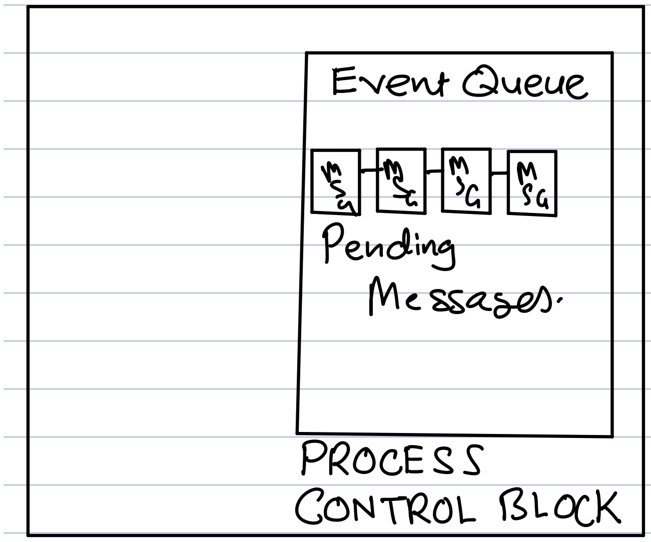
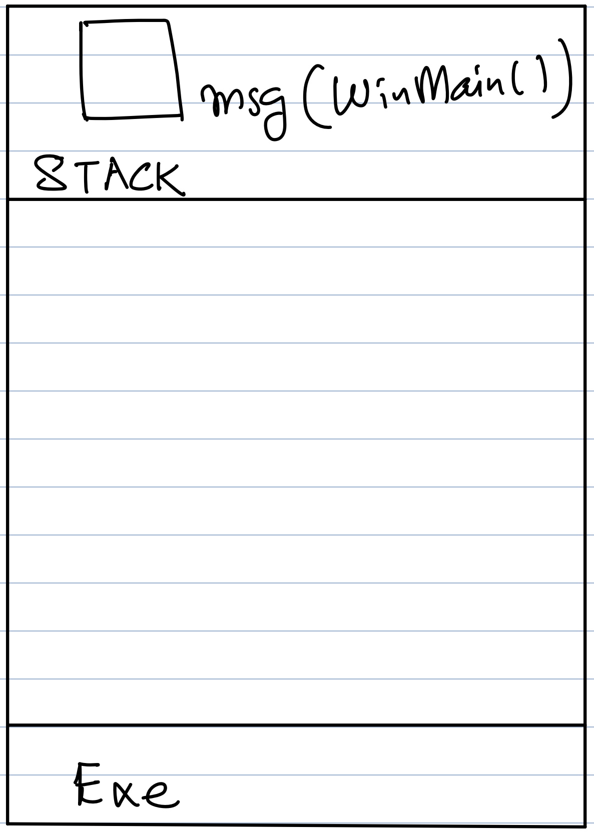


KERNEL ADDRESS SPACE



PROCESS ADDRESS SPACE (GUI PROCESS)

```
MSG msg;

typedef struct tagMSG
.
    PAD
    -
    -
    -
MSG;
```

Information Regarding the event delivered.

End User | event deliver → event process

```
while (GetMessage(&msg, NULL, 0,0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

```
Struct tagMSG
{
        HWND hwnd;
        UINT uMsgType;
        WPARAM wParam;
        LPARAM lParam;
        Time Stamp
        Mouse Co-ordinates. (if applicable)
  };
```

- After UpdateWindow() function, the application window is visible to the end user.
- the end user interacts with the window.

Possible Interactions.

1) Hover the mouse over the client area.
2) Left click on the client area.
3) Right click on the client area.
4) Grab verticle / horizontal resizing border or the corner of window.
5) Release resizing border / corner.
6) Use scroll in upward / downward sense while the window is on the foreground!

7) KB interaction : Press/Release key or key combinations while the window is active

- Every such interaction of end user with application is trapped by an O.S. as a H/W event.
- As a part of event processing an O.S. does the following.

1) Detect an application whose window is interacted with.
2) Detect a handle of a window
(An application may have number of windows and therefore it is important to detect a handle of a window)
3) Detect an event type!
Anything/everything that the end user can do with app. window is already anticipated by the window O.S.
therefore, O.S. is always able to convert <u>EVERY</u> interaction of end user with the window in an appropriate event

type.

HWND hwnd = handle of window on which even ocurred.

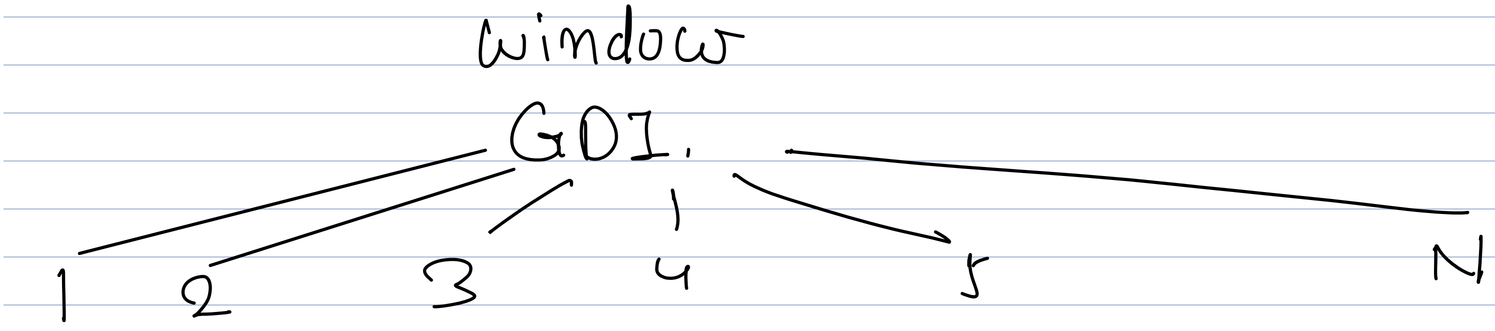
UINT uMsg = Number oj event depicting an event type.

WPARAM = unsigned long long int.

LPARAM = long long int.

Depending on the event type, a data regarding an event is collected.

Window Resize : Window new width
window new height.

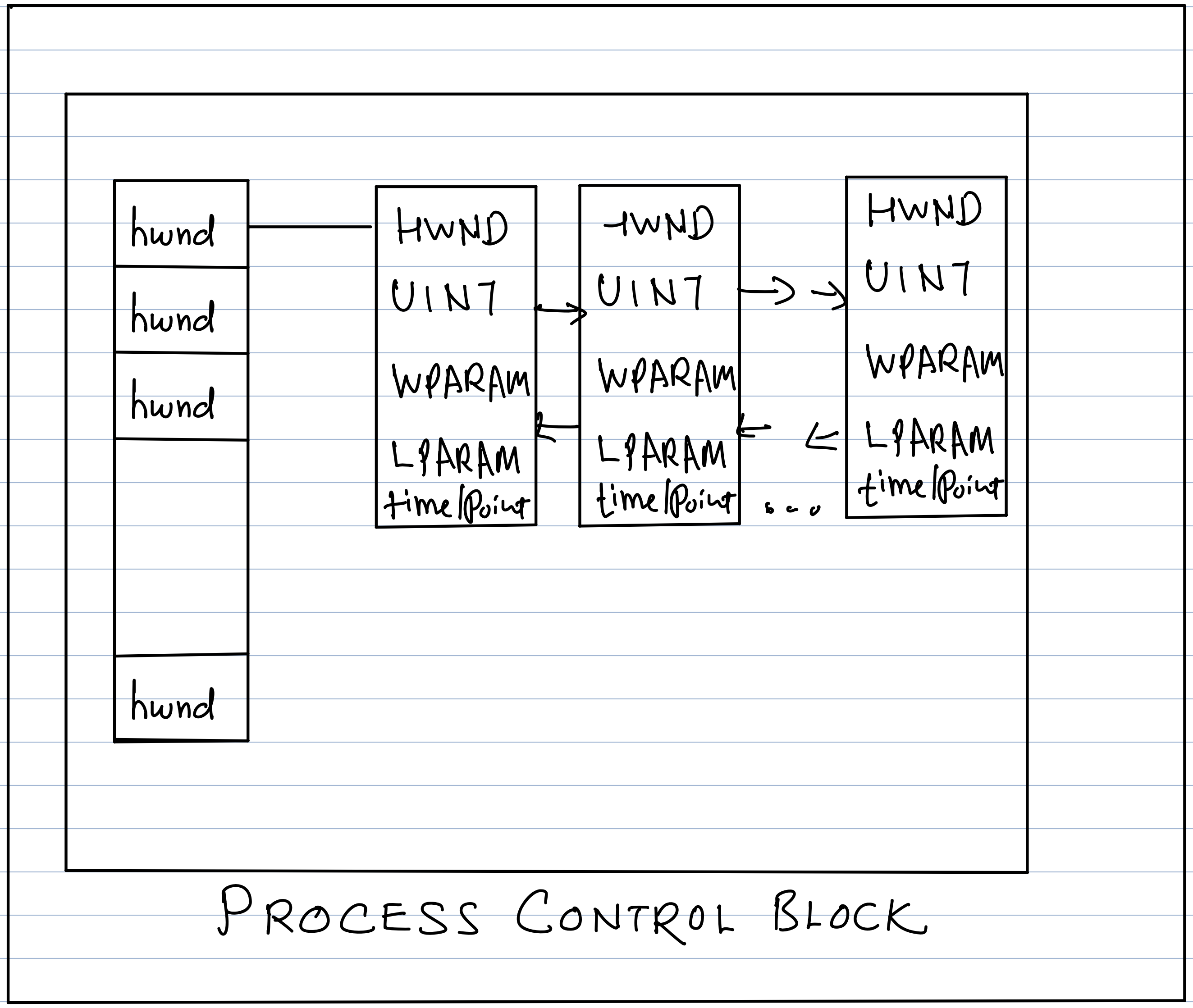


WM_CREAT #define WM_CREAT 1

WM_PAINT WM_DESTROY WM_QUIT

WM_KEYPRESSED WM_LBUTTONDOWN

WM_RBUTTONDOWN 16 byte

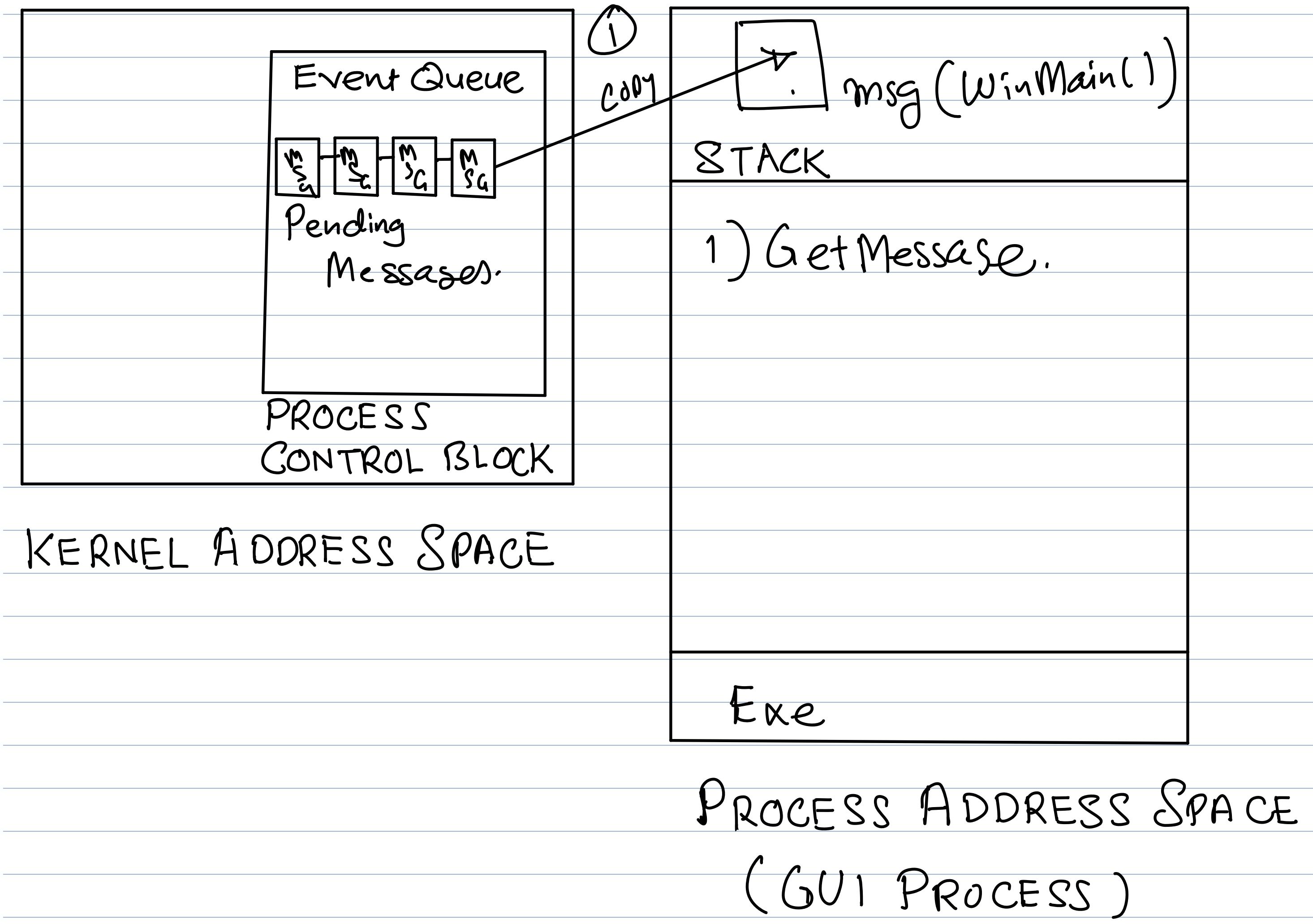


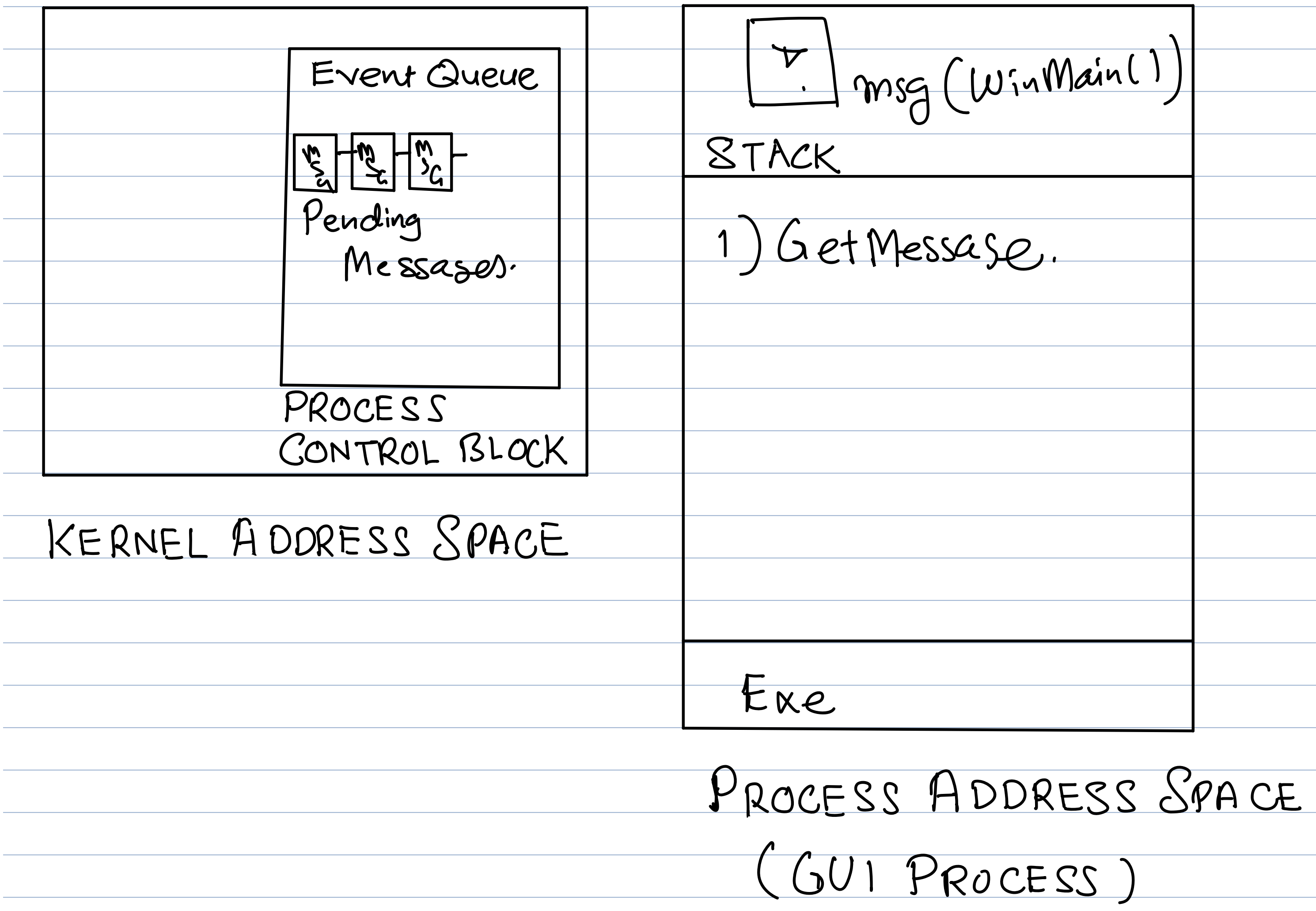
KERNEL ADDRESS SPACE

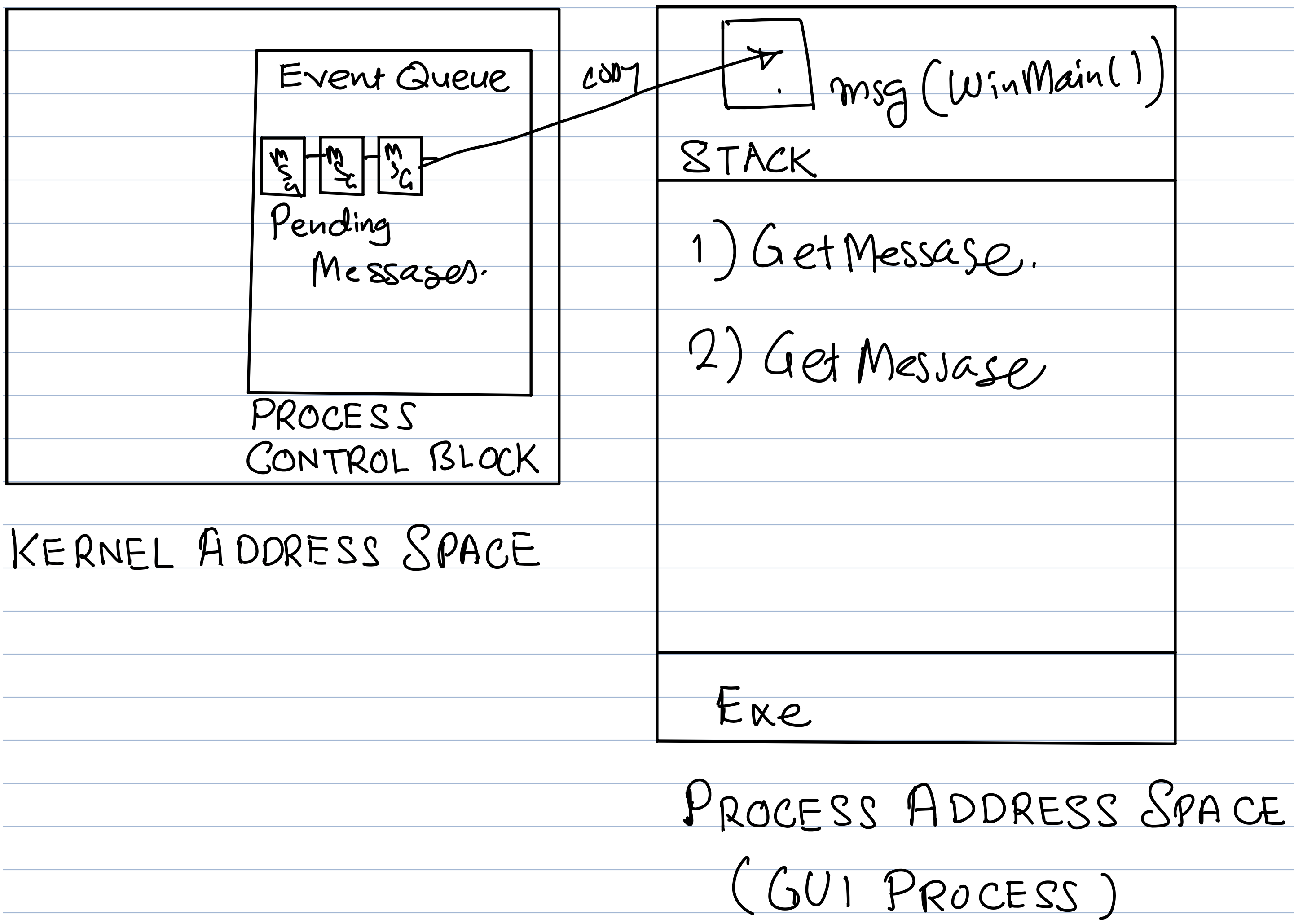
GetMessage(&msg);

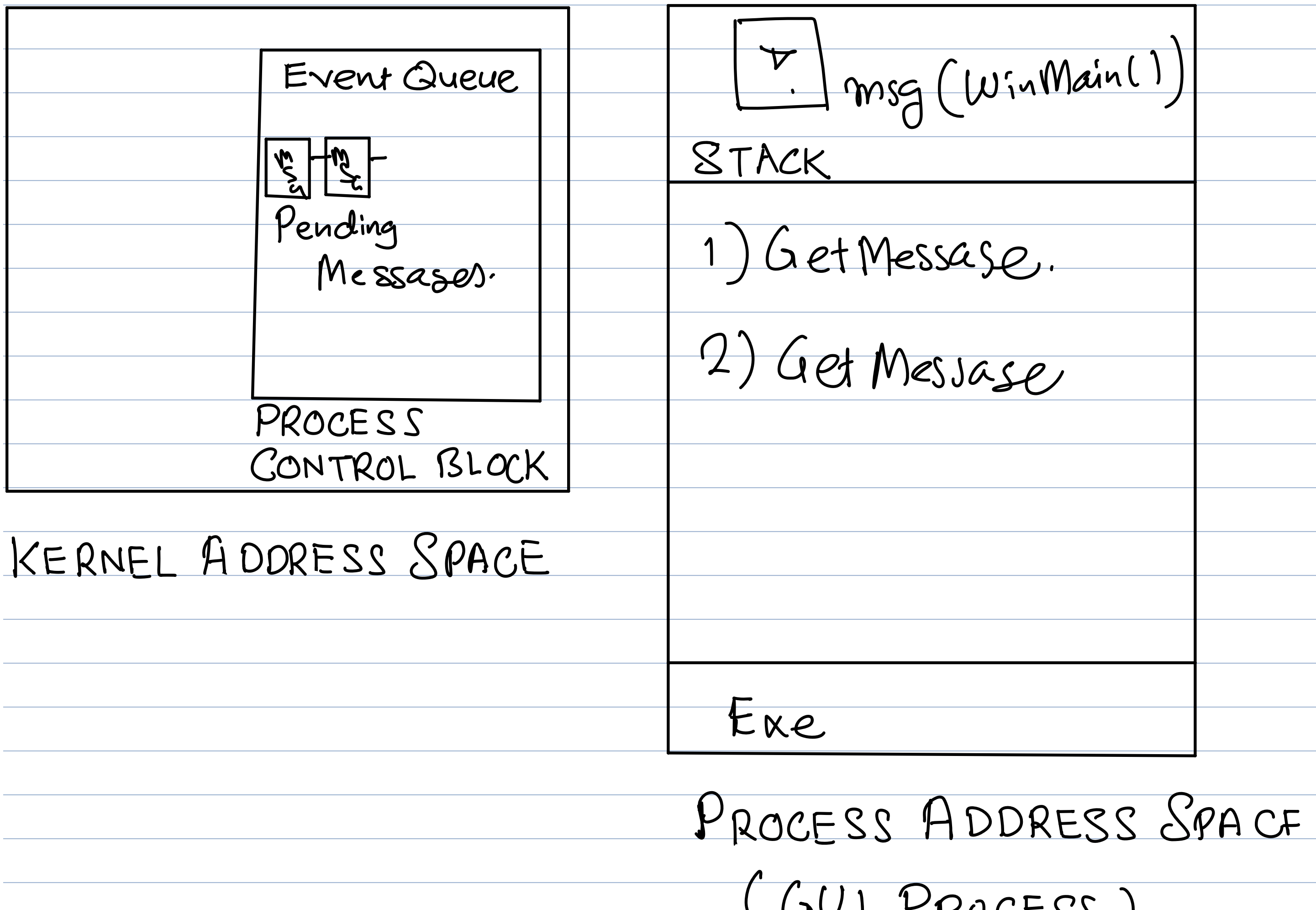
Winuser.h. $\xleftarrow[\text{include.}]{}$ window.h

```
Struct tagMSG.
{


};

HelloWin.C

#include <windows.h>.
     └> winuser.h
          └> Struct tagMSG.

WinMain()
{
     MSG msg;     // Allocate instance of
                  // struct tagmsg in stack.
     ZeroMemory(&msg, sizeof(MSG));



     while(GetMessage(&msg))
```


1
copy
Event Queue
MSG
MSG
MSG
MSG
Pending Messages.
PROCESS CONTROL BLOCK
KERNEL ADDRESS SPACE
msg (WinMain())
STACK
1) GetMessage.
Exe
PROCESS ADDRESS SPACE
(GUI PROCESS)


Event Queue
MSG
MSG
MSG
Pending Messages.
PROCESS CONTROL BLOCK
KERNEL ADDRESS SPACE
msg (WinMain())
STACK
1) GetMessage.
Exe
PROCESS ADDRESS SPACE
(GUI PROCESS)


Event Queue
msg
msg
msg
Pending Messages.
PROCESS CONTROL BLOCK
KERNEL ADDRESS SPACE
copy
msg (WinMain())
STACK
1) GetMessage.
2) Get Message
Exe
PROCESS ADDRESS SPACE (GUI PROCESS)


Event Queue
msg
msg
Pending Messages.
PROCESS CONTROL BLOCK
KERNEL ADDRESS SPACE
msg (WinMain())
STACK
1) GetMessage.
2) Get Message
Exe
PROCESS ADDRESS SPACE
(GUI PROCESS)

(OUT PROCESS)

while(GetMessage(&msg, - -)).

1) GetMessage()
if the event queue is empty
then block until it becomes non-empty.
(at least one pending event)

2) if the event queue is not-empty.,
take the first event on the event
queue, copy paste its contents
into MSG Structure instance provided
by programmer.

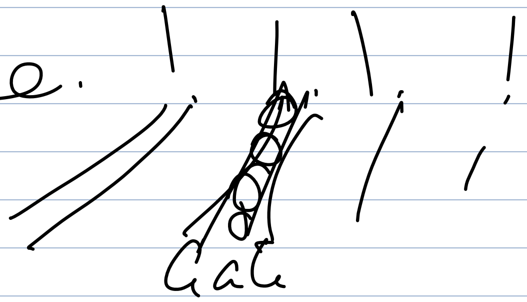
3) GetMessage() removes the copy of
MSG in event queue.

GetMessage(&msg, NULL, 0, 0)

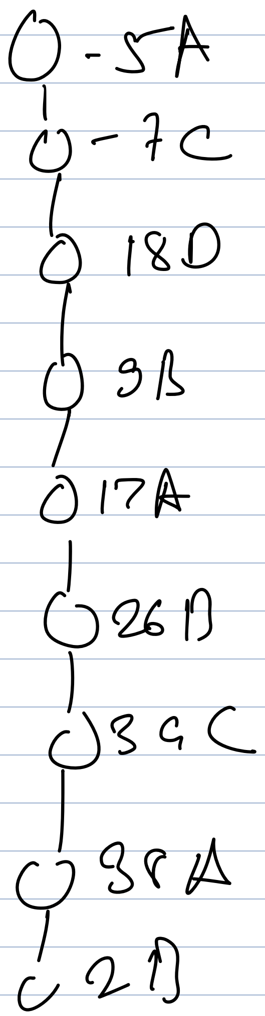
```
Bool GetMessage( PMSG , HWND,
                 UINT, UINT );
```

```
                              min   max
GetMessage(&msg, NULL, 0, 0)
```

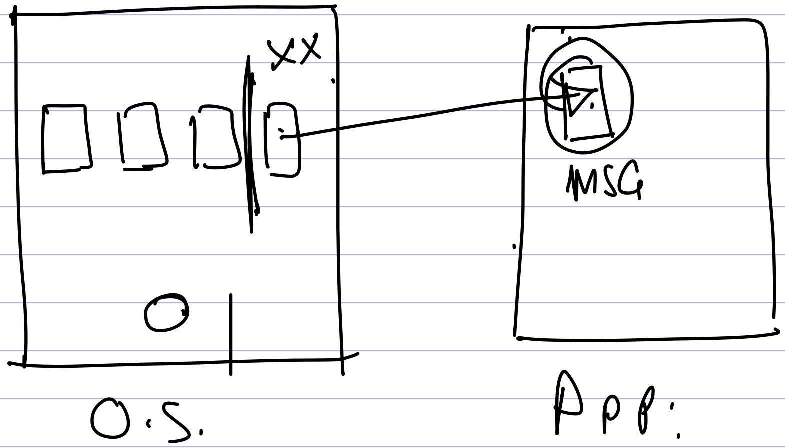
Message id's range.

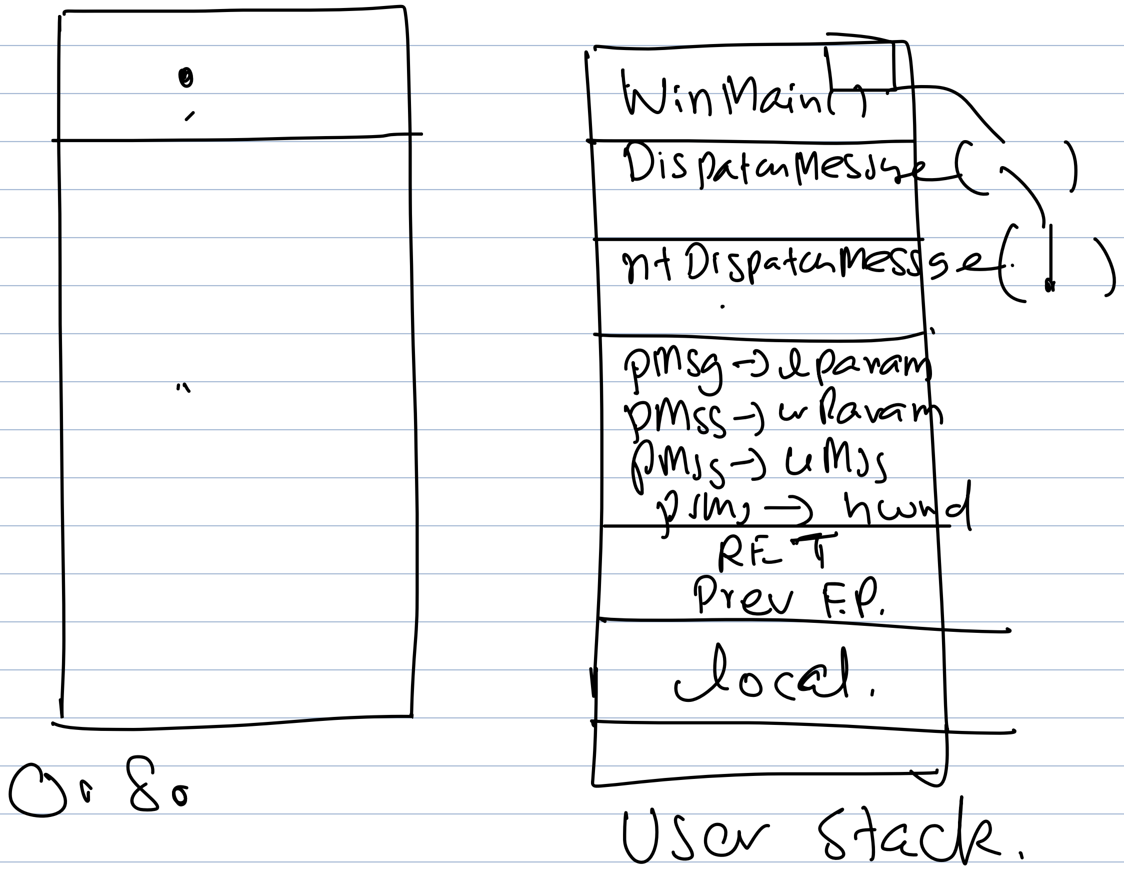


15 to 35



```
while (GetMessage( &msg, NULL, 0, 0 ) )
{
        TranslateMessage(&msg);
        DispatchMessage (&msg).
}.
              ↓ WndProc
```





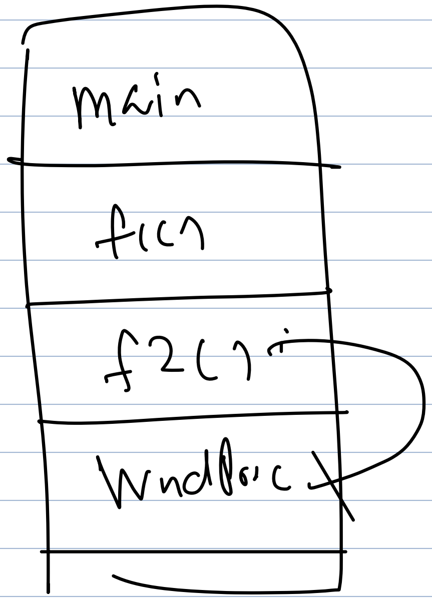
pMsg.

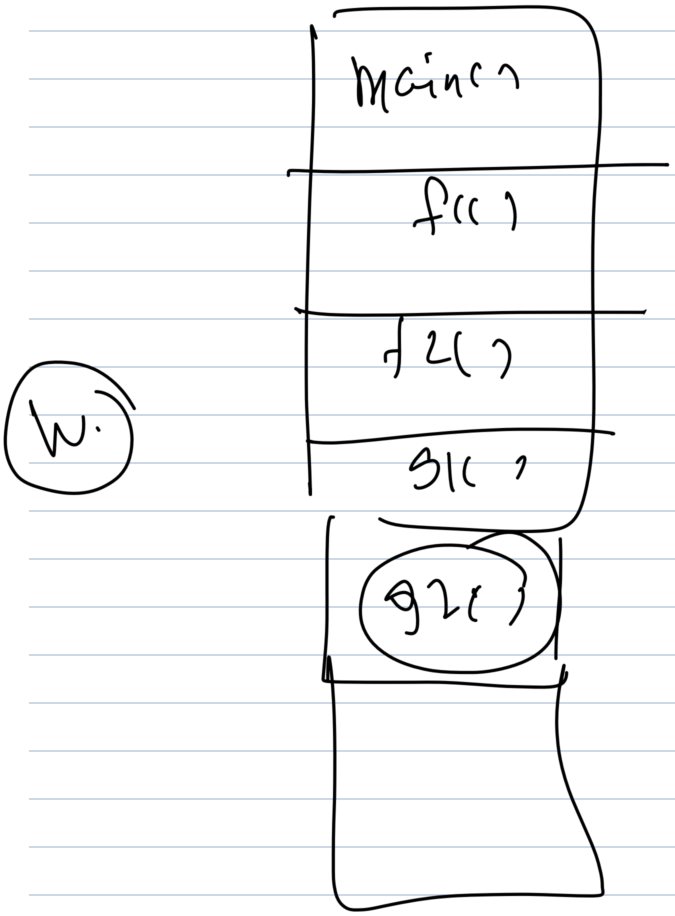
```
DispatchMessage(struct tagMSG*)
               (PMSG pMsg)      &msg
```

```
pMsg -> hwnd

pMsg -> uMsg
pMsg -> wParam
pMsg -> lParam
```

```
DispatchMessage(&msg)

pWnd->lpfnWndProc(pMsg->hwnd,
                  pMsg->uMsg,
                  pMsg->wParam,
                  pMsg->lParam
                 );
```

WndProc()


main
f1()
f2()
WndProc


main()
f1()
f2()
W.
g1()
g2()