**CODE EXPLANATION :**

**static gboolean update\_text\_view(gpointer data) {**

**GtkTextBuffer \*buffer = gtk\_text\_view\_get\_buffer(GTK\_TEXT\_VIEW(text\_view));**

**GtkTextIter end;**

**gtk\_text\_buffer\_get\_end\_iter(buffer, &end);**

**gtk\_text\_buffer\_insert(buffer, &end, (char \*)data, -1);**

**free(data);**

**return FALSE;**

**}**

1. **Getting the Text Buffer:**
   * Think of a GtkTextView **as a box that can hold text. The text buffer is the content inside this box. The line** GtkTextBuffer \*buffer = gtk\_text\_view\_get\_buffer(GTK\_TEXT\_VIEW(text\_view)); **gets that content, allowing us to work with it.**
2. **Creating an "End" Marker:**
   * GtkTextIter end; **creates a marker called "end." This marker is used to show where we're going to put new text inside the text box.**
3. **Moving the Marker to the End:**
   * gtk\_text\_buffer\_get\_end\_iter(buffer, &end); **moves the "end" marker to the actual end of the text that's already in the box. This step is necessary to make sure new text goes at the end and doesn't overwrite existing text.**
4. **Adding New Text:**
   * gtk\_text\_buffer\_insert(buffer, &end, (char \*)data, -1); **adds new text to the buffer at the "end" marker's position. The** data **is a pointer to the text that we want to add, and** -1 **tells the function to add all the text until it finds a null character (end of the string).**
5. **Freeing Memory:**
   * free(data); **releases the memory used by** data**. This is important because if we don't free memory when we're done with it, it could lead to memory leaks, slowing down or crashing the program.**
6. **Returning a Value:**
   * return FALSE; **tells GTK that we're done with this update and don't need it to happen again. If this was part of a loop or repeating function, it would return** TRUE **to keep the updates going.**

**Gboolean is like boolean**

**Gpointer is to point to any pointer in gtk**

**void Spinlock\_lock(Spinlock\* lock, int thread\_id) {**

**while (1) {**

**bool expected = false;**

**if (atomic\_compare\_exchange\_weak(&lock->locked, &expected, true)) {**

**char \*start\_msg = g\_strdup\_printf("Thread %d locked.\n", thread\_id);**

**gdk\_threads\_add\_idle(update\_text\_view, start\_msg);**

**return;**

**}**

**while (atomic\_load(&lock->locked)) { }**

**}**

**}**

### **The Atomic Operation**

* Atomic Compare and Exchange:
  + atomic\_compare\_exchange\_weak(&lock->locked, &expected, true)**: This is an atomic operation that tries to set the lock to** true **(locking it), but only if the current state matches** expected**.**
  + **If** expected **is** false **(meaning the lock is open) and the operation succeeds, the lock is acquired, and the function exits the loop.**
  + **If it fails, it means another thread acquired the lock, so the loop continues.**
* gdk\_threads\_add\_idle **provides a safe way to update the UI asynchronously. It ensures that UI updates happen at the right time and in the correct thread context, preventing potential conflicts or crashes.**

g\_strdup\_printf **is handy for creating formatted strings dynamically in GTK applications, where memory management and string manipulation are common tasks.**

### **Function Name:**

* Name: action\_button

### **Parameters:**

* **\**GtkButton button*: This parameter is a pointer to a** GtkButton **object. In GTK applications, buttons are UI elements that users can click to perform actions. This parameter represents the button that triggered the action.**
* **gpointer data: This parameter is a generic pointer (**gpointer**) used to pass additional data to the function. It can be used to provide context or extra information needed for handling the button click. The type** gpointer **is a generic pointer type used in GLib, which can point to any data type.**

**const char \*old\_pin\_text = gtk\_entry\_get\_text(GTK\_ENTRY(n3));**

**const char \*new\_pin\_text = gtk\_entry\_get\_text(GTK\_ENTRY(n4));**

**const char \*damount\_text = gtk\_entry\_get\_text(GTK\_ENTRY(n6));**

**const char \*wamount\_text = gtk\_entry\_get\_text(GTK\_ENTRY(n7));**

**if(n3){**

**if (old\_pin\_text && strlen(old\_pin\_text) > 0) {**

**old\_pin = atoi(old\_pin\_text);**

**}**

1. **Getting Text from GTK Entry Fields:**
   * gtk\_entry\_get\_text**: This function is used to retrieve the text entered by the user into a GTK entry field, such as a text box.**
   * GTK\_ENTRY**: This is a macro that converts a generic GTK widget pointer to a specific type, in this case, an entry widget.**
   * (n3)**,** (n4)**,** (n6)**,** (n7)**: These are variables representing GTK entry widgets. They likely correspond to fields where users can input text, such as PIN numbers or amounts.**
2. **Storing Text in Variables:**
   * const char \*old\_pin\_text**: This declares a pointer variable** old\_pin\_text **to store the text entered in the entry field** n3**.**
   * **Similarly,** new\_pin\_text**,** damount\_text**, and** wamount\_text **store text from entry fields** n4**,** n6**, and** n7**, respectively.**
3. **Converting Text to Numbers:**
   * old\_pin = atoi(old\_pin\_text)**: After retrieving the text from the entry field** n3**, this line converts it to an integer using the** atoi **function. This assumes that the text entered represents a numerical value (like a PIN code).**
   * **Similarly,** atoi **is used for** new\_pin**,** damount**, and** wamount**, converting the text to integer values for further processing.**
4. **Validation:**
   * strlen(old\_pin\_text) > 0**: Before converting the text to a number, the code checks if the entered text is not empty. If it's not empty, it proceeds with conversion and further processing**
   * **GtkWidget \*window,\*descr,\*layout;**
   * **window = gtk\_window\_new(GTK\_WINDOW\_TOPLEVEL);**
   * **gtk\_window\_set\_title(GTK\_WINDOW(window), "Withdraw Window");**
   * **gtk\_window\_set\_default\_size(GTK\_WINDOW(window), 500, 300);**
   * **g\_signal\_connect(window, "destroy", G\_CALLBACK(gtk\_main\_quit), NULL);.**
5. Declaring Variables:
   * GtkWidget \*window, \*descr, \*layout;**: This declares three variables of type** GtkWidget**, which are used to represent graphical user interface (GUI) elements in GTK applications.**
     1. window **is typically used to represent the main application window.**
     2. descr **and** layout **are used for other GUI elements, such as labels or layout containers.**
6. **Creating a Window:**
   * window = gtk\_window\_new(GTK\_WINDOW\_TOPLEVEL);**: This line creates a new top-level window using the** gtk\_window\_new **function.**
     1. GTK\_WINDOW\_TOPLEVEL **specifies that the window should be a top-level window, which means it's a standalone window and not a child window or dialog.**
   * gtk\_window\_set\_title(GTK\_WINDOW(window), "Withdraw Window");**: This sets the title of the window to "Withdraw Window".**
   * gtk\_window\_set\_default\_size(GTK\_WINDOW(window), 500, 300);**: This sets the default size of the window to 500 pixels wide and 300 pixels high.**
7. **Connecting Signals:**
   * g\_signal\_connect(window, "destroy", G\_CALLBACK(gtk\_main\_quit), NULL);**: This connects the "destroy" signal of the window to the** gtk\_main\_quit **function.**
     1. **When the window is closed, the** gtk\_main\_quit **function is called to quit the GTK main loop and exit the application.**
     2. **This ensures that when the user closes the window, the application terminates gracefully.**
8. **Setting Border Width:**
   * gtk\_container\_set\_border\_width(GTK\_CONTAINER(window), 10);**: This sets the border width of the window to 10 pixels.**
     1. **This adds padding around the contents of the window, creating some space between the edges of the window and its contents.**

**layout = gtk\_layout\_new(NULL, NULL);**

**gtk\_container\_add(GTK\_CONTAINER(window), layout);**

**descr = gtk\_label\_new("Enter Withdraw Amount:");**

**gtk\_layout\_put(GTK\_LAYOUT(layout), descr, 10, 10);**

**n7 = gtk\_entry\_new();**

**gtk\_entry\_set\_max\_length(GTK\_ENTRY(n7), LENGTH1);**

**gtk\_layout\_put(GTK\_LAYOUT(layout), n7, 250, 10);**

**check\_button6 = gtk\_check\_button\_new\_with\_label("Done");**

**gtk\_layout\_put(GTK\_LAYOUT(layout), check\_button6, 250, 50);**

**gtk\_widget\_show\_all(window);**

1. Creating a Layout Container:
   * layout = gtk\_layout\_new(NULL, NULL);**: This line creates a new layout container using the** gtk\_layout\_new **function.**
     1. **A layout container is used to arrange other widgets (such as labels, buttons, or entry fields) within a window or dialog.**
2. **Adding the Layout to the Window:**
   * gtk\_container\_add(GTK\_CONTAINER(window), layout);**: This line adds the layout container to the main window.**
     1. **This ensures that the layout container is displayed within the window, allowing other widgets to be placed inside it.**
3. **Creating Widgets Inside the Layout:**
   * descr = gtk\_label\_new("Enter Withdraw Amount:");**: This line creates a new label widget with the text "Enter Withdraw Amount:".**
     1. **A label widget is used to display static text in the user interface.**
   * gtk\_layout\_put(GTK\_LAYOUT(layout), descr, 10, 10);**: This positions the label widget (**descr**) within the layout container at coordinates (10, 10).**
     1. **The coordinates specify the position of the top-left corner of the widget relative to the layout container.**
   * n7 = gtk\_entry\_new();**: This line creates a new entry field widget (**n7**) where users can input text.**
     1. **An entry field widget allows users to enter text or numerical values.**
   * gtk\_entry\_set\_max\_length(GTK\_ENTRY(n7), LENGTH1);**: This sets the maximum length of text that can be entered into the entry field (**n7**).**
   * gtk\_layout\_put(GTK\_LAYOUT(layout), n7, 250, 10);**: This positions the entry field widget (**n7**) within the layout container at coordinates (250, 10).**
4. **Adding a Check Button:**
   * check\_button6 = gtk\_check\_button\_new\_with\_label("Done");**: This line creates a new check button widget with the label "Done".**
     1. **A check button widget allows users to toggle a binary state (checked or unchecked).**
   * gtk\_layout\_put(GTK\_LAYOUT(layout), check\_button6, 250, 50);**: This positions the check button widget (**check\_button6**) within the layout container at coordinates (250, 50).**
5. **Displaying Widgets:**
   * gtk\_widget\_show\_all(window);**: This line displays all widgets contained within the main window.**
     1. **After creating and positioning the widgets, this function call ensures that they are visible to the user.**

**In summary, these lines of code create and arrange widgets within a layout container, add the layout container to the main window, and then display the widgets within the window.**

****

**You**

**what is layout? explain briefly**

**In the context of the provided code,** layout **refers to a GTK layout container. Here's a brief explanation:**

* **GTK Layout Container:**
  + **A layout container is a widget in GTK (GIMP Toolkit) used to organize and arrange other widgets within a window or container.**
  + **It allows you to specify the position and size of child widgets relative to the layout's coordinate system.**
  + **The layout container does not automatically resize to fit its children; instead, it provides a fixed-size canvas for positioning widgets.**
  + **Widgets added to a layout container are typically positioned using specific coordinates within the layout's coordinate system.**
  + **Layout containers are useful for creating custom user interface layouts, such as arranging widgets in specific patterns or designs.**

**In the provided code:**

* layout = gtk\_layout\_new(NULL, NULL); **creates a new layout container.**
* **Widgets such as labels, entry fields, and buttons are added to this layout container using functions like** gtk\_layout\_put**.**
* **Finally, the layout container is added to the main window using** gtk\_container\_add**.**

**Overall,** layout **serves as a container for organizing and positioning other widgets within the user interface of the application.**