Monitor de procesos estilo top

Acceso al sistema y monitoreo

Uriel Felipe Vázquez Orozco, Euler Molina Martínez, Carlos Edwin Bautista Zintzun, Luis Angel Gutierrez Juarez, Alfredo Pineda Prado, Magdalena Reyes Chavez

Septiembre 21, 2025

Monitor de Procesos Estilo TOP

Resumen del Proyecto

Este proyecto implementa un monitor de procesos personalizado similar al comando `top` de Linux, desarrollado en C utilizando la librería ncurses para crear una interfaz de usuario en modo terminal. El programa permite visualizar información de los procesos del sistema de forma interactiva y generar reportes de procesos con mayor uso de memoria.

Arquitectura del Sistema

Componentes Principales

El proyecto está compuesto por los siguientes archivos:

- **`maintop.c`**: Archivo principal que contiene la función `main()` e inicializa el sistema
- **`funTop.c`**: Implementación de todas las funciones del sistema
- **`funtop.h`**: Archivo de cabecera con declaraciones de estructuras y funciones
- **`kbhit.h`**: Implementación de función para detección de teclas presionadas

Compilación

```
gcc maintop.c funTop.c -o pTOP -lncurses
```

Análisis Detallado del Código

1. Archivo Principal (`maintop.c`)

Este archivo contiene el punto de entrada del programa:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ncurses.h>
#include <dirent.h>
#include <unistd.h>
#include "funtop.h"

int main(){
    // Inicialización de ncurses
```

```
    initscr();     // Inicializa pantalla de ncurses
    cbreak();      // Las teclas se leen directamente sin esperar "enter"
    noecho();      // Evita que se imprima lo que se esté capturando
    curs_set(0);   // Oculta el cursor de la terminal

    // Declaración de estructuras de control
    BUFFER bufferControl;  // Para manejar el texto
    SCREEN screenControl;  // Maneja las ventanas
    CLI cliControl;        // Maneja el CLI
    ARCHIVE archive;       // Para manejar directorios
    PROCESS process;       // Para manejar el contenido de cada proceso

    // Función principal de ejecución
    initial(&cliControl, &bufferControl, &screenControl, &archive, &process);

    endwin();  // Finaliza ncurses
    return 0;
}
```

Funcionalidad:

- Configura el entorno ncurses para la interfaz de usuario
- Declara e inicializa las estructuras de datos principales
- Llama a la función `initial()` que controla toda la ejecución del programa

2. Estructuras de Datos (`funtop.h`)

```
typedef struct cli{
    int cliState;           // Estado de la CLI (-1 para salir)
    char cliImput[10];      // Entrada capturada del teclado
    int charsCount;         // Contador de caracteres
    char namearch[100];     // Nombre del archivo a leer
    char message[200];      // Mensaje que se imprimirá
    char currentFile[200];  // Nombre del archivo actual mostrado
    int row;                // Renglón leído (para scroll)
    int maxrow;             // Renglones máximos
    char lines[3000][200];  // Arreglo para copiar el archivo
}CLI;
```

CLI - Controlador de Interfaz de Línea de Comandos

```c
typedef struct bff{
    char c;                 // Carácter auxiliar
    char buffer[250];       // Buffer principal
    int bufcont;            // Contador del buffer
    char parameter1[50];    // Primer parámetro del comando
    char parameter2[50];    // Segundo parámetro del comando
    char command[50];       // Comando principal
    int commandstatus;      // Estado del comando (número de parámetros)
}BUFFER;
```

BUFFER - Manejo de Entrada de Usuario

```c
typedef struct screen{
    WINDOW* wind[5];        // Arreglo de punteros a ventanas
    int windupdate[5];      // Control de actualización de ventanas
}SCREEN;
```

SCREEN - Control de Ventanas

```c
typedef struct archive{
    DIR *dr;                // Directorio abierto
    struct dirent *di;      // Entrada del directorio
    const char *path;       // Ruta del directorio ("/proc/")
}ARCHIVE;
```

ARCHIVE - Manejo del Directorio /proc/

```c
typedef struct process{
    int pid;                // ID del proceso
    char command[256];      // Nombre del comando
    long memorySize;        // Tamaño en memoria (kB)
} PROCESS;
```

PROCESS - Información de Procesos

3. Funciones Principales (`funTop.c`)

Funciones de Inicialización `initializeCLI(CLI *cliControl)`
- Inicializa todos los campos de la estructura CLI a valores por defecto
- Establece el estado inicial del programa

`InitializeBuffer(BUFFER *bufferControl)`
- Limpia el buffer de entrada y sus componentes
- Prepara el sistema para recibir comandos del usuario

`InitializeScreen(SCREEN *screenControl)`
- Crea tres ventanas principales:
- Ventana 0 (3x200): Muestra el buffer de entrada
- Ventana 1 (7x150): Muestra mensajes del sistema
- Ventana 2 (34x100): Muestra contenido de archivos
- Marca todas las ventanas para actualización inicial

`InitializeArchive(ARCHIVE *arch, SCREEN *screen)`
- Abre el directorio `/proc/` para lectura
- Verifica que el directorio sea accesible

Función de Control Principal `initial(...)`
- Función principal que coordina toda la ejecución
- Inicializa todas las estructuras
- Ejecuta el bucle principal hasta que el usuario salga

`cli(...)`
- Maneja la entrada del usuario mediante teclado
- Procesa diferentes tipos de teclas:
- Caracteres imprimibles (32-126)
- Teclas especiales (Backspace, Escape, Enter)
- Teclas de navegación (flechas arriba/abajo)
- Actualiza el buffer y controla el estado del programa

Procesamiento de Comandos `processBuffer(BUFFER *bff)`
- Analiza el contenido del buffer usando `strtok()`
- Separa la entrada en comando y hasta dos parámetros
- Establece el estado del comando según el número de parámetros

`loadComand(...)`
- Ejecuta los comandos reconocidos:
- `exit`: Termina el programa
- `proclist`: Genera lista completa de procesos
- `proc topmem [n]`: Genera lista de top N procesos por memoria

Manejo del Sistema de Archivos /proc/ `isNumber(const char *str)`
- Verifica si una cadena representa un número válido
- Utilizada para identificar PIDs en el directorio /proc/

`getProcessCommand(int pid, char *command, size_t size)`
- Lee el archivo `/proc/[pid]/comm` para obtener el nombre del proceso
- Maneja errores y casos donde el proceso ya no existe

`getProcessMemorySize(int pid)`
- Lee el archivo `/proc/[pid]/status`
- Busca la línea “VmSize:” para obtener el uso de memoria en kB
- Retorna 0 si no puede obtener la información

`writeProcessInfo(...)`
- Genera archivo con lista completa de procesos
- Formato: PID, COMMAND, SIZE(kB)
- Itera sobre todas las entradas en /proc/ que sean números

`writeTopMemoryProcesses(...)`
- Crea un arreglo dinámico de estructuras PROCESS
- Lee información de todos los procesos
- Ordena por uso de memoria usando `qsort()` y `compareProcesses()`
- Escribe los top N procesos al archivo especificado

Funciones de Visualización `windowcontrol(...)`
- Controla qué ventanas necesitan actualizarse
- Llama a las funciones de impresión correspondientes

`printbuffer(...)`
- Muestra el contenido actual del buffer en la ventana superior
- Incluye prompt “->” y cursor visual

`printMessage(...)`
- Muestra mensajes del sistema en la ventana de mensajes
- Utilizada para errores, confirmaciones y estado del programa

`printArchive(...)`
- Función compleja que maneja la visualización de archivos
- Implementa scroll vertical con las teclas de flecha
- Carga archivos en memoria para navegación eficiente
- Muestra información de posición actual en el archivo

4. Detección de Teclas (`kbhit.h`)

```c
int kbhit(void) {
    int ch;
    timeout(0);          // Hace getch() no bloqueante
    ch = getch();        // Intenta leer un carácter
    timeout(-1);         // Restaura comportamiento bloqueante

    if(ch != ERR) {
        ungetch(ch);     // Devuelve el carácter al buffer
        return 1;        // Hay tecla disponible
    }
    return 0;            // No hay tecla presionada
}
```

Funcionalidad:
- Implementa detección no bloqueante de teclas presionadas
- Compatible con ncurses
- Permite verificar entrada de usuario sin pausar el programa

Comandos Disponibles

1. `proclist`

- **Sintaxis**: `proclist`
- **Función**: Genera un archivo llamado “proclist” con información de todos los procesos
- **Contenido**: PID, nombre del comando, uso de memoria en kB
- **Salida**: Archivo de texto plano ordenado por PID

2. `proc topmem [número]`

- **Sintaxis**: `proc topmem [n]`
- **Función**: Genera archivo “topmem” con los N procesos que más memoria consumen
- **Parámetros**:
  - n: Número entero positivo de procesos a mostrar
- **Contenido**: Lista ordenada descendentemente por uso de memoria
- **Salida**: Archivo de texto con formato tabular

3. `exit`

- **Sintaxis**: `exit`
- **Función**: Termina la ejecución del programa
- **Alternativa**: Tecla Escape (ESC)

Navegación del Sistema

Controles de Teclado

- **Caracteres alfanuméricos**: Entrada de comandos
- **Enter**: Ejecutar comando ingresado
- **Backspace**: Borrar último carácter
- **Escape**: Salir del programa
- **Flecha Arriba**: Desplazar hacia arriba en el archivo mostrado
- **Flecha Abajo**: Desplazar hacia abajo en el archivo mostrado

Interfaz de Usuario

La interfaz está dividida en tres áreas principales:

1. **Área de Comando** (superior): Muestra el prompt "->" y el comando siendo escrito
2. **Área de Mensajes** (media): Muestra el estado del sistema, errores y confirmaciones
3. **Área de Contenido** (inferior): Muestra el contenido de los archivos generados con capacidad de scroll

Aspectos Técnicos

Manejo de Memoria

- Utiliza asignación dinámica con `malloc()` para el arreglo de procesos
- Libera memoria apropiadamente con `free()`
- Límite de 1000 procesos para evitar consumo excesivo de memoria

Manejo de Errores

- Verificación de apertura de archivos y directorios
- Validación de parámetros de entrada del usuario
- Manejo graceful de procesos que desaparecen durante la ejecución

Optimizaciones

- Cache de archivos leídos para evitar re-lectura innecesaria
- Actualización selectiva de ventanas solo cuando es necesario
- Scroll eficiente mediante arreglo de líneas en memoria

Compatibilidad

- Compatible con sistemas Linux que tengan el sistema de archivos /proc/
- Requiere librería ncurses instalada
- Diseñado para terminales con capacidades de color y control de cursor

Limitaciones Conocidas

1. **Límite de procesos**: Máximo 1000 procesos pueden ser procesados simultáneamente
2. **Tamaño de archivos**: Los archivos mostrados están limitados a 3000 líneas
3. **Dependencia del sistema**: Requiere acceso al directorio /proc/ (específico de Linux)
4. **Memoria**: Los archivos se cargan completamente en memoria para navegación

Conclusión

Este monitor de procesos representa una implementación sólida que demuestra conceptos importantes de sistemas operativos como:

- Acceso al sistema de archivos del kernel (/proc/)
- Programación de interfaces de usuario en modo terminal
- Manejo de estructuras de datos complejas
- Procesamiento de información del sistema en tiempo real
- Técnicas de ordenamiento y búsqueda

El proyecto proporciona una base excelente para entender cómo funcionan internamente herramientas como `top`, `htop` y `ps`.