学習駆動なので全力で学習してみた件

学習駆動コース 坂井ゼミ 村上和馬

一 動機

大学入学！！

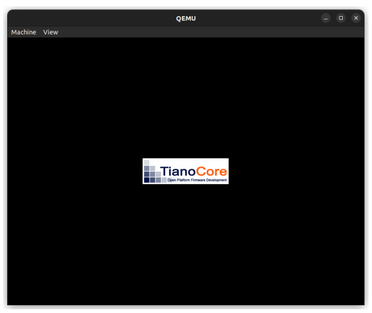
授業でLinux!

Linux
なんもわからん…

自分でOS作れば
全部わかるのでは！？

二 過程

7月



EDK2を使ってローダを作る.

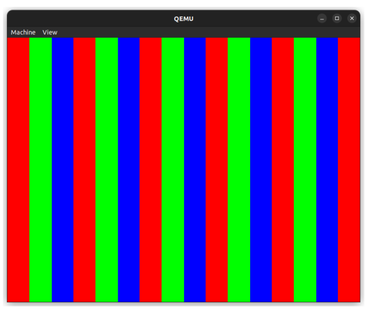
ブロトス

BrotOS

GitHub : https://github.com/yakishamo/BrotOS

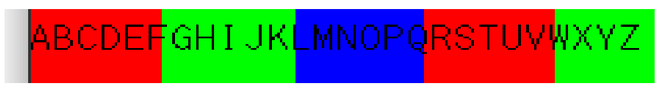
broto…ポルトガル語で「つぼみ」
brotosで複数形
（MandelbrotOS）

8月



自分で簡単なカーネルを作って, それを起動させることに成功.

←カーネルが起動した証拠に画面が三色で塗りつぶされている.



文字列を表示することに成功
（フォントはMikanOSから拝借）.

```
ReadData : 0
ReadData : -1
ReadData : -1
ReadData : -1
ReadData : -1
ReadData : -1
ReadData : -1
ReadData : -1
ScanAllBus: 0
invdev = 7
```

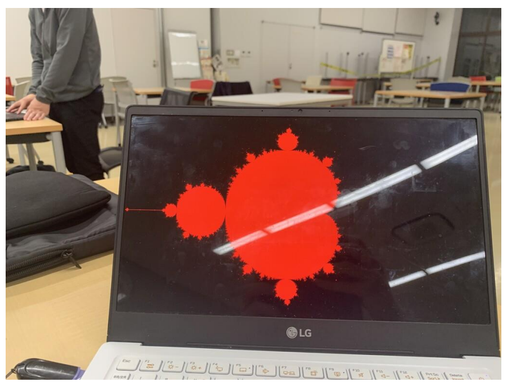
Cライブラリが使えないことに気づき自作ライブラリを試みる. しかし, sprintfを作るときに可変長引数が実装できず失敗.

←文字列処理を書いたので数値を文字列として表示できている.

9月 夏休み

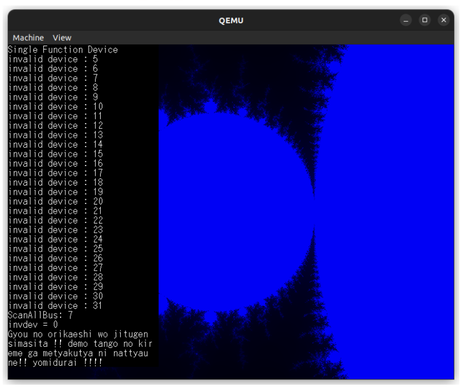
10月

マンデルブロ集合を描画する.



←サークルの人の制作物を見て思いつき, 自分のOSにも取り込もうとしたら, あっという間にできてしまったので実機で実演.

10月



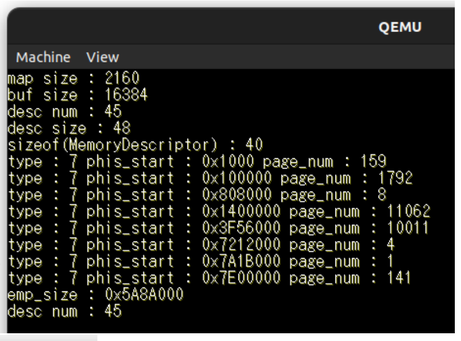
表示する文字列のスクロール, 折り返しの機能を実装. ついでにUSBを使いたいためPCIデバイスをスキャン.

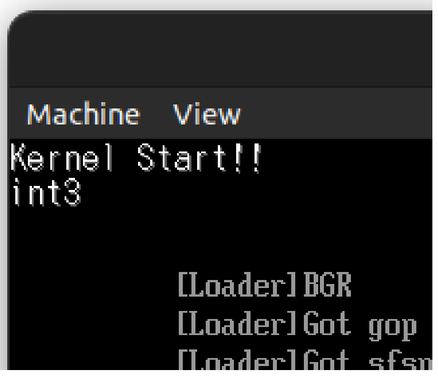
←「行の折り返しを実現しました!! でも単語の切れ目がめちゃくちゃになっちゃうね!読みづらい!!!!」と一番下にローマ字で書いている.

11月

メモリ管理を作り始める.

→UEFIからメモリマップを得て, それを表示している. これを元に空き領域を判断して, その空き領域に対して管理を行う処理を作る.





以下の割り込みを実装
- Break Point
- General Protection
- Page Fault
- Segmentation Fault

←表示されている文字列の通りカーネルが起動した直後にBreakPointの割り込みを起こし, その割り込みを処理した証拠として"int3"と表示している.

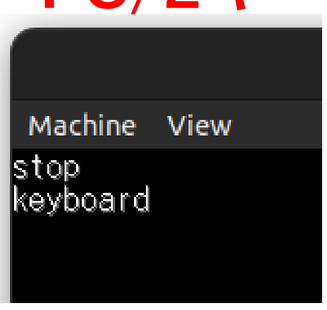
自作Cライブラリは断念し, newlibとlibc++を導入.

```
extern "C" int Printf(const char *format, ...) {
```

↑ライブラリのprintfを使おうとするもメモリ管理が要求されたので断念. 代わりにカーネル用にsprintfを使って文字列処理をして画面に表示するエセprintfを作成.

12月

PS/2キーボードに対応.

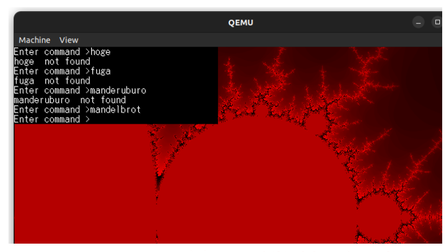


USBキーボードは困難であったため断念.

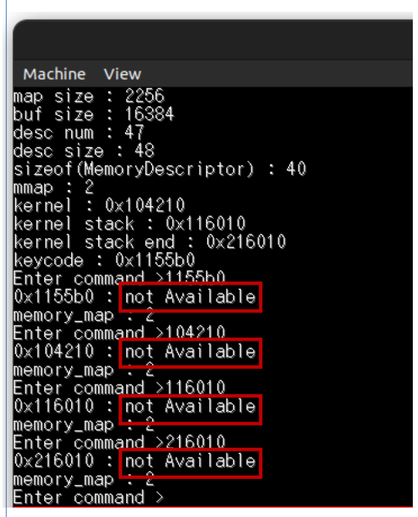
←初めてキーボードによる割り込みを受け取った様子. 割り込み処理の中で"keyboard"と表示している.

コマンドラインを実装.

→BrotOS最初のコマンドである"mandelbrot"を実行してマンデルブロ集合を描画している様子.



1月



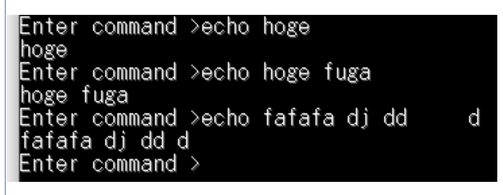
メモリ管理機能完成.
また, UEFIのメモリマップによるとカーネルを配置しているはずのアドレスが空き領域になっていたため, ローダを変更してカーネル領域が使用中となるように.

←カーネルは0x100000に配置されているが, エントリポイント（0x104210）などもしっかり保護されている.

←実は11月のところを見ると, 0x100000はメモリタイプが7（EfiConventionalMemory）となっており, 空き領域として扱われている.

2月

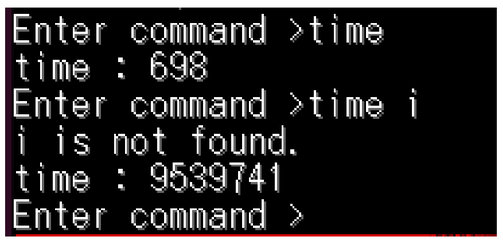
コマンドが引数を取れるように.



←スペースが複数あってもトークンを解析できている.

タイマー実装.

タイマーを使うコマンドとしてtimeコマンドを実装. 引数としてコマンド名とそのコマンド名に対する適切な引数をまとめてとり, そのコマンドを実行するのにかかる時間を計測する.



↑コマンドが存在しないことを確認するだけでtimeコマンド単体と比較して1万倍の実行時間がかかっていることがわかった.

三 得たもの

OSに対する理解！

デバイス管理
例えば…
- 画面の表示
- マウス入力
- キーボード入力
- 音の出力

などなど

メモリ管理
UEFIからメモリマップを受け取ってから…
- セグメントの設定
- ページングの設定
- アプリにメモリを提供

などなど

アプリ管理
- アプリを読み込んでメモリ上に展開
- システムコールに答える
- 命令実行権限の設定
- 割り込みなどの通知

などなど

Twitter : @y_saionji_tech

Hatena Blog: https://y-saionji-tech.hatenablog.com/

四 今後の展望

大学入学時には, OSが何者でなにをしているものなのか全く知らない状態だったが, 一年間SecHack365の中の取り組みとして自作OSをすることで左のような理解を得た.

自作OSが低レイヤー学習において強力なものであることがわかった. しかし周りを見渡しても低レイヤーを触っている人は少ない. この原因に低レイヤーのハードルの高さというのがあると, 自身が自作OSをしていて実感した. 具体的には, エラーを調べても英語のサイトに飛ばされる, そもそもエラーが出ない場合が多いので自身に知識や経験がなければ原因を推測することができないなどのことだ.

この問題の解決を目指してこのSecHack365の期間で行った自作OSを体験記のようにしてブログに事細かに書いたり, 本にして出すなどしてトラブルシューティング的に読めるものを出したい. 低レイヤーの知識は脆弱性などに対する深い理解につながるため, セキュリティについて勉強したい人の一助になればと思う.