Appendix A:
IPC Mechanism

This appendix gives the format for the IPC messages that the ROUTER, CONTROLLER and URL-RENDERING processes exchange.

1. IPC message formats:

```
typedef struct child_req_to_parent
{
  req_type type;
  child_request req;
}

typedef union
{
  create_new_tab_req new_tab_req;
  new_uri_req uri_req;
  tab_killed_req killed_req;
}child_request;


typedef enum req_type
{
  CREATE_TAB,
  NEW_URI_ENTERED,
  TAB_KILLED,
}req_type;

typedef struct tab_killed_req
{
      int tab_index; // Index of killed-tab
}tab_killed_req;

typedef struct create_new_tab_req
{
      int tab_index; //Index of the newly created tab
}create_new_tab_req;

typedef struct new_uri_req
{
  char uri[512]; // url that the user has entered in the URL region
  int render_in_tab;//tab index where the user wants to render the page
}new_uri_req;
```

APPENDIX B:
Wrapper over GTK+ and WebKit toolkit:

Multi-proc-web-browser uses WebKit open source layout engine for rendering web-pages. It uses GTK+ for basic browser window creation. As the goal of the assignment is to learn about IPC, we attempt to hide the intricacies of the GTK+ and WebKit by providing you a wrapper over these, which you may use to accomplish the desired functionality. Next follows the detailed description of each of the wrapper function:

1.
```
int create_browser (tab_type type,
int tab_index,
void (*create_new_tab_cb)(void),
void (*uri_entered_cb)(void),
browser_window** b_window,
comm_channel channel)
```

   Description: This is the wrapper for creating the browser window for various kinds of tabs. It accepts the following arguments:

   a) `tab_type type`: 'type' is a variable of type '`tab_type`' defined as following.
   ```
   typedef enum tab_type
   {
     CONTROLLER_TAB,
     URL_RENDERING_TAB
   }
   ```
   Depending on the `type` of browser-window you want to create, pass the 'type' parameter. If the `type` parameter is CONTROLLER_TAB, window shown in Fig. 1 will be created; otherwise window shown in Fig. 2 is created.

   b) `int tab_index`: The index of the tab being created. This index (starting from 1) should be incremented every time a tab is created (ROUTER process manages this count). Since CONTROLLER tab is created first, the index for it is 0. For the URL-RENDERING tabs, the index begins from 1.

   c) `void (*create_new_tab_cb)(void)`: This is the callback function, pointer to which you need to pass to '`create_browser()`' function, for invocation when the user clicks the '`create_new_tab`' button in CONTROLLER tab (refer Fig.1). Refer Appendix C for more details on this callback function.

   d) The callback function (whose pointer you pass to '`create_browser()`' function) should have the following prototype:
   Prototype: `void create_new_tab_cb(GtkButton* button, gpointer data);`
   You would notice that there is a difference in the prototype of the pointer to callback function and the actual callback function. You should typecast your callback function (using G_CALLBACK) appropriately in order to avoid any compiler errors (the exact reason for this difference is to accommodate the heterogeneity in various callbacks function associated with different GTK+ event. You may ignore its intricacies for now!).

In nutshell, when the user clicks the 'create_new_tab' button on CONTROLLER tab, the callback function (in this case 'create_new_tab_cb()') which you specify at the time of browser window creation (via 'create_browser()' function call), will be called. Template for this callback function is given in Appendix C.

It should be clear from the description that CONTROLLER tab process receives this event and should send a CREATE_NEW_TAB message (via pipe) to the ROUTER process. The ROUTER process then should then fork a new URL-RENDERING tab process which should create a URL-RENDERING window (shown in Fig.2) and perform the function discussed in section 3.1 (Under CREATE_NEW_TAB IPC message).

e) `void (*uri_entered_cb)(void)`: This is the callback that will get invoked when the user hits the enter key in the 'URL region' of CONTROLLER tab (refer fig. 1). Refer Appendex C for more information on this callback function.

f) `browser_window** b_window`: This is the internal data structure used by the supplied wrapper-code. You should create a pointer to it in your code and initialize it to NULL and pass its address to the 'create_browser()' function for initialization. 'create_browser()' manages the allocation for `b_window`.

g) `comm_channel channel`: This structure holds the pipe descriptors for communication back-and-forth between the child and the parent process. You should pass those pipe descriptors which the ROUTER process creates before forking any child process here.

2. `int render_web_page_in_tab(`
   `int tab_index,`
   `char* uri,`
   `browser_window* b_window)`:
   Renders the web-page specified by the 'uri' in tab 'tab_index'. The function also accepts 'b_window' as a parameter. You will use this function when you want to render the web-page to the user in the specified tab. 'b_window' parameter is populated when you create the browser window via 'create_browser' function call.

3. `void show_browser(void)`:
   Once you are done creating the web browser window, you must call `show_browser()` function to enable GTK wait for events. Failing to call this function would make your window irresponsive to user interactions.

4. `void process_single_gtk_event()`:
   Processes single GTK+ window related event in a non-blocking fashion i.e., if there are no pending events, the call to this function will not cause the current thread to block.

5. `void process_all_gtk_events()`:
   Processes all pending GTK+ window related event. You will invoke this function prior to exiting the child (CONTROLLER/URL-RENDERING tab).

APPENDIX C:
Template for Callback functions

1. Template code for `create_new_tab_cb` callback function:

```
void create_new_tab_cb(GtkButton* button, gpointer data);
{
  if(!data)
    return;
  browser_window* b_window = ((browser_window*)data);
  comm_channel channel = ((browser_window*)data)->channel;
  // Append your code here
  ...
  ...
}
```

The callback function receives two parameters:

i) `GtkButton* button`: Pointer to `GtkButton 'create_new_button'` in the CONTROLLER tab (refer fig. 1). You would not use this parameter in the code you write in the callback function.

ii) `gpointer data`: This is a pointer to the auxiliary data that is passed to this callback function. In the above code it is type-casted to pointer of type `'browser_window'`. You can refer to its definition in `main.h` header file. Here is an abridged description of the fields you *would* use to write your code:

> This is the internal data structure used by the supplied wrapper-code and for most part of your assignment, you will *not* modify any of the parameters of this data-structures. The `'comm_channel'` holds the pipe descriptor that you will create when a new child is forked by the ROUTER (/parent) process. `'create_browser'` function accepts `'comm_channel'` as in input parameter which it copies over to `browser_window.channel` in `'create_browser'` wrapper. At any later point, you can use these descriptors for communication between the CONTROLLER/URL-RENDERING child processes and the ROUTER (/parent) process. Pipes are uni-directional. Hence there is a pipe for ROUTER(/parent) process to send its messages to the child processes and a pipe for CONTROLLER/URL-RENDERING tab child-processes to send its messages to the ROUTER(/parent) process. The parent process keeps track of all the pipe descriptors created so far for communication with various child processes in an array of `comm_channel`. A maximum of UNRECLAIMED_TAB_COUNTER (=10, defined in main.h) tabs can be opened. **Note that this count is inclusive of the closed tabs count.**

2. `uri_entered_cb` callback function: Following is the template code for this callback function.

```
void uri_entered_cb(GtkWidget* entry, gpointer data)
{
```

```
  if(!data)
    return;
  browser_window* b_window     = (browser_window*)data;
  comm_channel    comm_channel = b_window->comm_channel;

  //Retrieves the tab-index where the web-page is to be rendered
  int tab_index = query_tab_id_for_request(entry, data);
  if(tab_index <= 0)
  {
    //error handling
  }

  //Get the uri to render in tab_index URL-RENDERING tab.
  char *uri = get_entered_uri(entry);

  //Append your code here
}
```

CONTROLLER-tab process on receiving this event, sends NEW_URI_ENTERED message to the ROUTER process (via pipe) which routes this message to appropriate child URL-RENDERING process (again via pipes). On receiving this message, the URL-RENDERING tab process, renders the web page via `'render_web_page()'`, discussed in Appendix B, wrapper function call.