Projekt 2

Multimaus

BFH-TI – Berner Fachhochschule

Student: Rico Bachmann – bachr2

Betreuer: Andreas Danuser - dsa2

Sourcecode des Projekts:

https://github.com/i3luefirech/mmP2bachr2

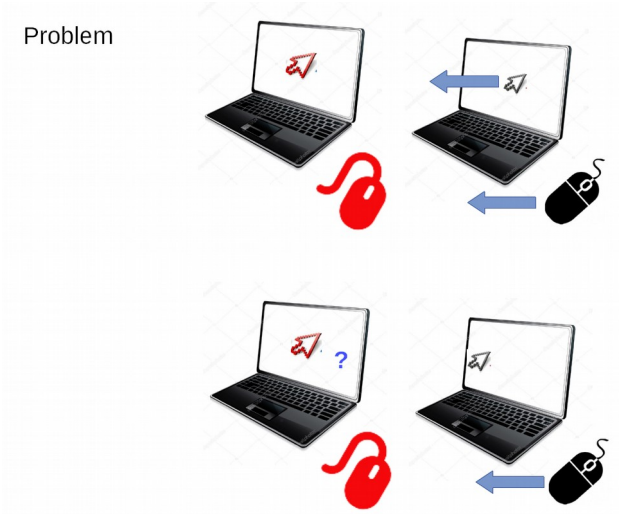
Online-Doku des Projekts:

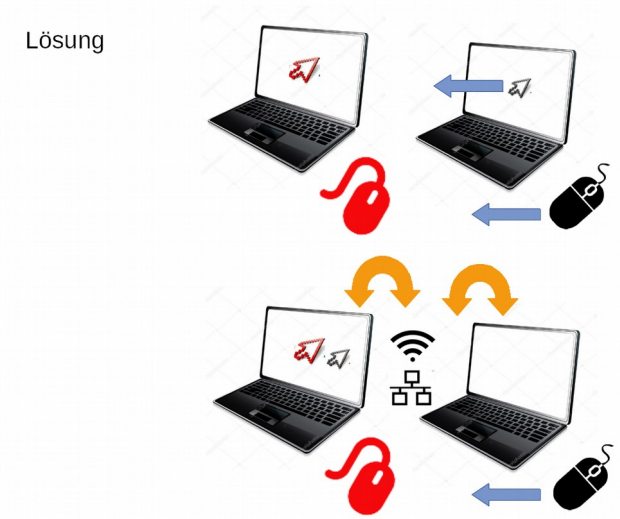
https://github.com/i3luefirech/mmP2bachr2/wiki

Table of Contents

Einführung & Zusammenfassung

Sie kennen das Problem. Man sitzt direkt nebeneinander, alle Personen an Ihrem Laptop. Dann will man seinem Laptopnachbar nur schnell zeigen wie etwas funktioniert. Noch mit der eigenen Maus in der Hand, beginnt man die nötigen Bewegungen zu vollführen. Doch egal wie oft man die Bewegung wiederholt, der Mauszeiger auf dem Nachbarbildschirm verändert sich nicht. Nach kurzem Nachdenken bemerkt man, dass der Mauszeiger des Nachbarlaptops sich ja nicht bewegen kann, solange man die eigene Maus bewegt. Die Idee der Multimaus/Multimouse ist geboren





In diesem Projekt wird ein "Proof of Concept" gemacht, ob die nötigen Informationen zur Umsetzung einer Multimouse vorhanden sind und ob es technisch möglich ist mehr als eine Maus zu rendern. Es wird versucht auf allen gängigen Betriebssystemen die globalen Mausevents abzufangen. Dabei wird sich auf wichtige Mausevents beschränkt wie Bewegung, links & Rechtsklick und das Mausrad

Aufgabenstellung

PoC

Bei diesem Projekt handelt es sich um ein "Proof of Concept" Projekt

Was will ich herausfinden

Es wird versucht herauszufinden ob es technisch möglich ist, die nötigen Informationen der Maus auf allen Betriebssystemen zu erhalten.

- Erhalte globale Mausevents (z.B. Klicks, Bewegung, Scroll, etc.)
- Setze richtigen Mauszeiger auf eine bestimmte Positionen und führe eine Mausfunktion aus
- Ausblenden des richtigen Mauszeigers (evtl. rendern mehrerer Falscher Mauszeiger)

Problemanalyse

Betriebssystem

Grundsätzlich gehe ich von drei Gruppen von Betriebssystemen aus, welche sich beim Einsatz von c++ über Precompiler #ifdef Kommandos unterscheiden lassen, dabei handelt es sich um folgende Gruppen:

- Windows
- Linux
- OSX

Es geht darum herauszufinden, ob die gewünschten Informationen und Funktionen der Maus über eine OS-Schnittstelle angesteuert werden können. Ich werde versuchen den ganzen Ablauf zu abstrahieren, so dass sich der Code so weit als möglich für alle Systeme brauchen lässt. Unterliegend wird der spezifische Code via #ifdef Kommandos unterteilt.

Windows

Windows ist ein geschlossenes Betriebssystem, hat aber inzwischen sehr starke und vor allem gut Dokumentierte Schnittstellen und APIs. Hier wird vermutlich die Schwierigkeit darin liegen, alle globalen Events zu erhalten, da sich normale Maushandler eigentlich nur auf ein Fenster beziehen

Linux

Linux ist ein offenes Betriebssystem. das hat den Vorteil, das sich viel Code einsehen lässt. Das Problem mit Linux ist, das nicht jede Distribution den selben Fenstermanager verwendet und die meisten verwendeten Fenstermanager sind sehr restriktiv was die Datenweitergabe angeht, dass heisst, es muss wohl eine Stufe tiefer auf die Mausevents, etc. zugegriffen werden

OSX

OSX ist ein geschlossenes System und mehrheitlich darauf ausgelegt, dass es mit den von Apple gepushten Programmiersprachen verwendet wird, für c++ werden keine mächtigen APIs angeboten, oder ich konnte die nicht ausfindig machen. Die grösste Herausforderung hier wird es sein zu schauen, wie weit sich der Linux-teil auf OSX portieren lässt und ob alle nötigen Libraries sich binden lassen. Notfalls wird der OSX Teil fallen gelassen

Wichtige Mausevents & Funktionen

- Linksklick
- Rechtsklick
- Scroll up/down
- Mausposition
- Mausbewegung

Die aufgelisteten Events sollten auf allen Systemen erkannt werden können. Sicher möchte ich das ganze unter Linux und Windows betreiben können, wenn der Aufwand nicht zu gross ist, sollen die Events und Funktionen auch unter OSX getestet werden.

Ausblenden der Maus

Der original Mauszeiger muss ausgeblendet werden können. Es wird sich herausstellen ob dies von den Beitreibsystemen eher einfach angeboten wird, oder ob sich da eher ein "Hack" anbahnt. Ich vermute aber, dass dies machbar sein sollten, da auch viele Vollbildanwendungen keinen Mauszeiger vom OS erscheinen lassen, auch wenn die Vollbildanwendung nicht im Fokus ist.

Rendern mehrerer Mäuse

Ich kann mir vorstellen, dass dies eine der grössten Herausforderungen wird. Mit meinem Momentanen Wissensstand, kann ich noch nicht genauer sagen wo hier die Schwierigkeiten liegen. Falls sich das Problem zum darstellen von mehreren Mauszeigern nicht einfach Lösen lässt, so gibt es immer noch die Möglichkeit, den original Cursor zu "Multiplexen" und ihn schnell zwischen den verschiedenen Positionen hin und her zu switchen. (Dieser Switch Vorgang ist sonst nur Kurzfristig zur Ausführung von Aktionen nötig)

Lösungsansatz

Windows

Globale Mausevents

Die Mausevents konnten wie geplant über einen einfachen Hook erledigt werden. Dieser Lösungsansatz hat von Anfang an geklappt.

```
DWORD CALLBACK Thread(LPVOID pVoid)
{
    // activate hook
    hook=SetWindowsHookEx(WH_MOUSE_LL, LowLevelMouseProc, hInst, 0);
    Mainmouse->run();
    UnhookWindowsHookEx(hook);
    return 0;
}
```

Die einzigen Schwierigkeiten welche aufgetaucht sind waren das umziehen des Testcodes aus der main Funktion in eine eigene Klasse. Zum teil mussten statische und globale Helfer-Funktionen eingesetzt werden.

```
LRESULT CALLBACK LowLevelMouseProc(int nCode,WPARAM wParam,LPARAM lParam)
{
    Mainmouse->LowLevelMouse(nCode,wParam,lParam);
    return (::CallNextHookEx(hook,nCode,wParam,lParam));
}
```

Mausfunktion

Die Mausfunktionen liessen sich auch mehr oder weniger "straight forward" implementieren. Es kamen drei Techniken zum Einsatz:

SetCursorPos der Windows API, wurde verwendet um die Maus zu positionieren

```
void realmouse::setpos(int x, int y) {
    SetCursorPos(x,y);
}
```

SendMessage von der Windows API, wurde verwendet um Clicks zu emulieren

```
void realmouse::doleftclick() {
    POINT pos_cursor;
    HWND hwnd_outra_win;
    GetCursorPos(&pos_cursor);
    hwnd_outra_win = WindowFromPoint(pos_cursor);

SendMessage(hwnd_outra_win,WM_LBUTTONDOWN,MK_LBUTTON,MAKELPARAM(pos_cursor.x,pos_cursor.y
));
    SendMessage(hwnd_outra_win,WM_LBUTTONUP,0,MAKELPARAM(pos_cursor.x,pos_cursor.y));
}
```

SendInput von der Windows API, wurde verwendet um das Scrollrad zu emulieren

```
void realmouse::doscrolldown() {
    INPUT in;
    in.type = INPUT_MOUSE;
    in.mi.dx = 0;
    in.mi.dy = 0;
    in.mi.dwFlags = MOUSEEVENTF_WHEEL;
    in.mi.time = 0;
    in.mi.dwExtraInfo = 0;
    in.mi.mouseData = (DWORD)-1;
    SendInput(1,&in,sizeof(in));
}
```

Ausblenden & Rendern

Das ausblenden des globalen Mauszeigers unter Windows stellt sich als viel schwieriger da als erwartet. Laut API lässt sich der Cursor ganz einfach mit folgendem Befehl verstecken:

```
ShowCursor(hide);
```

Leider funktioniert dies nur im eigenen Fenster und wenn es den Fokus hat. Online wurde öfter empfohlen es zu Versuchen, indem zuerst dem aktuellen Fenster unter dem Cursor die Capion gegeben wird. Aber auch dieser Code führe zu keinem Erfolg!

Das Verstecken des Cursors ist somit nicht ganz einfach zu implementieren. Es gäbe meiner Meinung nach 3 Mögliche Lösungsansätze für die Bachelorarbeit:

- Ein unsichtbares Fenster über den gesamten Bildschirm
  - Pro: Einfaches Mauscursor Handling
  - Contra: Alle Events müssen richtig an die unterliegenden Fenster weitergegeben werden
- Denn Cursor im unteren rechten Bildschirmrand verstecken
  - Pro: Einfache zu implementieren zum Verstecken
  - Contra: Komplizierteres Handling wenn der Cursor in Aktion treten muss
- Denn Cursor nicht verstecken, eine Maus kontrolliert immer den original Cursor
  - Pro: Einfacheres switchen bei Mausaktionen für den richtigen Cursor
  - Contra: Komplizierteres Handling wer jetzt wann den Maincursor erhält

Für das Rendern eines eigenen Cursors hat die Zeit leider nicht mehr gereicht. Dies sollte aber wirklich keine zu grosse Herausforderung darstellen

Linux & OSX

Globale Mausevents

Leider konnte ich keine praktische Verbreitete API für Linux finden um die Mausevents global abzufangen. Dadurch blieb mir eigentlich nur die Variante eine Stufe tiefer, unter dem Fenstermanager, nach einer Lösung zu Suchen. Hierzu muss direkt auf das Devicefile zugegriffen werden. Dies geschieht über einen normalen Filehandler

```
void *Thread(void *pVoid) {
    if((hook = open(MOUSEFILE, O_RDONLY)) == -1) {
        std::cout << "can not open mousefile..." <<  std::endl;
        return (void *)-1;
    }
    struct thread_data *my_data;
    my_data = (struct thread_data *) pVoid;
    Mainmouse->run();
    close(hook);
    pthread_exit(nullptr);
}
```

Mausfunktion

Zum setzen des Cursors wird im Moment noch der laufende XWindow Server verwendet

```
void realmouse::setpos(int x, int y) {
    Display *dpy;
```

```
    Window root_window;
    dpy = XOpenDisplay(0);
    root_window = XRootWindow(dpy, 0);
    XSelectInput(dpy, root_window, KeyReleaseMask);
    XWarpPointer(dpy, None, root_window, 0, 0, 0, 0, x, y);
    XFlush(dpy);
}
```

Unter Linux gibt es keinen unterschied zwischen einem Button und dem Scrollrad, scrolle up/down werden einfach als gedrückte Buttons Übertragen:

```
void realmouse::doleftclick() {
    Display *display = XOpenDisplay(NULL);
    XEvent event;
    if(display == NULL)
    {
        fprintf(stderr, "Error!!!\n");
        exit(EXIT_FAILURE);
    }
    event.type = ButtonPress;
    event.xbutton.button = 1;   // left button, 1 = left, 2 = middle, 3 = right, 4 =
scrolle up, 5 = scrolle down

    event.xbutton.same_screen = True;
    XQueryPointer(display, RootWindow(display, DefaultScreen(display)),
&event.xbutton.root, &event.xbutton.window, &event.xbutton.x_root,
&event.xbutton.y_root, &event.xbutton.x, &event.xbutton.y, &event.xbutton.state);
    event.xbutton.subwindow = event.xbutton.window;
    while(event.xbutton.subwindow)
    {
        event.xbutton.window = event.xbutton.subwindow;
        XQueryPointer(display, event.xbutton.window, &event.xbutton.root,
&event.xbutton.subwindow, &event.xbutton.x_root, &event.xbutton.y_root,
&event.xbutton.x, &event.xbutton.y, &event.xbutton.state);
    }
    if(XSendEvent(display, PointerWindow, True, 0xfff, &event) == 0) fprintf(stderr,
"Error!!!\n");
    XFlush(display);
    usleep(10000);
    event.type = ButtonRelease;
    event.xbutton.state = 0x100;
    if(XSendEvent(display, PointerWindow, True, 0xfff, &event) == 0) fprintf(stderr,
"Error!!!\n");
    XFlush(display);
    XCloseDisplay(display);
}
```

Ausblenden & Rendern

Für das Ausblenden & Rendern eines eigenen Cursors hat die Zeit leider nicht mehr gereicht. Dies sollte aber wirklich keine zu grosse Herausforderung darstellen

Fazit

Allgemein

Das Projekt hat sehr gut gestartet. Ich konnte viele schnelle Erfolge verzeichnen. Einfache Mousclicks simulieren, oder das Auslesen eines auf die SW bezogenen Maushandlers klappten nahezu auf Anhieb. Leider habe ich durch diese Anfangseuphorie den restlichen Aufwand etwas unterschätzt und bin zum Schluss noch in Zeitnot geraten. Für ein nächstes mal habe ich gelernt, dass auch nach einem guten Start nicht einfach knapper geplant werden darf!

Informationen

Die Suche nach Informationen zu globalen Mausevents hat sich als extrem schwierig herausgestellt. Dies könnte daran liegen, dass mit globalen Mausevents und Keyevents sehr einfach das ganze Verhalten des Benutzers aufgezeichnet werden kann, hier ist zu erwähnen das in diesem PoC-Projekt keine Daten gespeichert werden.

Globale Mausevents

Ausser Windows, welches den LowLevel-Zugriff zu den Mausevents über einen ziemlich einfach Hook erlaubt, muss bei den anderer Systemen direkt via InputDevice File gearbeitet werden. Dadurch wurde das Coden aufwändiger als erwartet. Evtl. würde es hier Sinn machen auf ein Framework wie Qt zurückzugreifen, aber auch hier müsste Abgeklärt werden, ob dies wirklich auf allen Systemen globale Mausevents erlaubt, ohne Fenster.

Gewählter Lösungsweg

Ich würde wohl schon wieder einiges anders gestalten am Aufbau dieser SW. Es wäre sicher eine optimalere Abtrennung von OS spezifischem Code und von reusable Code machbar. Bei einem nächsten solchen Projekt würde ich jetzt auch von beginn an sauber mit Klassen arbeiten. Der "quick and dirty" Code in meinem Lokalen repository war zwar praktisch für schnelles "Try and Error", doch leider hat es mich am Schluss nochmals extrem viel Zeit gekostet, denn Code in saubere Klassen zu porten.

Aussicht – Bachelorarbeit

Auf Aussicht auf die Bachelorarbeit für die Multimaus/Multimouse bin ich sehr optimistisch. Vom technischen Standpunkt her, wie auch von meinen Möglichkeiten, sollte sich dieses Projekt gut umsetzten lassen (Ausser einigen Workarounds beim Maus "verstecken"). Ich glaube es würde Sinn machen beim Maus "verstecken" die Variante zu wählen in welcher der original Cursor am unteren rechten Bildschirmrand versteckt wird, bis er eine Aktion tätigen muss. Ich mache mir ein wenig Sorgen, das ganze Projekt gleich für alle OSs zu realisieren. Evtl. würde sich eine erste saubere Implementation nur für Windows oder Linuxclients besser eignen um nicht den Rahmen der Arbeit zu sprengen.