Assignment_21-08-2024

Author: ThanhTh10
Date: 22/08/2024

1. C/C++ files usage in embedded applications

Data Logging: Storing sensor data or application logs onto an SD card or external memory.
Configuration Files: Reading and writing configuration settings that can be adjusted without recompiling the firmware.
Firmware Updates: Reading binary data from a file to perform over-the-air (OTA) firmware updates.

2. dmesg command in linux OS, How this command works

dmesg Command in Linux OS: The dmesg command is used to display the kernel ring buffer messages in a Linux system. The kernel ring buffer stores messages related to the kernel's operations, such as hardware detection, driver loading, and boot processes. dmesg allows users to view these messages, which can be useful for debugging hardware and kernel-related issues.

How dmesg Works:

- **Kernel Ring Buffer**: The Linux kernel logs messages about system events (e.g., hardware initialization, drivers, errors) to a ring buffer. This buffer is maintained in memory and constantly updated with new log entries.

- **dmesg command**: When the dmesg command is executed, it reads the contents of this ring buffer and displays it. This gives you a snapshot of the latest kernel activity and can provide insights into the system's hardware interactions, boot sequence, and errors.

3. what is dangling pointer

A **dangling pointer** is a pointer that continues to reference a memory location after the memory it points to has been deallocated or freed. This can lead to unpredictable behavior and bugs, as the pointer is pointing to a location that is no longer valid, potentially causing segmentation faults or corrupted data if accessed.
Case:
- **Freeing Memory**: When a block of memory is deallocated (e.g., via free() in C or delete in C++), any pointers referencing that memory become dangling pointers.
- **Out-of-Scope Pointers**: If a pointer refers to a local variable that goes out of scope (e.g., a function ends), the pointer becomes dangling.
- **Reassignment**: If a pointer is assigned to another location without updating the original pointer, the original pointer can become dangling.

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *ptr = (int *)malloc(sizeof(int)); // Allocating memory
    *ptr = 42;  // Assign value
    free(ptr);  // Memory is deallocated, but ptr is now dangling
```

```
    printf("%d\n", *ptr);  // Accessing the dangling pointer, undefined behavior
    return 0;
}
```

Avoid Dangling Pointers:

```
free(ptr);
ptr = NULL;
```

Or we can using smart pointer:

```
#include <iostream>
#include <memory>  // Required for smart pointers

int main() {
    // Create a unique_ptr to manage the memory automatically
    std::unique_ptr<int> ptr = std::make_unique<int>();
    *ptr = 42;  // Assign value
    std::cout << *ptr << std::endl;  // Access value
    // No need to manually free the memory; unique_ptr will automatically
    // release the memory when it goes out of scope
    return 0;
}
```

4. write C program to frame datapacket by using strcture pointer to character buffer

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// Define a structure to represent the data packet
typedef struct
{
    char header[4];  // 4-byte header
    char *data;      // pointer to character buffer for data
    int data_len;    // length of data in bytes
    char trailer[2]; // 2-byte trailer
} DataPacket;ss
// Function to frame a data packet
DataPacket *frame_packet(char *data, int data_len)
{
    DataPacket *packet = (DataPacket *)malloc(sizeof(DataPacket));
    if (!packet)
    {
        printf("Memory allocation failed\n");
        return NULL;
    }
    // Initialize header
    strcpy(packet->header, "HDR ");
    // Set data pointer and length
    packet->data = (char *)malloc(data_len * sizeof(char));
    if (!packet->data)
    {
        printf("Memory allocation failed\n");
        free(packet);
        return NULL;
    }
    memcpy(packet->data, data, data_len);
    packet->data_len = data_len;
    // Initialize trailer
    strcpy(packet->trailer, "TRL");
```

```
    return packet;
}
// Function to print a data packet
void print_packet(DataPacket *packet)
{
    printf("Header: %s\n", packet->header);
    printf("Data: ");
    for (int i = 0; i < packet->data_len; i++)
    {
        printf("%c", packet->data[i]);
    }
    printf("\n");
    printf("Trailer: %s\n", packet->trailer);
}
int main()
{
    char data[] = "Hello, World!";
    int data_len = strlen(data);
    DataPacket *packet = frame_packet(data, data_len);
    if (packet)
    {
        print_packet(packet);
        free(packet->data);
        free(packet);
    }
    return 0;
}
```