



**POLITECHNIKA  
RZESZOWSKA**  
im. IGNACEGO ŁUKASIEWICZA



**WYDZIAŁ  
ELEKTROTECHNIKI  
I INFORMATYKI**  
POLITECHNIKI RZESZOWSKIEJ



**Katedra Informatyki i Automatyki**

**Studenckie Koło Naukowe Informatyków „KOD”**

**Dokumentacja MicrOSa**

Opiekun: dr. inż Bartosz Trybus

Wersja: 1.0

Twórcy dokumentu:

Paweł Osikowski, Paweł Miękina

Rzeszów, 2021

# Spis treści

<b>Wstęp</b>	<b>4</b>
<b>1. Opis projektu</b>	<b>5</b>
<b>2. Funkcjonalności</b>	<b>6</b>
2.1. Oprogramowanie	6
2.1.1. cat	6
2.1.2. help	6
2.1.3. ls	7
2.1.4. mkdir	8
2.1.5. reboot	9
2.1.6. rename	9
2.1.7. rm	9
2.1.8. shell	10
2.1.9. shutdown	10
2.1.10. snake	11
2.1.11. space	11
2.1.12. tasks	12
2.1.13. time	13
2.1.14. touch	13
2.2. Sterowniki	14
2.2.1. Prosty sterownik VGA	14
<b>3. Opis struktury projektu</b>	<b>16</b>
3.1. Podział plików	16
3.2. Oprogramowanie	17
3.3. Jądro	18
<b>4. Implementacja</b>	<b>19</b>
4.1. Oprogramowanie	19
4.2. Sterowniki	19
4.2.1. Prosty sterownik VGA	19
<b>5. Przygotowanie środowiska programistycznego</b>	<b>20</b>

5.1. Przygotowanie oprogramowania . . . . .	20
5.1.1. Edytor . . . . .	20
5.1.2. Emulator procesora . . . . .	20
5.1.3. Montowanie dyskietki . . . . .	20
5.1.4. Środowisko kompilacyjne . . . . .	20
5.2. Konfigurowanie środowiska przy użyciu MicrOS DevTools na Windows . . . . .	20
5.2.1. Instalacja starszej wersji GDB . . . . .	23
5.3. Konfigurowanie środowiska przy użyciu MicrOS DevTools na Linuxie lub WSL . . .	23
5.3.1. Problem z połączeniem GDB pod WSL2 . . . . .	24
5.4. Kompilacja cross-kompilatora GCC . . . . .	24
5.4.1. Pobranie źródeł . . . . .	24
5.4.2. Pobranie dodatkowych zależności i skompilowanie binutils . . . . .	25
5.4.3. Pobranie dodatkowych zależności i skompilowanie GCC . . . . .	25
5.4.4. Znane problemy . . . . .	26
<b>6. Ogólne zalecenia dotyczące kodu źródłowego . . . . .</b>	<b>28</b>
6.1. Pliki . . . . .	28
6.1.1. Przedrostki . . . . .	28
6.2. Struktury . . . . .	28
6.3. Funkcje . . . . .	29
6.4. Zmienne . . . . .	30
<b>7. Programowanie . . . . .</b>	<b>31</b>
7.1. Dostępne standardowe pliki nagłówkowe . . . . .	31
7.2. Dyrektywy preprocesora podczas pisania plików nagłówkowych biblioteki . . . . .	31
<b>Zakończenie . . . . .</b>	<b>33</b>
<b>Literatura . . . . .</b>	<b>34</b>
<b>Spis rysunków . . . . .</b>	<b>35</b>
<b>Spis tablic . . . . .</b>	<b>36</b>
<b>Spis listingów . . . . .</b>	<b>37</b>

## Wstęp

Dokument ten stanowi dokumentację MicrOSa oraz opisuje kwestie techniczne związane z programowaniem systemu MicrOS. Zawiera opis konfiguracji środowiska oraz obsługi mechanizmów potrzebnych do programowania.

## 1. Opis projektu

MicrOS to 32-bitowy system operacyjny, opracowany przez członków Sekcji Aplikacji Desktopowych Mobilnych i Webowych. Naszym głównym celem było stworzenie od podstaw OS’a posiadającego możliwości podobne do systemu operacyjnego MS-DOS.

System działa w 32 bitowym trybie chronionym. Co daje możliwość uruchomienia wielu procesów, z których każdy ma swoją pamięć. Aktywne jest również stronicowanie. System pozwala na interakcję z użytkownikiem w postaci terminali, jednak zawiera również obsługę trybów graficznych. Możliwy jest zapis plików na dyskietce i dysku twardym. Własny bootloader dał również możliwość poznania sposobu rozruchu komputera, a zarazem systemu operacyjnego. Obsługuje przerwania, pamięć fizyczną i wirtualną, procesy i wątki, a także wiele innych.

Powstał on jako projekt edukacyjny, na którym moglibyśmy poznać sposób działania sprzętu i implementacji wielu elementów systemu operacyjnego. Nie tworzyliśmy go z myślą o podboju rynku systemów operacyjnych, czy wypełnianiu jakiejś niszy.

Jako architekturę wybraliśmy x86, z uwagi na jej obecną popularność oraz ustandaryzowaną obsługę sprzętu, która pozwala na proste zarządzanie zasobami, bez konieczności implementacji skomplikowanych sterowników dla urządzeń różnych producentów. Nie celowaliśmy tutaj w architektury innego typu jak ARM, z uwagi na to, że prościej było o standardowy PC oraz podstawowa obsługa sprzętu na poziomie DOSa jest prostsza na x86, właśnie z uwagi na proste i dostępne standardy. Dodatkowo x86 zawiera FPU, które na ARM jest opcjonalne na niektórych procesorach, a znacznie ułatwia pisanie biblioteki standardowej[1].

## 2. Funkcjonalności

Rozdział ten będzie zawierał opis funkcjonalności MicroSa z punktu widzenia użytkownika.

### 2.1. Oprogramowanie

Ten rozdział opisuje dostępne programy pod MicroSem.

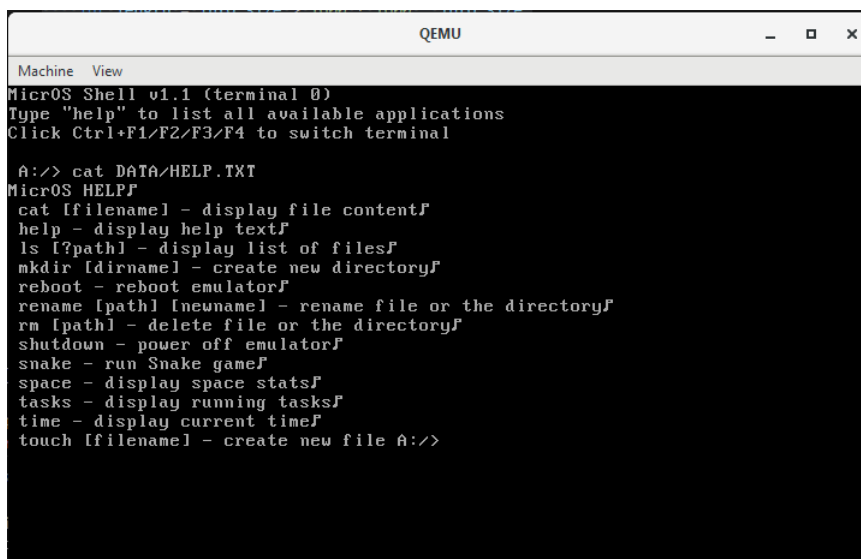
#### 2.1.1. cat

Program wywołuje się w następujący sposób:

```
cat plik
```

Służy do wyświetlania plików tekstowych. Jeśli plik jest większy niż 1600 znaków, cat, wyświetli tylko 1600 pierwszych znaków z komunikatem „First 1600 bytes of the file:”.

Rysunek 2.1 przedstawia przykładowe użycie programu.



```

QEMU
Machine View
MicroS Shell v1.1 (terminal 0)
Type "help" to list all available applications
Click Ctrl+F1/F2/F3/F4 to switch terminal

A:/> cat DATA/HELP.TXT
MicroS HELPF
cat [filename] - display file content
help - display help text
ls [path] - display list of files
mkdir [dirname] - create new directory
reboot - reboot emulator
rename [path] [newname] - rename file or the directory
rm [path] - delete file or the directory
shutdown - power off emulator
snake - run Snake game
space - display space stats
tasks - display running tasks
time - display current time
touch [filename] - create new file A:/>

```

Rysunek 2.1: Przykładowe użycie programu cat

#### 2.1.2. help

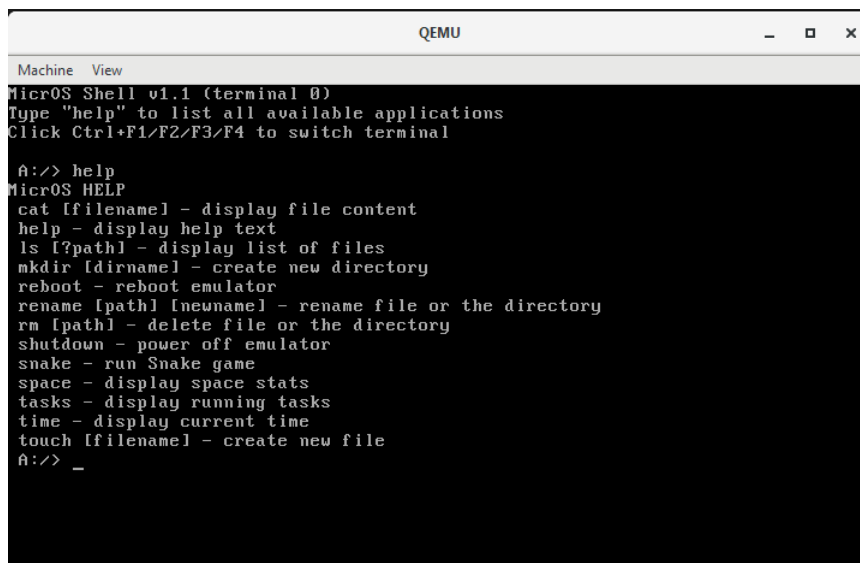
Program wywołuje się w następujący sposób:

```
help
```

Służy do wyświetlenia zawartości pliku pomocy, znajdującego się w katalogu /DATA/HELP . TXT. Jeśli pliku tego nie ma wypisywany jest tekst „No HELP.TXT file”. Plik szukany jest na aktualnie

wybranej partycji, więc działa to tylko na partycji systemowej. Można również umieścić różne pliki help na różnych partycjach.

Rysunek 2.2 przedstawia przykładowe użycie programu.



```
QEMU
Machine View
MicroOS Shell v1.1 (terminal 0)
Type "help" to list all available applications
Click Ctrl+F1/F2/F3/F4 to switch terminal

A:~> help
MicroOS HELP
cat [filename] - display file content
help - display help text
ls [path] - display list of files
mkdir [dirname] - create new directory
reboot - reboot emulator
rename [path] [newname] - rename file or the directory
rm [path] - delete file or the directory
shutdown - power off emulator
snake - run Snake game
space - display space stats
tasks - display running tasks
time - display current time
touch [filename] - create new file
A:~> _
```

Rysunek 2.2: Przykładowe użycie programu help

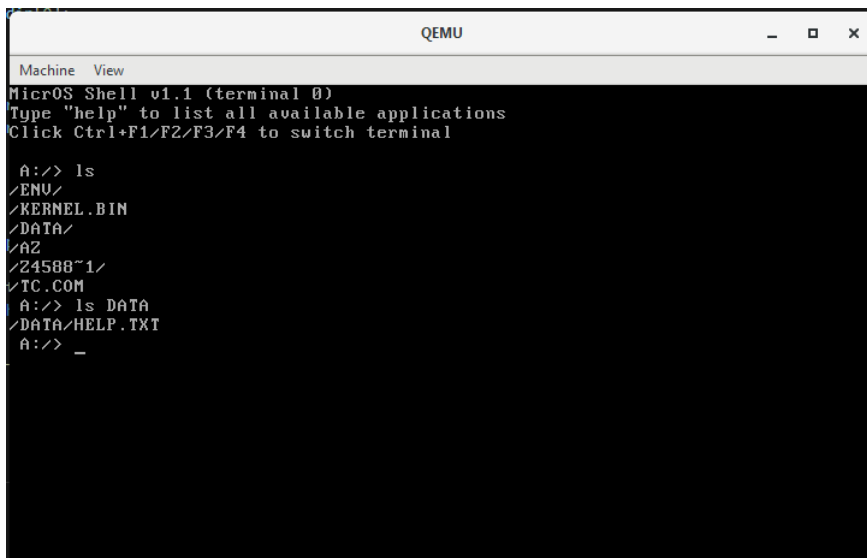
### 2.1.3. ls

Program wywołuje się w następujący sposób:

```
ls [folder]
```

Program wyświetla zawartość obecnego folderu lub wskazanego przez użytkownika. Aktualnie nie posiada żadnych dodatkowych parametrów.

Rysunek 2.3 przedstawia przykładowe użycie programu.



```

QEMU
Machine View
MicroOS Shell v1.1 (terminal 0)
Type "help" to list all available applications
Click Ctrl+F1/F2/F3/F4 to switch terminal

A: /> ls
/ENU/
/KERNEL.BIN
/DAŹA/
/AZ
/24588~1/
/TC.COM
A: /> ls DAŹA
/DAŹA/HELP.TXT
A: /> _

```

Rysunek 2.3: Przykładowe użycie programu `ls`

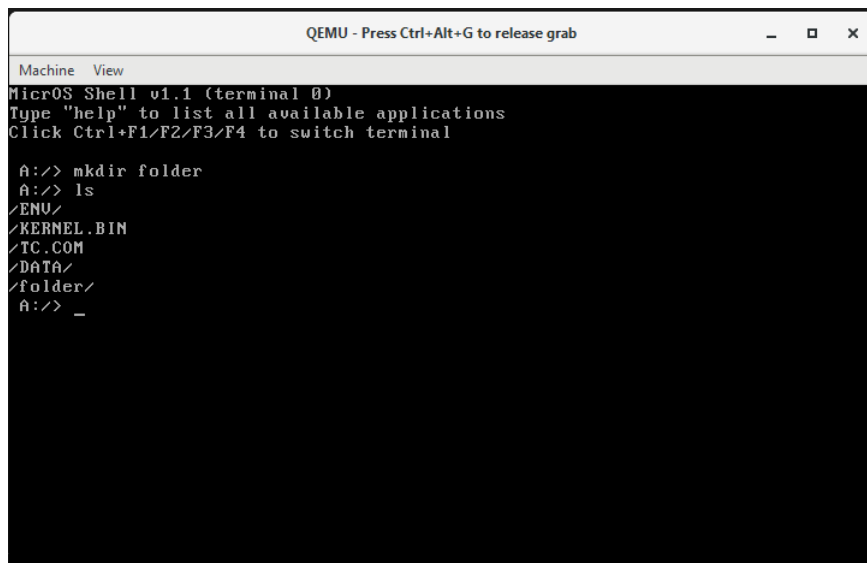
#### 2.1.4. `mkdir`

Program wywołuje się w następujący sposób:

```
mkdir folder
```

Program tworzy nowy folder w aktualnym katalogu, sprawdzając przy tym czy taki katalog już nie istnieje.

Rysunek 2.4 przedstawia przykładowe użycie programu.



```

QEMU - Press Ctrl+Alt+G to release grab
Machine View
MicroOS Shell v1.1 (terminal 0)
Type "help" to list all available applications
Click Ctrl+F1/F2/F3/F4 to switch terminal

A: /> mkdir folder
A: /> ls
/ENU/
/KERNEL.BIN
/TC.COM
/DAŹA/
/folder/
A: /> _

```

Rysunek 2.4: Przykładowe użycie programu `mkdir`



### 2.1.5. reboot

Program wywołuje się w następujący sposób:

```
reboot
```

Program wykonuje ponowne uruchomienie komputera. Aktualnie obsługuje tylko emulatory, a nie fizyczny sprzęt.

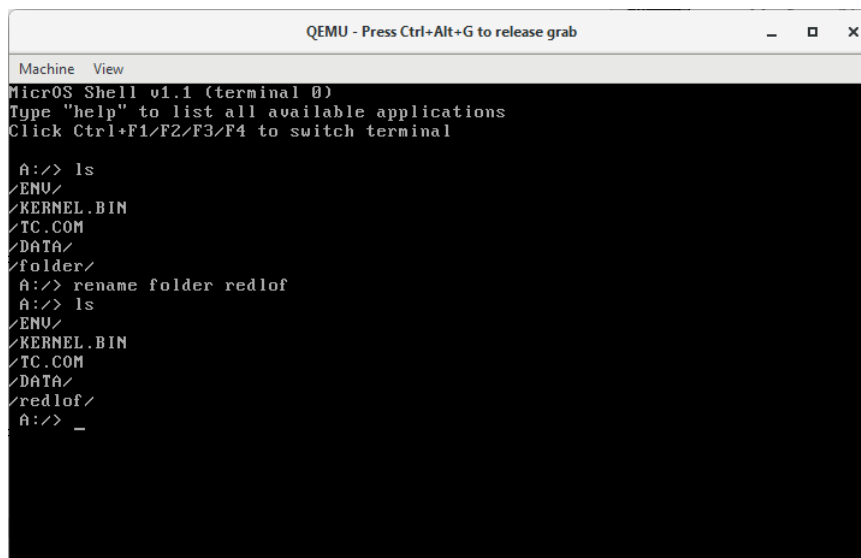
### 2.1.6. rename

Program wywołuje się w następujący sposób:

```
rename stara_nazwa nowa_nazwa
```

Program pozwala zmienić nazwę pliku lub folderu.

Rysunek 2.5 przedstawia przykładowe użycie programu.



Rysunek 2.5: Przykładowe użycie programu rename

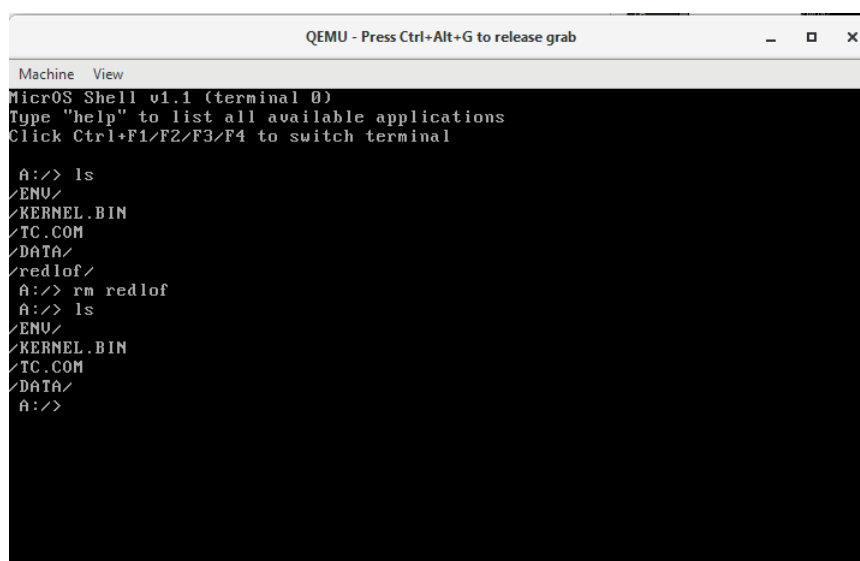
### 2.1.7. rm

Program wywołuje się w następujący sposób:

```
rm nazwa
```

Program pozwala usunąć plik lub folder z systemu.

Rysunek 2.6 przedstawia przykładowe użycie programu.



```

QEMU - Press Ctrl+Alt+G to release grab
Machine View
MicroS Shell v1.1 (terminal 0)
Type "help" to list all available applications
Click Ctrl+F1/F2/F3/F4 to switch terminal

A: /> ls
/ENU/
/KERNEL.BIN
/TC.COM
/DATA/
/redlof/
A: /> rm redlof
A: /> ls
/ENU/
/KERNEL.BIN
/TC.COM
/DATA/
A: />

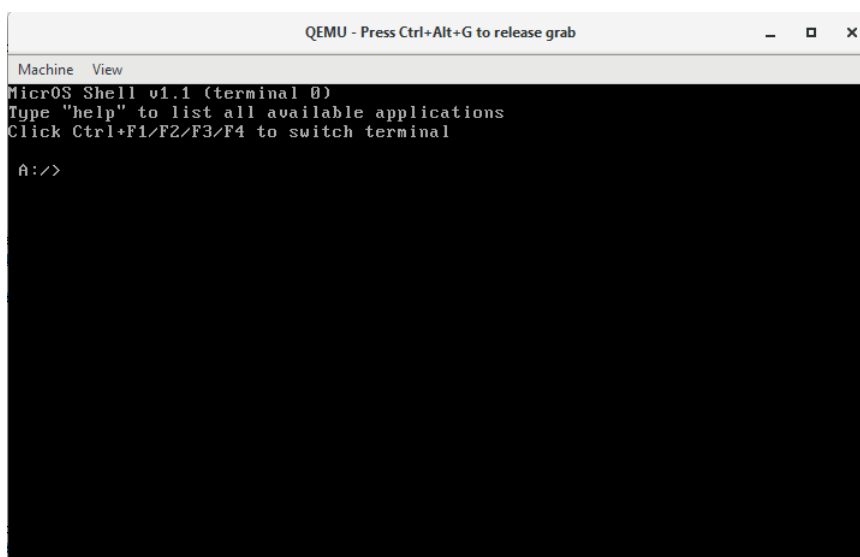
```

Rysunek 2.6: Przykładowe użycie programu `rm`

### 2.1.8. shell

Program uruchamiany jest razem z systemem i stanowi powłokę systemową. Shell, zawiera 4 terminale między, którymi można się przełączać skrótem klawiszowym `Ctrl + F1/F2/F3/F4`.

Rysunek 2.7 przedstawia przykładowe działanie programu.



```

QEMU - Press Ctrl+Alt+G to release grab
Machine View
MicroS Shell v1.1 (terminal 0)
Type "help" to list all available applications
Click Ctrl+F1/F2/F3/F4 to switch terminal

A: />

```

Rysunek 2.7: Przykładowe użycie programu `shell`

### 2.1.9. shutdown

Program wywołuje się w następujący sposób:

shutdown

Program wykonuje wyłącza komputer. Aktualnie obsługuje tylko emulatory, a nie fizyczny sprzęt.

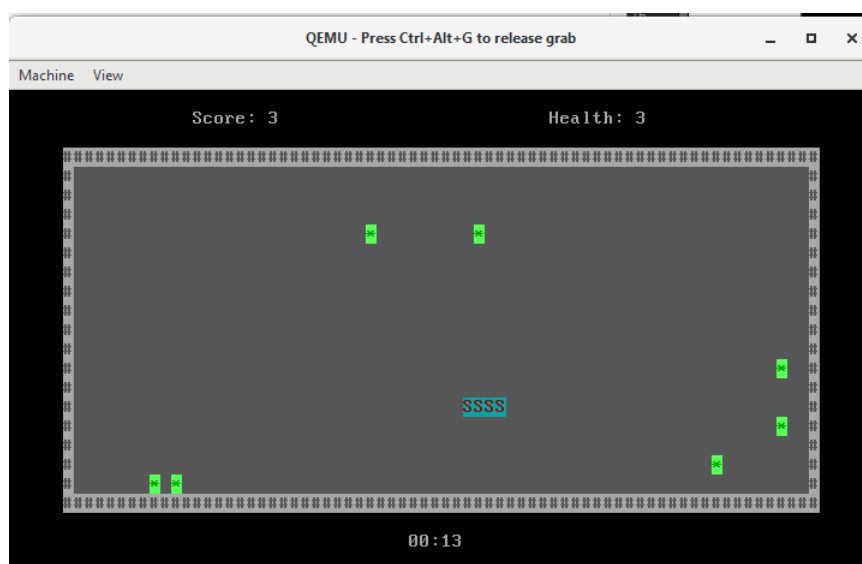
### 2.1.10. snake

Program wywołuje się w następujący sposób:

snake

Gra w węża, polegająca na zbieraniu punktów w postaci zielonych \*. Gracz reprezentowany jest przez ciąg czerwonych S na niebieskim tle. Liczba żyć to 3. Gra potrafi zapisywać najlepsze wyniki, trafiają one do pliku /DATA/SNAKE.SAV/.

Rysunek 2.8 przedstawia przykładowe działanie programu.



Rysunek 2.8: Przykładowe działanie programu snake

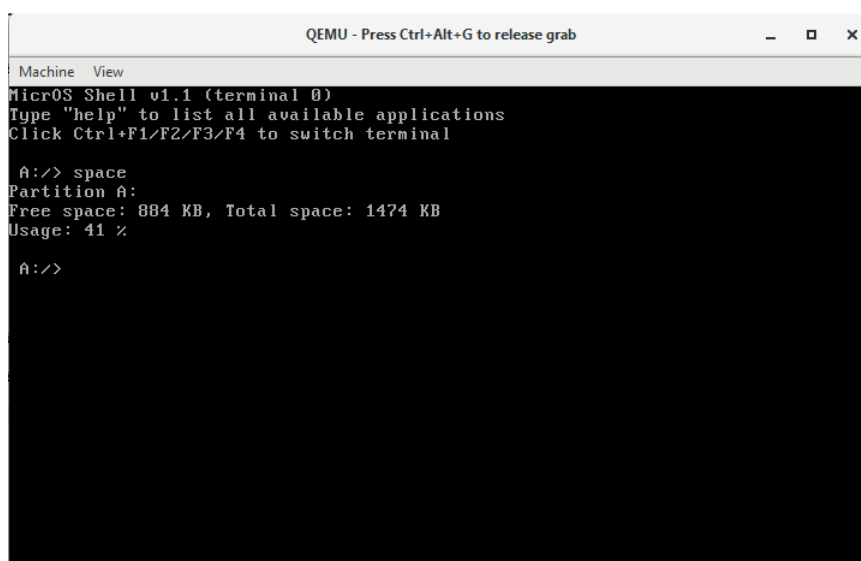
### 2.1.11. space

Program wywołuje się w następujący sposób:

space

Program wyświetla wszystkie podłączone partycje oraz zajęte i całkowite miejsce na nich, a także użyte miejsce w postaci procentowej.

Rysunek 2.9 przedstawia przykładowe działanie programu.



```

QEMU - Press Ctrl+Alt+G to release grab
Machine View
MicrOS Shell v1.1 (terminal 0)
Type "help" to list all available applications
Click Ctrl+F1/F2/F3/F4 to switch terminal

A:/> space
Partition A:
Free space: 884 KB, Total space: 1474 KB
Usage: 41 %

A:/>

```

Rysunek 2.9: Przykładowe działanie programu space

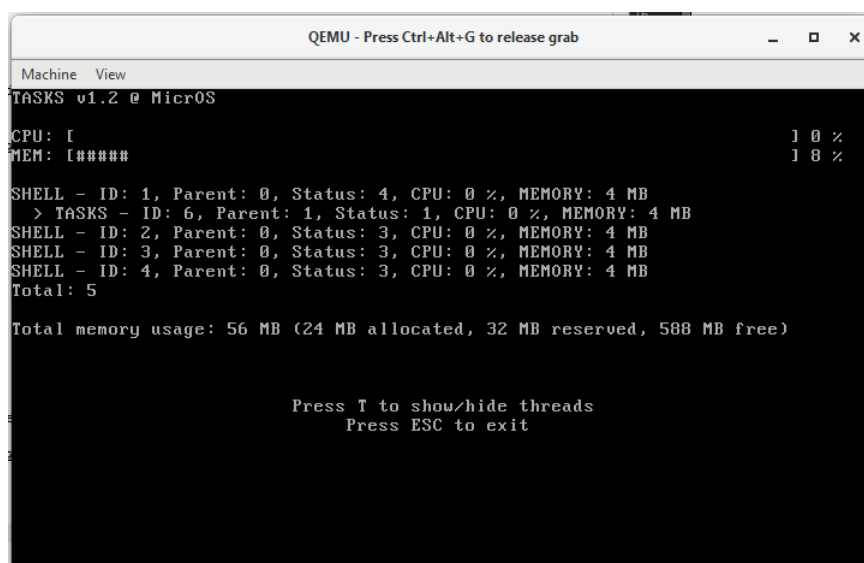
## 2.1.12. tasks

Program wywołuje się w następujący sposób:

```
tasks
```

Program wyświetla wszystkie działające procesy i wątki. Prezentowane jest całkowite zużycie pamięci i procesora, a także per proces. Zachowana jest także struktura rodzic - dziecko.

Rysunek 2.10 przedstawia przykładowe działanie programu.



```

QEMU - Press Ctrl+Alt+G to release grab
Machine View
TASKS v1.2 @ MicrOS

CPU: [                                     ] 0 %
MEM: [#####] 18 %

SHELL - ID: 1, Parent: 0, Status: 4, CPU: 0 %, MEMORY: 4 MB
> TASKS - ID: 6, Parent: 1, Status: 1, CPU: 0 %, MEMORY: 4 MB
SHELL - ID: 2, Parent: 0, Status: 3, CPU: 0 %, MEMORY: 4 MB
SHELL - ID: 3, Parent: 0, Status: 3, CPU: 0 %, MEMORY: 4 MB
SHELL - ID: 4, Parent: 0, Status: 3, CPU: 0 %, MEMORY: 4 MB
Total: 5

Total memory usage: 56 MB (24 MB allocated, 32 MB reserved, 588 MB free)

Press T to show/hide threads
Press ESC to exit

```

Rysunek 2.10: Przykładowe działanie programu tasks

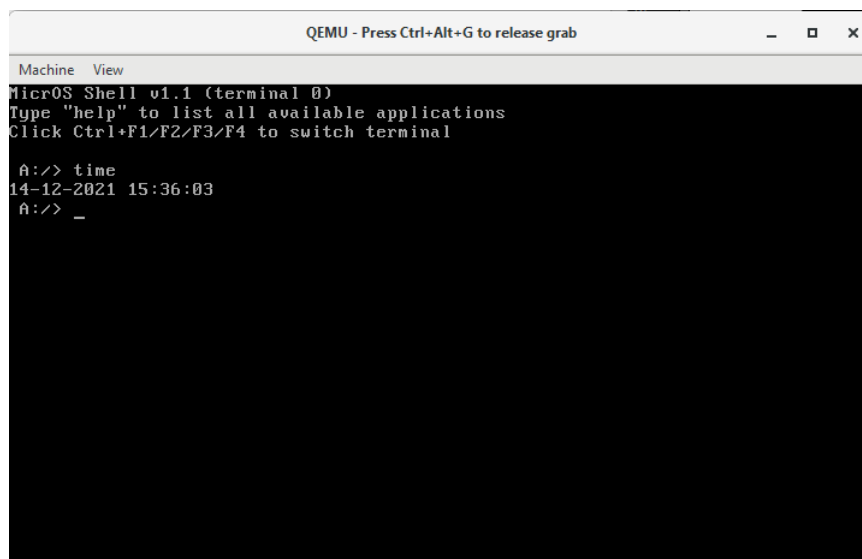
### 2.1.13. time

Program wywołuje się w następujący sposób:

```
time
```

Program wyświetla aktualną datę i czas.

Rysunek 2.11 przedstawia przykładowe działanie programu.



Rysunek 2.11: Przykładowe działanie programu `time`

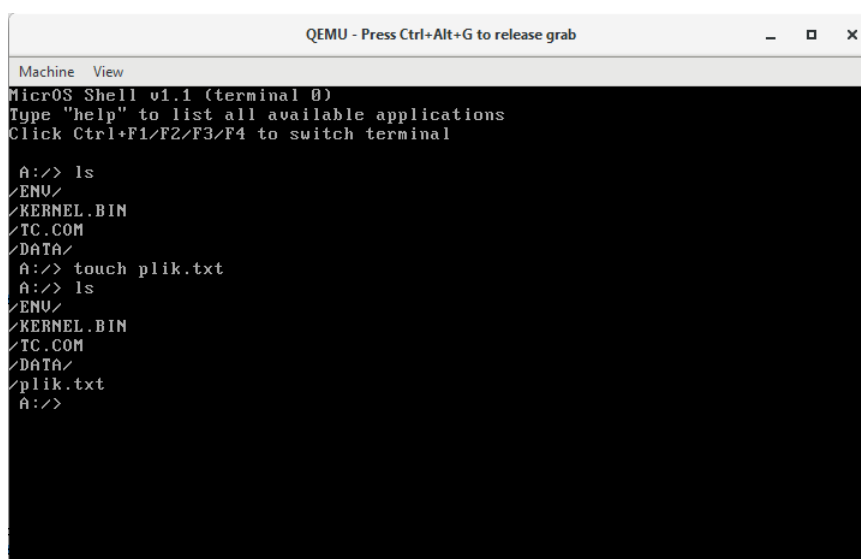
### 2.1.14. touch

Program wywołuje się w następujący sposób:

```
touch nazwa
```

Program tworzy nowy plik o zadanej nazwie. W przypadku niepodania rozszerzenia program sam dodaje kropkę na końcu nazwy pliku.

Rysunek 2.12 przedstawia przykładowe działanie programu.



```

QEMU - Press Ctrl+Alt+G to release grab
Machine View
MicrOS Shell v1.1 (terminal 0)
Type "help" to list all available applications
Click Ctrl+F1/F2/F3/F4 to switch terminal

A:/> ls
/ENU/
/KERNEL.BIN
/TC.COM
/DATA/
A:/> touch plik.txt
A:/> ls
/ENU/
/KERNEL.BIN
/TC.COM
/DATA/
plik.txt
A:/>

```

Rysunek 2.12: Przykładowe działanie programu touch

## 2.2. Sterowniki

### 2.2.1. Prosty sterownik VGA

Prosty sterownik VGA umożliwia obsługę trybu tekstowego, a konkretnie trybów przedstawionych w tabeli 2.1. Pozwala on na obsługę wielu ekranów i przechowywanie na nich danych, jednak nie ma obecnie możliwości przełączenia aktywnego ekranu na inny. Ekrany należy kopiować jego zawartość znajdującymi się w sterowniku funkcjami. Sterownik obsługuje też zmianę pozycji oraz rozmiaru kursora, a także aktywację migania liter. Wbudowane struktury i funkcje ułatwiają zarządzanie danymi na ekranie, między innymi wypisywanie tekstu i kolorowego tekstu, ustawianie i pobieranie znaków z danej pozycji ekranu. Nie zapamiętuje on linii, które znikają z ekranu w wyniku jego przepełnienia.

Tablica 2.1: Obsługiwane przez prosty sterownik tekstowy tryby

Tryb	Kolumny	Wiersze
0h	40	25
1h	40	25
2h	80	25
3h	80	25
7h	80	25

Sterownik obsługuje tylko znak nowej linii. Finalnie obsługa tego typu rzeczy powinna być wy-

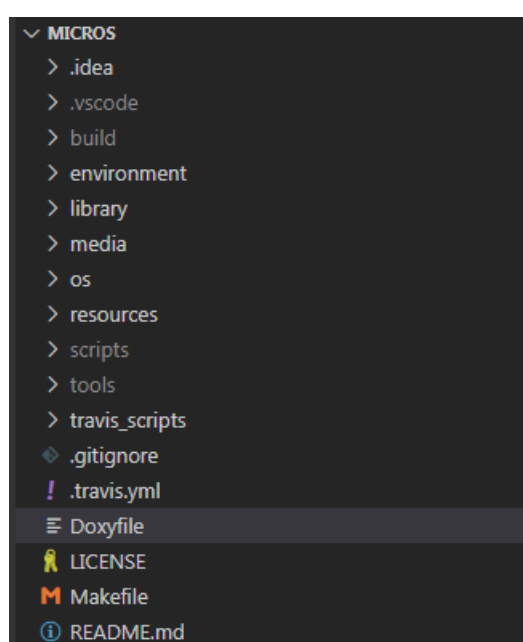
niesiona poza ten sterownik.

### 3. Opis struktury projektu

Rozdział ten traktuje o układzie projektu.

#### 3.1. Podział plików

Pliki w projekcie podzielone są na odpowiednie katalogi, które odpowiadają ich przeznaczeniu. Obecna struktura przedstawiona jest na rysunku 3.13.



Rysunek 3.13: Główny katalog projektu

1. Folder `.idea` zawiera pliki środowiska CLion.
2. Folder `.vscode` zawiera ustawienia wykorzystywane przez Visual Studio Code. Używane są one do budowania oraz debugowania.
3. Folder `build` zawiera skompilowane pliki jądra, bibliotek, programów, obraz dyskiety, obrazy dysków i inne tego typu rzeczy.
4. Folder `environment` zawiera programy MicrOSa.
5. Folder `library` zawiera biblioteki, w tym naszą bibliotekę standardową oraz bibliotekę MicrOSa.



6. Folder `media` zawiera obrazki wykorzystywane na GitHubie.
7. Folder `os` zawiera kod jądra i bootloadera.
8. Folder `resources` zawiera dodatkowe pliki MicroSa wrzucane na dyskietkę podczas kompilacji.
9. Folder `scripts` zawiera skrypty wykorzystywane do budowania oraz debugowania.
10. Folder `tools` zawiera narzędzia używane do budowania i linkowania.
11. Folder `travis_scripts` zawiera skrypty dla Trávisa.
12. Plik `.travis.yml` zawiera ustawienia dla Trávisa.
13. Plik `Doxyfile` zawiera ustawienia dla Doxygena.
14. Plik `LICENSE` zawiera licencję.
15. Plik `Makefile` to główny makefile projektu.

## 3.2. Oprogramowanie

W katalogu `environment` znajduje się oprogramowanie MicroSa. Zastosowanie poszczególnych programów zostało opisane w tabeli 3.2.

Tablica 3.2: Opis programów z folderu `environment`

Program	Opis
<code>cat</code>	Wyświetla zawartość pliku tekstowego.
<code>help</code>	Wyświetla listę komend.
<code>ls</code>	Wyświetla zawartość folderu.
<code>mkdir</code>	Tworzy folder
<code>reboot</code>	Uruchamia ponownie system.
<code>rename</code>	Zmienia nazwę pliku lub folderu.
<code>rm</code>	usuwa plik lub folder.
<code>shell</code>	Program wyświetlający konsolę MicroSa.
<code>shutdown</code>	Wyłącza system.
<code>snake</code>	Gra w węża.
<code>space</code>	Zwraca dostępne partycje i wolne miejsce na nich.
<code>tasks</code>	Wyświetla aktualnie uruchomione procesy.
<code>time</code>	Wyświetla czas systemowy.
<code>touch</code>	Tworzy plik.

### 3.3. Jądro

W katalogu `os/kernel/src` znajduje się oprogramowanie jądra MicroSa. Zastosowanie poszczególnych katalogów zostało opisane w tabeli 3.3.

Tablica 3.3: Opis katalogów i plików z folderu `os/kernel/src`

Program	Opis
<code>assembly</code>	Zawiera kod dotyczący portów pisany w asemblerze.
<code>cpu</code>	Kod dotyczący obsługi procesora i rzeczy z nim związanych.
<code>debug_helpers</code>	Kod wspierający debugowanie jądra.
<code>drivers</code>	Sterowniki.
<code>filesystems</code>	Obsługa systemów plików.
<code>gdb</code>	Komunikacja z GDB.
<code>init</code>	Muzyka startowa systemu.
<code>klibrary</code>	Biblioteka jądra.
<code>logger</code>	Mechanizm logowania informacji.
<code>memory</code>	Kod dotyczący zarządzania pamięcią.
<code>power</code>	Kod dotyczący zarządzania zasilaniem.
<code>process</code>	Kod dotyczący procesów i wątków.
<code>terminal</code>	Kod dotyczący terminali.
<code>v8086</code>	Maszyna wirtualna v8086
<code>kernel.c</code>	Główny plik jądra.

## 4. Implementacja

Rozdział ten będzie zawierał szczegóły implementacji MicroSa.

### 4.1. Oprogramowanie

### 4.2. Sterowniki

#### 4.2.1. Prosty sterownik VGA

Implementacja prostego sterownika VGA opiera się na bezpośrednim operowaniu na pamięci, gdzie znajduje się zmapowana pamięć karty graficznej. Zdefiniowane są definicje dla każdego z 5 trybów tekstowych. Obejmują one: wymiary, adres bazowy, liczbę dostępnych ekranów i kolory.

Zanim rozpoczniemy korzystanie ze sterownika należy wywołać funkcję `vga_init()`, która inicjuje struktury i zmienne w sterowniku VGA. Struktury te wykorzystywane są do zapamiętywania danego trybu graficznego, pozycji kursora na ekranie, aktywnego ekranu i odległości w pamięci między ekranami.

Wypisywanie na ekran odbywa się poprzez wstawianie kolejnych znaków i kolorów do pamięci RAM aktualizując przy tym kursor, który wskazuje aktualną pozycję na ekranie. Sterownik sam obsługuje dodanie nowej linii poprzez przeniesienie wszystkich linii o jedną wyżej na ekranie i wyczyszczenie ostatniej. Obsługa rozmiaru i wyświetlania kursora odbywa się poprzez pisanie na porty procesora.

Ekran zdefiniowany jest jako wskaźnik na tablicę typu `screen_char`. Jest to unia zawierająca liczbę 16-bitową oraz `vga_character`. Obie te wartości reprezentują znak oraz jego kolor. Co staje się jasne gdy zajrzemy do implementacji `vga_character`. Mamy tam kod znaku `ascii` jako zmienną `ascii_code` i kolor jako `vga_color`. Unia `vga_color` zawiera wartość 8-bitową, oraz 2 struktury odpowiadające za kolor, gdy włączony jest tryb mrugania literami lub gdy jest wyłączony.

## 5. Przygotowanie środowiska programistycznego

Rozdział zawiera przydatne informacje o tym jak przygotować i używać środowisko programistyczne MicroSa.

### 5.1. Przygotowanie oprogramowania

Aby programować MicroSa będziesz potrzebował kilku programów.

#### 5.1.1. Edytor

Niezależnie od systemu czy to Windows czy to Linux polecamy zainstalować Visual Studio Code<sup>1</sup>, gdyż pod niego powstają wszystkie skrypty. Jest to edytor kodu, który pozwala uruchamiać kompilację czy przeprowadzić debugowanie kodu.

#### 5.1.2. Emulator procesora

Do tego celu użyjcie QEMU<sup>2</sup>, gdyż wszystkie skrypty są pod niego dostosowane.

#### 5.1.3. Montowanie dyskiety

Na Windowsie do montowania dyskiety, skrypty używają programu Imdisk<sup>3</sup>.

#### 5.1.4. Środowisko kompilacyjne

Na Windowsie do kompilacji używamy MSYSa<sup>4</sup>.

### 5.2. Konfigurowanie środowiska przy użyciu MicroS DevTools na Windows

Kiedy już zainstalujecie sobie całe potrzebne oprogramowanie, należy sklonować repozytorium MicroSa z GitHuba<sup>5</sup> oraz pobrać MicroS DevTools<sup>6</sup>. Okno główne programu znajduje się na rysunku 5.14.

---

<sup>1</sup><https://code.visualstudio.com/>

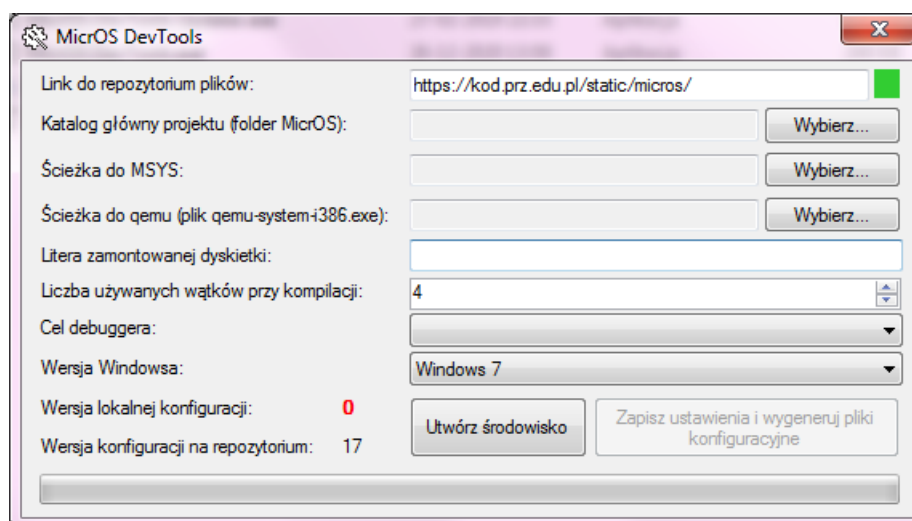
<sup>2</sup><http://www.qemu.org/>

<sup>3</sup><https://sourceforge.net/projects/imdisk-toolkit/>

<sup>4</sup><https://www.msys2.org/>

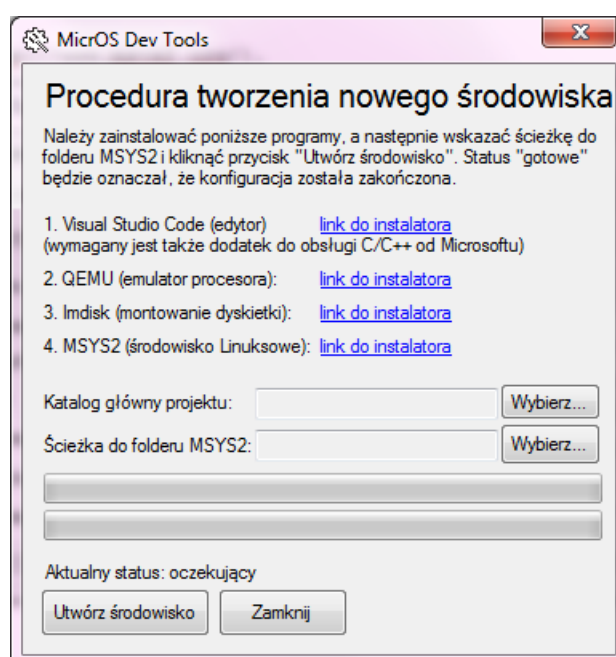
<sup>5</sup><https://github.com/Tearth/MicroS>

<sup>6</sup><https://ldrv.ms/u/s!Av37PqgdOf2M0xS0-VWV1S4Pm2zR?e=BIHNfL>



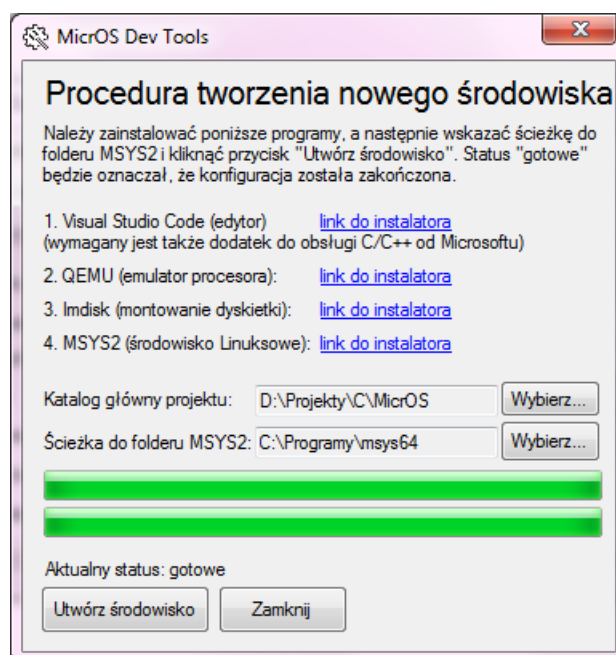
Rysunek 5.14: Główne okno programu MicroS DevTools

Najpierw klikamy przycisk **Utwórz środowisko**. Pojawi się nam okno z rysunku 5.15.



Rysunek 5.15: Okno tworzenia środowiska MicroS DevTools

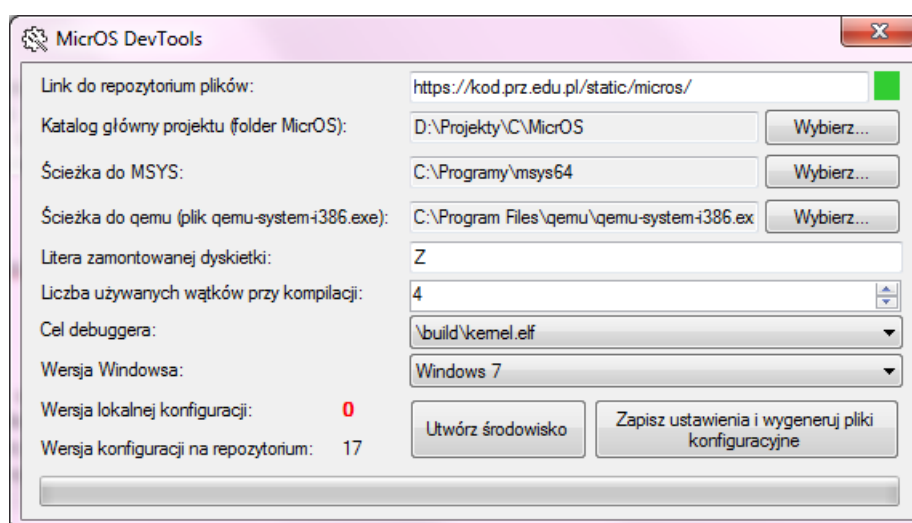
Mamy tutaj linki do pobrania potrzebnych narzędzi oraz dwa pola w których musimy wskazać lokalizację katalogu projektu oraz MSYSa. Po wskazaniu tych opcji klikamy **Utwórz środowisko**. Pobrane zostaną obrazy dyskietki i dysku, kompilator, potrzebne skrypty oraz doinstalowane zostaną dodatkowe komponenty do MSYSa. Kiedy status będzie **Gotowe** zamykamy okno przyciskiem **zamknij**, tak jak na rysunku 5.16.



Rysunek 5.16: Utworzone środowisko

Wróciliśmy do pierwszego okna, gdzie należy wygenerować odpowiednio skrypty używane przez Visual Studio Code. Część pól takie jak katalog główny projektu, jest już wypełniona. Uzupełniamy kolejne pola zgodnie z opisem. Wskazujemy lokalizację QEMU W polu `Litera` zamontowanej dyskiety, podajemy nieprzydzieloną na naszym komputerze literę dysku. Możemy też sprecyzować na ilu wątkach będzie odbywać się kompilacja. Jako cel debuggera wskażcie plik, który chcecie debugować oraz wybierzcie swoją wersję systemu Windows. Przykładowo wypełnione okno jest przedstawione na rysunku 5.17. Gdy wszystko zostało skonfigurowane tak jak chcemy klikamy `Zapisz ustawienia i wygeneruj pliki`. Po tym program zaciągnie z serwera pliki konfiguracyjne i wstawi je do katalogu projektu.

Gdybyś chciał zmienić cel debuggera wystarczy, że wrócisz do tego programu, wybierzesz inny plik i znów wygenerujesz skrypty. Zrób tak samo gdy pojawią się wersje skryptów, aby mieć pewność, że zawsze będziesz pracować na najnowszych.



Rysunek 5.17: Okno przygotowane do wygenerowania skryptów

### 5.2.1. Instalacja starszej wersji GDB

Najnowsza wersja GDB nie działa w naszym przypadku dlatego należy pobrać stąd <sup>7</sup> i zainstalować starszą wersję w MSYS.

Po pobraniu pliku instalacyjnego kolejne komendy znajdujące się na listingu 5.1 wykonujemy w konsoli `mingw64.exe`.

Listing 5.1: Instalacja starszej wersji GDB

---

```
1 pacman -R mingw-w64-x86_64-gdb
2 pacman -U mingw-w64-x86_64-gdb-8.3.1-3-any.pkg.tar.xz
```

---

## 5.3. Konfigurowanie środowiska przy użyciu MicrOS DevTools na Linuxie lub WSL

Jak ktoś woli pracować na Linuxie albo przez WSL to tutaj są narzędzia <sup>8</sup>.

---

<sup>7</sup>[https://kod.prz.edu.pl/MicrOS/mingw-w64-x86\\_64-gdb-8.3.1-3-any.pkg.tar.xz](https://kod.prz.edu.pl/MicrOS/mingw-w64-x86_64-gdb-8.3.1-3-any.pkg.tar.xz)

<sup>8</sup><https://github.com/jaenek/MicrOS-DevTools>

### 5.3.1. Problem z połączeniem GDB pod WSL2

Jeżeli ktoś chce korzystać z WSL2 w celu poprawnego działania debuggera trzeba uruchomić<sup>9</sup>. W innym wypadku nie da się połączyć GDB z QEMU.

## 5.4. Kompilacja cross-kompilatora GCC

*„Building and bootstrapping GCC may take quite a while, so sit back and relax, and enjoy that you are about to use the latest and greatest version of the GNU Compiler Collection.”*

MicrOS wymaga skompilowanego przez nas GCC[2]. MicrOS DevTools ściągają go, jednak jeśli w przyszłości będzie konieczna ponowna kompilacja ten rozdział ją opisuje.

Wszystkie operacje wykonujemy używając konsoli `mingw64.exe`, wersja GCC na potrzeby tej instrukcji to 10.2.0, a binutils to 2.35, oba wypakowane do folderu `$HOME/src`. Ważna uwaga co do wypakowywania tarów: najlepiej zrobić to z poziomu konsoli za pomocą polecenia `tar`, na przykład na listingu 5.2.

### 5.4.1. Pobranie źródeł

Ściągamy najnowszą wersję źródeł binutils ze strony<sup>10</sup> i wypakowujemy do `$HOME/src`. Należy zwrócić uwagę na to by nie powstał nadmiarowy folder nadrzędny. Folder `binutils-2.35` musi bezpośrednio zawierać foldery i pliki binutils.

Ściągamy najnowszą wersję źródeł GCC ze stron<sup>11</sup> i wypakowujemy do `$HOME/src`. Należy zwrócić uwagę na to by nie powstał nadmiarowy folder nadrzędny. Folder `gcc-10.2.0` musi bezpośrednio zawierać foldery i pliki kompilatora.

Listing 5.2: Wypakowywanie plików tar

---

```
1 pacman -S tar
2 tar -xzf binutils-2.35.tar.gz
3 tar -xzf gcc-10.2.0.tar.gz
```

---

---

<sup>9</sup><https://gist.github.com/toryano0820/6ee3bfff2474cdf13e70d972da710996a>

<sup>10</sup><https://ftp.gnu.org/gnu/binutils/>

<sup>11</sup><https://ftp.gnu.org/gnu/gcc/>



### 5.4.2. Pobranie dodatkowych zależności i skompilowanie binutils

Aby kompilować potrzebujemy dodatkowych programów. Instalujemy zależności potrzebne w trakcie kompilacji komendą z listingu 5.3.

Listing 5.3: Instalacja zależności

---

```
1 pacman -S mingw-w64-x86_64-gcc make bison flex gmp-devel mpc-devel mpfr-devel  
   ↪ texinfo diffutils
```

---

Następnie konfigurujemy zmienne środowiskowe zgodnie z listingiem 5.4.

Listing 5.4: Ustawienie zmiennych środowiskowych

---

```
1 export PREFIX="$HOME/opt/cross"  
2 export TARGET="i386-elf"  
3 export PATH="$PREFIX/bin:$PATH"
```

---

Teraz możemy zbudować binutils. Po wykonaniu ostatniego polecenia z listingu 5.5, wszystkie exeki zostaną umieszczone w katalogu \$HOME/opt/cross/bin.

Listing 5.5: Budowanie binutils

---

```
1 cd $HOME/src  
2 mkdir build-binutils  
3 cd build-binutils  
4 ../binutils-2.35/configure --target=$TARGET --prefix="$PREFIX" --with-sysroot  
   ↪ --disable-nls --disable-werror  
5 make -j4  
6 make install
```

---

### 5.4.3. Pobranie dodatkowych zależności i skompilowanie GCC

Ściągamy dodatkowe zależności wymagane przez GCC tak jak na listingu 5.6.

## Listing 5.6: Instalacja zależności

---

```
1 cd $HOME/src/gcc-10.2.0
2 ./contrib/download_prerequisites
```

---

Budujemy GCC. Po wykonaniu ostatniego polecenia z listingu 5.7, wszystkie exeki zostaną umieszczone w katalogu `$HOME/opt/cross/bin`.

## Listing 5.7: Budowanie binutils

---

```
1 cd $HOME/src which -- $TARGET-as || echo $TARGET-as is not in the PATH
2 mkdir build-gcc
3 cd build-gcc
4 ../gcc-10.2.0/configure --target=$TARGET --prefix="$PREFIX" --disable-nls
  ↳ --enable-languages=c,c++ --without-headers
5 make all-gcc -j4
6 make all-target-libgcc -j4
7 make install-gcc
8 make install-target-libgcc
```

---

Folderem wynikowym jest `$HOME/opt/cross`, w którym są wszystkie skompilowane binarki.

#### 5.4.4. Znane problemy

Oto lista znanych problemów wraz z rozwiązaniem:

- a) `download_prerequisites` zawiesza się po pobraniu zipów lub bzip2 invalid argument – należy w takim wypadku przerwać proces i wypakować je ręcznie
- b) `undefined reference to mpfr_fmma` - nie zostały ściągnięte dodatkowe zależności GCC poprzez skrypt `download_prerequisites`
- c) `fatal error: stddef.h: No such file or directory. #include_next<stddef.h>` – przy kompilacji GCC nie został przekazany parametr `--without-headers`, który każe GCC nie używać bibliotek systemowych
- d) `error: '_O_BINARY' undeclared (first use in this function)` – używana jest konsola MSYS2 zamiast `mingw64.exe`

- e) No rule to make target ‘../../gcc/libgcc.mvars’ – GCC jest budowane w folderze ze źródłami zamiast w build-gcc
- f) ln: failed to create symbolic link ‘gmp’: No such file or directory – można zignorować kompilacja powinna się udać

## 6. Ogólne zalecenia dotyczące kodu źródłowego

W przypadku gdy w funkcji używana jest dynamiczna alokacja pamięci, należy zawsze sprawdzić czy nie doszło do wycieku pamięci według przykładu poniżej. Jeżeli na wyjściu obu funkcji będą różne wartości, to oznacza to że doszło do wycieku. Przykład na listingu 6.8.

Listing 6.8: Sprawdzanie dynamicznej alokacji pamięci

---

```
1 heap_dump();
2 strange_function_with_mallocs();
3 heap_dump();
```

---

### 6.1. Pliki

Nazwy folderów w stylu snake\_case, np. kernel, file\_systems, hdd\_wrapper. Nazwy plików małymi literami, podkreślenie jako separator, np. keyboard.c, gdt\_entry.h.

#### 6.1.1. Przedrostki

Stosujemy przedrostki przed nazwami funkcji, zmiennych globalnych, stałych, dyrektyw preprocesora itp. w jądrze, żeby potem łatwiej było nawigować w kodzie. Lepiej gdy funkcja nazywa się floppy\_readdata(), niż readdata(), bo to drugie na pierwszy rzut oka może być wszystkim.

### 6.2. Struktury

Nazwa struktur powinna składać się z małych liter, podkreślenie jako separator, np. idt\_entry, directory\_entry\_time. Każda struktura powinna znajdować się w osobnym pliku nagłówkowym (wyjątek stanowi sytuacja gdy składa się ona z mniejszych podstruktur lub typów wyliczeniowych, wtedy mogą one znajdować się wszystkie w tym samym pliku). Nazwa struktury powinna być identyczna jak nazwa pliku. Struktury które muszą mieć ściśle określone rozmiary pól muszą być pakowane przez dyrektywę \_\_attribute\_\_((packed)) przykład na listingu 6.9.

Listing 6.9: Definicja struktury z użyciem \_\_attribute\_\_((packed))

---

```
1 #ifndef IDT_ENTRY_H
2 #define IDT_ENTRY_H
```

```
3  #pragma pack(1)
4
5  #include <stdint.h>
6
7  /*
8   IDF entry gate types:
9   - task - hardware multitasking things
10  - interrupt - jump to the specified function
11  - trap - similar to interrupt, but without disabling other interrupts
12  */
13  typedef enum idt_entrygatetype {
14      Task_16Bit = 0x5,
15      Interrupt_16Bit = 0x6,
16      Trap_16Bit = 0x7,
17
18      Interrupt_32Bit = 0xE,
19      Trap_32Bit = 0xF
20  } __attribute__((packed)) idt_entrygatetype;
21
22  typedef struct idt_entry {
23      uint16_t offset_0_15;
24      uint16_t selector;
25      uint8_t reserved;
26      idt_entrygatetype type;
27      uint8_t storage_segment : 1;
28      uint8_t privilege_level : 2;
29      uint8_t present : 1;
30      uint16_t offset_16_31;
31  } __attribute__((packed)) idt_entry;
32
33  #endif
```

---

## 6.3. Funkcje

Nazwa funkcji powinna składać się z małych liter, podkreślenie jako separator. Każda nazwa funkcji powinna rozpoczynać się przedrostkiem modułu, do którego należy, np. `floppy_init()`, `fat12_parse_path()`, tak jak na listingu 6.10.

Listing 6.10: Przykład definicji funkcji

---

```
1  #ifndef FAT12_H
2  #define FAT12_H
3
4  #include "directory_entry.h"
5
6  void fat12_init();
7  void fat12_load_fat();
8  void fat12_load_root();
9
10 #endif
```

---

## 6.4. Zmienne

Nazwa zmiennej powinna składać się z małych liter, podkreślenie jako separator. Zmienne nie powinny nazywać się tak samo jak struktury - może powodować to błędy kompilacji. Modyfikator `volatile` powinien być stosowany zawsze, gdy dana zmienna może być zmieniona przez przerwanie. Przykład na listingu 6.11.

Listing 6.11: Przykład definicji zmiennych

---

```
1  floppy_header* floppy_header_data = FLOPPY_HEADER_DATA;
2  uint8_t* dma_buffer = DMA_ADDRESS;
3  uint32_t time_of_last_activity = 0;
4  bool motor_enabled = false;
5
6  volatile bool floppy_interrupt_flag = false;
```

---

## 7. Programowanie

Rozdział ten zawiera przykłady oraz informacje przydatne przy rozwoju MicroSa.

### 7.1. Dostępne standardowe pliki nagłówkowe

Pliki możliwe do zaincludowania dostarczane przez GCC, czyli niewystępujące w naszej bibliotece standardowej:

- `float.h`,
- `iso646.h`,
- `limits.h`,
- `stdalign.h`,
- `stdarg.h`,
- `stdbool.h`,
- `stddef.h`,
- `stdint.h`,
- `stdnoreturn.h`.

### 7.2. Dyrektywy preprocesora podczas pisania plików nagłówkowych biblioteki

Pisząc pliki nagłówkowe bibliotek należy umieszczać ich treść pomiędzy znacznikami z listingu 7.12.

## Listing 7.12: Dyrektywy preprocesora plików nagłówkowych

---

```
1  #ifdef __cplusplus
2  extern "C" {
3  #endif
4
5  //Funkcje, funkcje, funkcje
6
7  #ifdef __cplusplus
8  }
9  #endif
```

---



## Zakończenie

Mam nadzieję, że lektura dokumentu nie nudziła, a także dostarczyła odpowiedzi na niektóre pytania. Jeśli czegoś brakuje dajcie znać, zawsze można coś dopisać.

## Literatura

- [1] <https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/10-useful-tips-to-using-the-floating-point-unit>  
Dostęp 10.03.2021
- [2] <https://gcc.gnu.org/>. Dostęp 27.02.2021

## Spis rysunków

2.1	Przykładowe użycie programu <code>cat</code> . . . . .	6
2.2	Przykładowe użycie programu <code>help</code> . . . . .	7
2.3	Przykładowe użycie programu <code>ls</code> . . . . .	8
2.4	Przykładowe użycie programu <code>mkdir</code> . . . . .	8
2.5	Przykładowe użycie programu <code>rename</code> . . . . .	9
2.6	Przykładowe użycie programu <code>rm</code> . . . . .	10
2.7	Przykładowe użycie programu <code>shell</code> . . . . .	10
2.8	Przykładowe działanie programu <code>snake</code> . . . . .	11
2.9	Przykładowe działanie programu <code>space</code> . . . . .	12
2.10	Przykładowe działanie programu <code>tasks</code> . . . . .	12
2.11	Przykładowe działanie programu <code>time</code> . . . . .	13
2.12	Przykładowe działanie programu <code>touch</code> . . . . .	14
3.13	Główny katalog projektu . . . . .	16
5.14	Główne okno programu MicroOS DevTools . . . . .	21
5.15	Okno tworzenia środowiska MicroOS DevTools . . . . .	21
5.16	Utworzone środowisko . . . . .	22
5.17	Okno przygotowane do wygenerowania skryptów . . . . .	23

## Spis tablic

2.1	Obsługiwane przez prosty sterownik tekstowy tryby . . . . .	14
3.2	Opis programów z folderu <code>environment</code> . . . . .	17
3.3	Opis katalogów i plików z folderu <code>os/kernel/src</code> . . . . .	18

## Spis listingów

5.1	Instalacja starszej wersji GDB . . . . .	23
5.2	Wypakowywanie plików tar . . . . .	24
5.3	Instalacja zależności . . . . .	25
5.4	Ustawienie zmiennych środowiskowych . . . . .	25
5.5	Budowanie binutils . . . . .	25
5.6	Instalacja zależności . . . . .	26
5.7	Budowanie binutils . . . . .	26
6.8	Sprawdzanie dynamicznej alokacji pamięci . . . . .	28
6.9	Definicja struktury z użyciem <code>__attribute__((packed))</code> . . . . .	28
6.10	Przykład definicji funkcji . . . . .	30
6.11	Przykład definicji zmiennych . . . . .	30
7.12	Dyrektywy preprocesora plików nagłówkowych . . . . .	32