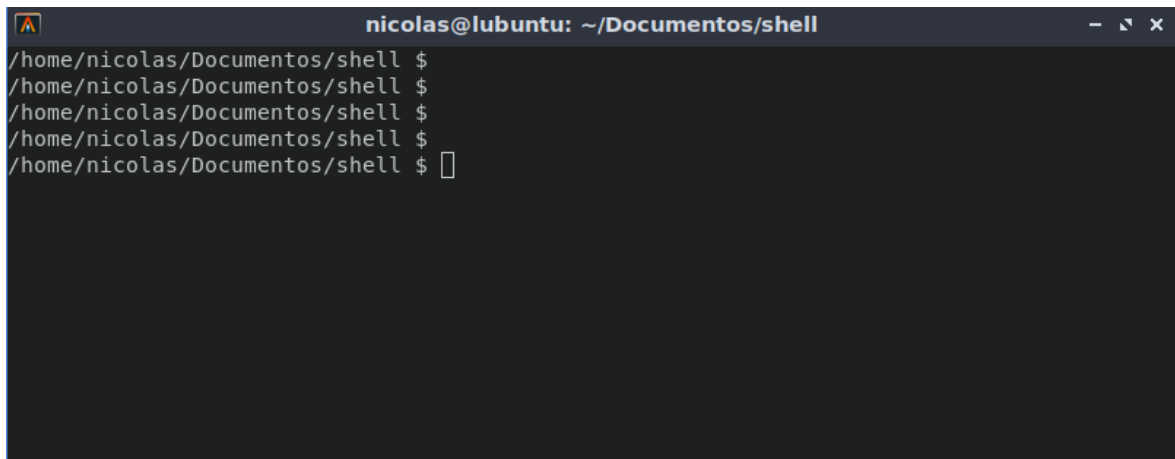
```
/home/nicolas/Documentos/shell $ apt-cache search sdk
default-jdk - Standard Java or Java compatible Development Kit
default-jdk-headless - Standard Java or Java compatible Development Kit (headless)
libglm-dev - C++ library for OpenGL GLSL type-based mathematics
```

c) Para la ejecución de un comando, se utiliza un llamado a sistema (fork()) y el comando se ejecuta utilizando execvp(). La shell espera por la ejecución hasta que esta termine para volver al prompt.

```
int pid = fork();
if(pid<0) cout << "error" << endl;
else if (pid == 0) {
    execvp(args[0], args);
    perror("comando no existente");
    exit(EXIT_FAILURE);
}
else{
    wait(NULL);
}
```

Para la ejecución, se tiene un arreglo args[0], que contiene el comando y los argumentos del mismo.

d) Si se pulsa “enter” simplemente se vuelve a mostrar prompt a la espera de un comando

A screenshot of a terminal window titled "nicolas@lubuntu: ~/Documentos/shell". The terminal shows the shell prompt "/home/nicolas/Documentos/shell \$" repeated five times, with the cursor visible at the end of the last line, indicating that the shell is waiting for input after each "enter" press.

```
nicolas@lubuntu: ~/Documentos/shell
/home/nicolas/Documentos/shell $
/home/nicolas/Documentos/shell $
/home/nicolas/Documentos/shell $
/home/nicolas/Documentos/shell $
/home/nicolas/Documentos/shell $
```

e) La shell es capaz de soportar comandos que se comunican con pipes. Tampoco tiene límite en la ejecución de cuantos comandos pueden estar unidos.

```
/home/nicolas/Documentos/shell $ ls -a | rev
.
..
tuo.a
ppc.llehs
/home/nicolas/Documentos/shell $
```

f) Si se ingresa como comando “exit” o “EXIT”, la ejecución de la shell finaliza

```
/home/nicolas/Documentos/shell $ exit
nicolas@lubuntu:~/Documentos/shell$
```

g) Si es que se ingresa un comando no válido, se manifiesta al usuario que el comando no existe, y se vuelve a la espera de una instrucción válida

```
/home/nicolas/Documentos/shell $ comando1
comando no existente: No such file or directory
/home/nicolas/Documentos/shell $ ls -a | rev
.
..
tuo.a
ppc.llehs
/home/nicolas/Documentos/shell $
```

Segunda parte

a) Al pulsar Control+C, la shell pedirá confirmación si realmente se quiere finalizar la shell. Para esto se debe responder con una ‘y’ en caso de ser afirmativo, y con una ‘n’ en caso contrario.

```
/home/nicolas/Documentos/shell $ ^C
Realmente quieres salir? [y/n] n

/home/nicolas/Documentos/shell $
```

En caso de responder negativamente, la shell vuelve a la espera por un comando.

b) En este ejercicio se utilizó la estructura de datos map para poder almacenar los comandos nuevos. Las “keys” del map corresponden a los nombres de los comandos y sus valores es un par de enteros, donde el primero es “x” y el segundo “z”. En la función exec() se insertan los nuevos comandos en caso de que la función hasCmdmonset() arroje el valor falso. Si el comando ya existe en el map, se activa la señal con una alarma que contiene como parámetro la función sig_handler2:

```
void sig_handler2(int sig) {
    vector<const char*> arg = parsing(now);
    char *args[arg.size()+1];
    for(int i = 0; i < arg.size(); i++){
        args[i] = (char*)arg[i];
```

```

    }
    args[arg.size()]=NULL;

    cerr << endl << "writing in log.txt..." << endl;
    int p[2];
    pipe(p);
    int pid = fork();
    if(pid<0) cout << "error" << endl;
    else if (pid == 0) {
        close(p[0]);
        dup2(p[1],1);
        dup2(p[1],2);
        close(p[1]);

        char *command[2];
        command[0] = strdup("vmstat");
        command[1] = NULL;

        execvp(command[0], command);
        perror("comando no existente");
        exit(EXIT_FAILURE);
    }
    else{
        char buffer[1024] = {0};
        close(p[1]);

        while(read(p[0], buffer, sizeof(buffer)) != 0){
            cerr << buffer << endl;
            file << buffer << '\n';
        }
    }
}

    if (++c < new_Comm[args[0]].second / new_Comm[args[0]].first )
alarm(new_Comm[args[0]].first);
    else {
        printf("Press any key for continue\n");
        file.close();
    }
}
}

```

El objetivo de esta función es cumplir con escribir la información del comando “vmstat” en un archivo .txt z/x veces en un tiempo z. Para lograr esto se hizo uso de una pipe y fork. Además para el reconocimiento del comando vmstat se llamó a execvp(). Cabe señalar que la cantidad de comandos que se pueden crear es indefinida, es decir pueden existir comandos ilimitados.

Ejemplo en consola:

```
/home/nicolas/Documentos/shell $ cmdmonset newcom 2 10
/home/nicolas/Documentos/shell $
```

Una vez ejecutado el comando, se monitorea el sistema según los tiempos establecidos, mientras dure el monitoreo se muestra por consola el resultado de cada ejecución del comando. Finalmente se crea un archivo log.txt que contiene todos los resultados recopilados

```
/home/nicolas/Documentos/shell $
writing in log.txt...
procs -----memory----- ---swap-- -----io---- -system-- -----cpu-----
r b  swpd  free  buff  cache  si  so  bi  bo  in  cs us sy id wa st
1 0  15180 389540 105744 2443464    0    0   45  196  823  613 12  1 86  1  0

writing in log.txt...
procs -----memory----- ---swap-- -----io---- -system-- -----cpu-----
r b  swpd  free  buff  cache  si  so  bi  bo  in  cs us sy id wa st
1 0  15180 389540 105744 2443464    0    0   45  196  823  613 12  1 86  1  0

writing in log.txt...
procs -----memory----- ---swap-- -----io---- -system-- -----cpu-----
r b  swpd  free  buff  cache  si  so  bi  bo  in  cs us sy id wa st
1 0  15180 389540 105748 2443464    0    0   45  196  823  613 12  1 86  1  0
```

```
/home/nicolas/Documentos/shell $ cat log.txt
procs -----memory----- ---swap-- -----io---- -system-- -----cpu-----
r b  swpd  free  buff  cache  si  so  bi  bo  in  cs us sy id wa st
1 0  15180 394320 105760 2443564    0    0   45  196  823  614 12  1 86  1  0

procs -----memory----- ---swap-- -----io---- -system-- -----cpu-----
r b  swpd  free  buff  cache  si  so  bi  bo  in  cs us sy id wa st
1 0  15180 394320 105760 2443564    0    0   45  196  823  614 12  1 86  1  0

procs -----memory----- ---swap-- -----io---- -system-- -----cpu-----
r b  swpd  free  buff  cache  si  so  bi  bo  in  cs us sy id wa st
3 0  15180 390532 105760 2443564    0    0   45  196  823  614 12  1 86  1  0

procs -----memory----- ---swap-- -----io---- -system-- -----cpu-----
r b  swpd  free  buff  cache  si  so  bi  bo  in  cs us sy id wa st
2 0  15180 390280 105760 2443564    0    0   45  196  823  614 12  1 86  1  0

procs -----memory----- ---swap-- -----io---- -system-- -----cpu-----
r b  swpd  free  buff  cache  si  so  bi  bo  in  cs us sy id wa st
1 0  15180 390028 105760 2443564    0    0   45  196  823  614 12  1 86  1  0
```