Proyecto 1

Roberto Artigues, Emilio Meza, Nicolás Soto

8 de Octubre de 2023

1. Introducción

Desarrollo de un intérprete de comandos simple en Linux (shell). La shell a implementar será similar a las disponibles actualmente en Linux.

2. Parte 1

La Parte 1 del código se encarga de implementar una shell simple que cumple con los siguientes requisitos:

1. Proporciona un prompt (indicador de modo de espera de comandos) para que el usuario ingrese comandos.
2. Lee el comando ingresado por el usuario desde la entrada estándar y lo parsea para identificar el comando y sus argumentos, con la capacidad de manejar un número indeterminado de argumentos.
3. Ejecuta el comando ingresado en un proceso hijo utilizando las llamadas al sistema `fork()` y `exec()`. La shell espera a que el comando en ejecución termine antes de mostrar nuevamente el prompt.
4. Maneja la entrada en blanco (cuando el usuario presiona ."enter") imprimiendo nuevamente el prompt.
5. Soporta comandos que se comunican mediante pipes (`|`), lo que permite redirigir la salida de un comando como entrada para otro comando.
6. Puede finalizarse si se ingresa el comando `"exit"`.
7. Proporciona un mensaje de error si se ingresa un comando que no existe.

La parte principal del código ejecuta un bucle infinito donde se espera que el usuario ingrese comandos. Luego, se parsea la entrada del usuario, se crean procesos hijos para ejecutar los comandos, y se maneja la comunicación a través de pipes. El código también verifica y ejecuta el comando `start_daemon` con dos argumentos (t y p) para iniciar la Parte 2 del código.

3. Parte 2

La Parte 2 del código se encarga de crear un demonio que mide y registra información del sistema en el archivo de registro del sistema `/var/log/syslog`. Esta parte cumple con los siguientes requisitos:

1. Crea un demonio que recopila información del sistema cada t segundos durante un tiempo total de p segundos.
2. El demonio extrae información como "processes", "procs_runningz "procs_blocked"del archivo `/proc/stat` para sistemas Linux.
3. Utiliza llamadas al sistema como `fork`, `umask`, `setsid`, `openlog`, `syslog`, y `closelog` para configurar y gestionar el demonio.
4. Puede ser detenido mediante el comando `killall` de Linux.

El código utiliza `fork` para crear un proceso hijo que se convierte en un demonio (daemon) utilizando `setsid`. Luego, el demonio recopila información del sistema a intervalos regulares y la registra en el archivo de registro del sistema usando las funciones de registro de `syslog`. El demonio se puede detener utilizando `killall`.

```
void start_daemon(int t, int p) {
    pid_t pid = fork(); // Se crea un proceso hijo
    if (pid == 0) {
        umask(0); // Cambiar la máscara de modo de archivo, para que los archivos creados por el demonio tengan todos los permisos
        setsid(); // Nueva sesión
        close(STDIN_FILENO);
        close(STDOUT_FILENO);
        close(STDERR_FILENO);

        openlog("SusDaemon", LOG_PID, LOG_DAEMON); // Iniciar el log
        syslog(LOG_NOTICE, "El diablillo se ha creado"); // Mensaje de inicio

        for (int i = 0; i < p / t; ++i) {
            syslog(LOG_NOTICE, "Diablillo corriendo por la %d vez", i + 1);
            std::ifstream file("/proc/stat"); // Sacando la info desde /proc/stat porque /proc/cpuinfo no tiene la info de procesos (AL menos en WSL Ubuntu)
            // std::ifstream file("/proc/cpuinfo"); //! Descomentar esta linea para probar con /proc/cpuinfo
            std::string line;
            int processes = 0, procs_running = 0, procs_blocked = 0;

            if (file.is_open()) {
                while (std::getline(file, line)) {
                    if (line.find("processes") != std::string::npos) {
                        std::istringstream iss(line);
                        std::string key;
                        iss >> key >> processes;
                    }
                    if (line.find("procs_running") != std::string::npos) {
                        std::istringstream iss(line);
                        std::string key;
                        iss >> key >> procs_running;
                    }
                    if (line.find("procs_blocked") != std::string::npos) {
                        std::istringstream iss(line);
                        std::string key;
                        iss >> key >> procs_blocked;
                    }
                }
                file.close();
            }

            syslog(LOG_NOTICE, "Procesos: %d, Corriendo: %d, Bloqueados: %d", processes, procs_running, procs_blocked);
            sleep(t);
        }
        syslog(LOG_NOTICE, "Diablillito ha terminado");
        closelog(); // Cerrar el log
        exit(0);
    }
}
```

Figura 1: Funcion Demonio

```
RobLand SusDaemon[16761]: El diablillo se ha creado
RobLand SusDaemon[16761]: Diablillo corriendo por la 1 vez
RobLand SusDaemon[16761]: Procesos: 17114, Corriendo: 3, Bloqueados: 0
RobLand SusDaemon[16761]: Diablillito ha terminado
```

Figura 2: Demonio haciendo sus procesos