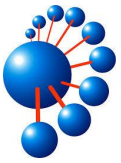

Proyecto 1:
Mini-Shell

Sistemas Operativos (2022-2)

Integrantes: Marco Antonio Aguayo Solís
Dazhi Enrique Feng Zong
Pablo Ignacio Zapata Schifferli
Profesora: Cecilia Hernández R.
Fecha: 25 de septiembre, 2022

Introducción

Este proyecto tiene como objetivo programar un shell simple con algunas funciones vistas en clases y laboratorios. El objetivo es profundizar en procesos, con llamadas como fork(), wait(), exec(), además de señales y pipes.

Desarrollo

La implementación consiste de un solo archivo .cpp, en donde la shell está implementada en el main(). A continuación, se explicará el detalle de la implementación.

1. El prompt es básicamente imprimir “mini-shell$:” y leer el input con getline().
2. Luego de leer el input con getline(), esta se parsea con strings y stringstream. Se guarda cada comando y sus argumentos en un vector de strings, y este vector se guarda en un vector de vectores.
3. Si lee enter (línea vacía), se imprime otra vez el prompt.
4. El comando exit terminará el programa.
5. El comando “usorecursos start <nombre>” inicializa el guardado de usos de recursos de cada comando introducido a partir de ese momento.
6. El comando “usorecursos stop” finaliza el guardado de usos de recursos.
7. El comando “sig <tipo> <pid>” envía una señal al proceso <pid> SIGUSR1 si <tipo> es 1 o SIGUSR2 si <tipo> es 2.
8. Se crea un hijo con fork() que ejecuta los comandos con execvp(). Si está activado el comando “usorecursos”, se almacenará el uso de recursos de los comandos introducidos en el archivo <nombre>.
9. La shell soporta pipes infinitas.
10. Si el comando no existe, se indica que no existe y se continúa normalmente.
11. Las señales están implementadas con kill()
12. Al apretar CTRL + C, este preguntará si desea continuar con el programa y activa una bandera. Para leer la respuesta se reusa el getline del main.

Pseudocódigo

```
flagSIGINT ← 0        // variable global

main():
    signal(SIGINT, signal_handler)
    flagRecursos ← 0
    myfile ← archivo usado para guardar los usos de recursos de los comandos

    while 1:
        print(“mini-shell:$ ”);   //sin nueva linea
        linea ← linea del input

        if flagSIGINT:
            if linea == “y” or linea == “Y”: exit(0)
            flagSIGINT ← 0
            continue
        if linea.empty(): continue

        vector<vector<string>> com
        por cada comando en línea:
            insertar en com un vector<strings> que contenga el comando y sus argumentos

        si comando es “exit”: exit(0)
        si comando es “usorecursos start <nombre>”:
            if flagRecursos:
                imprimir “Comando inicializado”
                continue
            flagRecursos ← 1
            myfile ← archivo de nombre com[0][2]
            insertar en myfile ("comando    tuser    tsys    maxrss” )
        `   continue
        si comando es “usorecursos stop”:
            if flagRecursos == 0: imprimir "Comando no iniciado"
            else flagRecursos = 0
            continue
        si comando es “sig <tipo> <pid>”:
            if tipo == 1:  kill(pid, SIGUSR1)
            if tipo == 2:  kill(pid, SIGUSR2)
            continue
```

```
            if fork() == 0:
                ejecutarComandos(com)
                if flagRecursos:
                    vector<string> vaux ← medirRecursos()
                    insertar en myfile (linea y datos de vaux)
                exit(0)

            else wait(NULL)

signal_handler(int sig):
    if sig == SIGINT:
        flagSIGINT ← 1
        print("¿Desea salir de la shell? (Y/N)")


ejecutarComandos(vector<vector<string>> &com):
    pipes ← (com.size()-1) pipes
    pids ← set de pids

    desde i ← 0 hasta com.size():
        myargs ← almacena com[i] en otro formato
        myargs[ com[i].size() ] = NULL

        pid ← fork()
        if(pid == 0){
            if i != 0: dup2(pipes[ (i-1)*2 ], 0)
            if i != com.size()-1: dup2(pipes[i*2 + 1], 1)
            cerrar todas las pipes
            execvp(myargs[0], myargs);
            print("Comando no encontrado")
            exit(0)
        else pids.insert(pid)

    cerrar todas las pipes
    error ← 0;
    desde i ← 0 hasta com.size():
        pids.erase( wait(&error) )
        if error:
            para cada pid p en pids: kill(p, SIGKILL)
            while wait(NULL) != -1:
            break


vector<string> medirRecursos():
    struct rusage r_usage
    getrusage(RUSAGE_CHILDREN, &r_usage)

    tuser ← tiempo de usuario de r_usage
```

```
tsys ← tiempo de sistema de r_usage
maxrss ← maximum resident set de r_usage

return vector<string>{tuser, tsys, maxrss}
```

Conclusión

Como es una shell simple, solo soporta comandos que existan como ejecutables. Por ejemplo: “cd”, “history”, “fg”, etc. no son ejecutables y no se pueden usar en la mini-shell.
El principal problema que tuvimos fue implementar el CTRL+C: Se intentó leer el input desde el signal_handler, lo que causaba bugs en el getline del main. Se arregló implementando una flag y reusando el getline del main para leer la respuesta.