Taller de Programación de Sistemas Linux Embebidos – 2025-1S

Docente: Juan Bernardo Gómez Mendoza
Fecha límite de entrega: viernes 23 de mayo de 2025.

Objetivos del Taller

- Fortalecer las competencias de trabajo en equipo y comunicación asertiva en el desarrollo de proyectos en grupos de trabajo.
- Familiarizarse con la herramienta GNU make para automatizar la gestión de compilado y enlazado de proyectos en C y/o C++.
- Aplicar Git para el control de versiones y administración de desarrollo en equipos de trabajo distribuidos.
- Familiarizarse con elementos del desarrollo de software con base en requerimientos y del desarrollo orientado a pruebas (test driven development).
- Fortalecer las competencias relacionadas con la generación de documentación técnica efectiva.

Descripción del proyecto

El proyecto consiste en la construcción de un programa llamado `resource_mon` que muestra por pantalla (CLI) la información relacionada con:

- El nombre y fabricante de la CPU del sistema; el número de núcleos y número de hilos (núcleos reales + núcleos virtuales); y el porcentaje de uso de cada hilo.
- La cantidad de memoria física instalada en el sistema en MB; la cantidad de memoria de intercambio (swap) en MB del sistema; el porcentaje de uso de la memoria física instalada; y el porcentaje de uso de la memoria de intercambio.

El programa se ejecuta hasta que el usuario presione la letra `q`.
La información de uso de la(s) CPU(s) se actualiza una vez cada segundo.
El proyecto se realizará en equipos de **tres o cuatro** integrantes.

Requerimientos

- El programa `resource_mon` debe mostrar la información de nombre y fabricante de la CPU del sistema; el número de núcleos y número de hilos (núcleos reales + núcleos virtuales); y el porcentaje de uso de cada hilo.
- El programa `resource_mon` debe mostrar la cantidad de memoria física instalada en el sistema en MB; la cantidad de memoria de intercambio (swap) en MB del sistema; el porcentaje de uso de la memoria física instalada; y el porcentaje de uso de la memoria de intercambio.
- El programa `resource_mon` debe ejecutarse hasta que el usuario presione la tecla `q`, presione la combinación de teclas de interrupción `Ctrl + c`, o cierre la `tty` en la cual se está ejecutando el programa.
- El programa debe actualizar la información uso de la(s) CPU(s) se actualiza una vez cada segundo.
- El programa debe mostrar la información en la misma posición en la pantalla. Esto quiere decir que no genera nuevas líneas cada vez que se actualice la información de uso de la CPU y la memoria.
- La estructura lógica de las fuentes del programa se divide de la siguiente manera:
  - Las funciones y estructuras de datos relacionadas con la obtención de información de la CPU (características y porcentajes de uso) se consignan en el par de archivos `cpuinfo_manip.h` (definiciones e interfaces) y `cpuinfo_manip.c` (implementación). Se sugiere el uso de los archivos `/proc/cpuinfo` y `/proc/stat` para este fin.
  - Las funciones y estructuras de datos relacionadas con la obtención de información de la memoria (características y porcentajes de uso) se consignan en el par de archivos `meminfo_manip.h` (definiciones e interfaces) y `meminfo_manip.c` (implementación). Se sugiere el uso del archivo `/proc/meminfo` para este fin.
  - Las funciones y estructuras de datos relacionadas con la interacción hombre-máquina (cómo se muestran los datos en la pantalla y cómo se captura la información del teclado) se consignan en el par de archivos `tui.h` (definiciones e interfaces) y `tui.c` (implementación).
  - El archivo `resource_mon.c` contiene el ciclo de ejecución principal del programa, y hace uso de las funciones y estructuras definidas en implementadas en los casos anteriormente descritos.
  - La administración de los procesos de compilado, enlazado y limpiado del programa debe realizarse a través de archivos `Makefile`, para uso con la herramienta `GNU make`.
  - Cada bloque funcional (`cpuinfo_manip, meminfo_manip` y `tui`) tiene un respectivo banco de pruebas, el cual demuestra el funcionamiento apropiado de las estructuras y funciones del bloque.

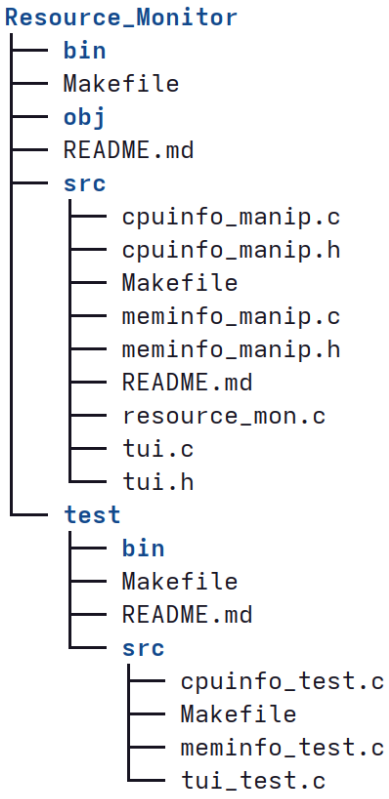

Figura 1. Estructura de archivos de la carpeta del proyecto, previo a la construcción.

Requerimientos - Comportamiento de make:

Al ser ejecutado el comando desde la carpeta principal del proyecto:

- En la carpeta `bin` deben quedar todos los ejecutables que genere `make`.
- El comando `make resource_mon` debe construir el ejecutable `resource_mon` (programa principal).
- El comando `make tests` debe construir los ejecutables `cpuinfo_test`, `meminfo_test` y `tui_test`.
- El comando `make all` debe construir tanto `resource_mon` como los ejecutables de test.
- El comando `make clean` debe borrar todos los archivos compilados `.o`, así como los ejecutables.

Al ser ejecutado desde las subcarpetas:

- El comando `make` en la carpeta `src` debe generar los archivos `.o` de `cpuinfo_manip.c`, `meminfo_manip.c` y `resource_mon.c`.
- El comando `make clean` debe borrar los archivos `.o` generados por `make`.

En cualquier caso, los archivos de extensión `.o` deben quedar en la carpeta `obj`. Cada archivo de extensión `.c` debe compilarse a un archivo de extensión `.o`, previo a la etapa de enlazado.

Requerimientos - Contenido de los archivos README.md:

- Cada archivo `README.md` debe tener la documentación en lenguaje Markdown de la carpeta correspondiente. La documentación debe ser explícita, sobre todo en el archivo `README.md` principal del proyecto.
- El archivo `README.md` de cada subcarpeta debe indicar de qué se trata el contenido de la misma (desde un punto de vista conceptual). También debe indicar qué se obtiene al ejecutar `make` con cada opción posible en dicha carpeta, en el caso de que la carpeta cuente con un archivo `Makefile`.
- El archivo `README.md` de la carpeta `src` debe indicar cuáles son las funciones y estructuras de datos que se proveen en cada uno de los bloques (`cpuinfo_manip`, `meminfo_manip` y `resource_mon`), así como sus parámetros de entrada y lo que proveen.
- El archivo `README.md` de la carpeta `test` debe indicar cuáles son los tests que se implementan en cada uno de los archivos de prueba, y los valores esperados de de cada uno de ellos.

Descripción del entregable

Cada equipo debe entregar los códigos en la estructura mostrada anteriormente, en un repositorio de `git` en un servidor público (Github, Gitlab, otro). El enlace del repositorio debe subirse a través de la plataforma Classroom, en el espacio que se cree para dicho fin.
El archivo `README.md` principal del proyecto debe tener los nombres y números de identificación de los integrantes del equipo.
La verificación del código entregado se realizará en dos pasos:

- Verificación funcional de los requerimientos.
- Verificación de plagio.

En caso de que haya inconvenientes en la segunda fase de verificación, los miembros del equipo serán sujetos a sustentación oral de los códigos entregados.