



# Milestone 1

Ashman Mehra

Joel Tony

Arnav Gupta

Kumarakrishna Valeti



## Project Description

We plan to port microps (a small C based TCP/IP protocol stack for learning) incrementally and study the challenges and performance impact of integrating Rust into existing C code. We also plan to study the possible memory safety guarantees on integration.

## Microps Demonstration - ICMP Response

```
Zellij (effulgent-cactus) - jaytau@BRAVO:~/Programming/microps

Zellij (effulgent-cactus) > Tab #1
jaytau@BRAVO:~/Programming/microps
off: 0x50 (20)
flg: 0x14 (---A-R--)
wnd: 0x213
sum: 0xc416
up: 0
22:53:47.355 [D] tcp_receive: connection closing (tcp.c:1336)
22:53:47.355 [D] main: connection closed (app/tcp.c:155)
22:53:47.355 [D] tcp_output_segment: 192.0.2.2:7 => 192.0.2.1:41998, len=20 (payload=0) (tcp.c:360)
src: 7
dst: 41998
seq: 1804289411
ack: 3940560091
off: 0x50 (20)
flg: 0x11 (---A---F)
wnd: 0x535
sum: 0xbef
up: 0
22:53:47.355 [D] ip_output_core: dev=net1, iface=192.0.2.2, protocol=TCP(0x06), len=40 (ip.c:393)
vhl: 0x45 [v: 4, hl: 5 (20)]
tos: 0x00
total: 40 (payload: 20)
id: 136
offset: 0x0000 [flags=0, offset=0]
ttl: 255
protocol: 6 (TCP)
sum: 0x3744 (0x3744)
src: 192.0.2.2
dst: 192.0.2.1
22:53:47.355 [D] arp_resolve: resolved, pa=192.0.2.1, ha=1a:2c:d5:9e:95:0d (arp.c:302)
22:53:47.355 [D] net_device_output: dev=net1, type=IP(0x0800), len=40 (net.c:156)
22:53:47.355 [D] ether_transmit_helper: dev=net1, type=IP(0x0800), len=60 (ether.c:181)
src: 00:00:5e:00:53:01
dst: 1a:2c:d5:9e:95:0d
type: 0x0800 (IP)
22:53:47.355 [D] tcp_pcb_release: released, local=0.0.0.0:7, foreign=0.0.0.0:0 (tcp.c:199)
22:53:47.355 [D] net_shutdown: close all devices... (net.c:354)
22:53:47.355 [D] tcp_pcb_release: released, local=192.0.2.2:7, foreign=192.0.2.1:41998 (tcp.c:199)
22:53:47.355 [D] intr_thread: irq=36, name=net1 (platform/linux/intr.c:101)
22:53:47.355 [E] ether_tap_read: read: Bad file descriptor, dev=net1 (platform/linux/driver/ether_tap.c:132)
22:53:47.355 [E] ether_poll_helper: input data is too short (ether.c:116)
22:53:47.355 [E] ether_tap_read: read: Bad file descriptor, dev=net1 (platform/linux/driver/ether_tap.c:132)
22:53:47.355 [E] ether_poll_helper: input data is too short (ether.c:116)
22:53:47.355 [E] ether_tap_read: read: Bad file descriptor, dev=net1 (platform/linux/driver/ether_tap.c:132)
22:53:47.355 [E] ether_poll_helper: input data is too short (ether.c:116)
22:53:47.355 [I] net_device_close: dev=net1, state=down (net.c:110)
22:53:47.355 [I] net_device_close: dev=net0, state=down (net.c:110)
22:53:47.355 [D] net_shutdown: shutdown (net.c:358)
(base) [jaytau@BRAVO microps]$

(base) [jaytau@BRAVO microps]
(base) [jaytau@BRAVO microps]$ nc 192.0.2.2 7
hello
hello
sending data
sending data
exiting
exiting
^C(base) [jaytau@BRAVO microps]$

jaytau@BRAVO:~/Programming/microps
22:53:50 BATV 39% 01:13 2000ms 1.4 GHz
Ryzen 7 5800HS
CPU 2% C0 2% C3 1% C7 6% C9 0% C12 1%
C1 1% C4 1% C7 1% C10 1% C13 1%
C2 1% C5 1% C8 2% C11 9% L 2 1 0 2%
up 00:04:48

mem disks io
Total: 15.0 GiB root 15.0 GiB 256 GiB
Used: 5.54 GiB 10%
Avail: 9.46 GiB Used: 60% 155 GiB
Cache: 4.77 GiB swap 15.5 GiB
Free: 4.79 GiB Used: 6% 0 Byte
home 15.0 GiB 256 GiB
IO% Used: 60% 155 GiB

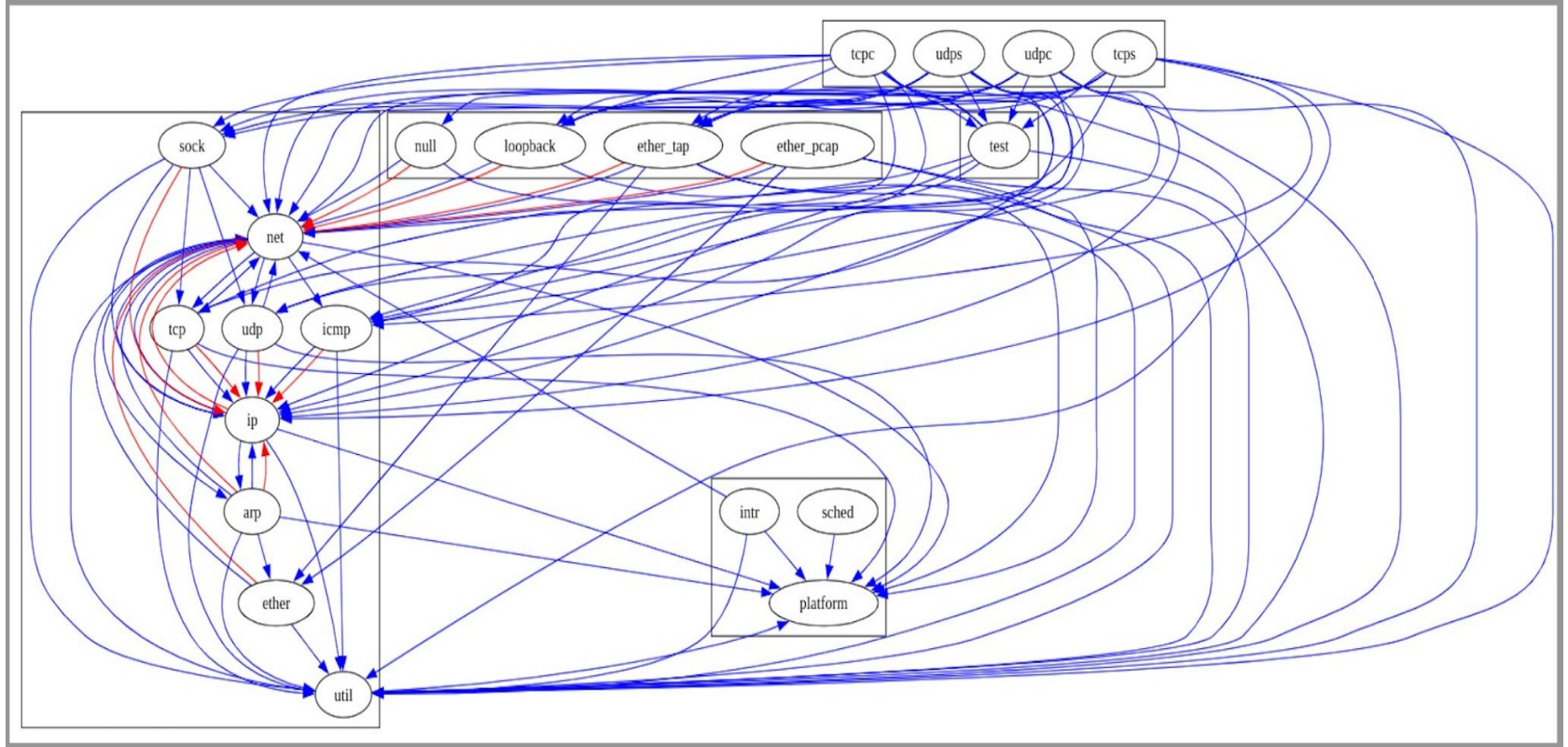
net sync auto zero wlan0 n
download 303 Byte/s (2.36 Kibps)
upload 313 Byte/s (2.44 Kibps)

proc filter per-core reverse tree cpu lazy
Pid: Program: Command: User: MemB Cpu%
9572 Isolated /home/jaytau/.local jayt 549M 0.0
9404 firefox /home/jaytau/.local jayt 393M 0.0
8156 firefox /usr/lib/firefox/fi jayt 510M 0.1
5212 kxg kxg jayt 284M 0.5
8410 WebExten /usr/lib/firefox/fi jayt 312M 0.0
2392 gnome-sh /usr/bin/gnome-shel jayt 343M 0.0
4451 code /opt/visual-studio- jayt 288M 0.0
8089 Web Cont /usr/lib/firefox/fi jayt 264M 0.0
4529 code /opt/visual-studio- jayt 337M 0.0
958 netdata /opt/netdata/bin/sn netd 111M 0.0
1329 apps.plu /opt/netdata/usr/li netd 5.6M 0.1
4576 code /opt/visual-studio- jayt 110M 0.0
4575 code /opt/visual-studio- jayt 106M 0.0
select info terminate kill signals 0/465

Ctrl + <Q> LOCK <P> PANE <T> TAB <R> RESIZE <M> MOVE <S> SEARCH <O> SESSION <Q> QUIT
Tip: Alt + <N> => open new pane. Alt + <C+> or Alt + <Hjkl> => navigate between panes. Alt + <+> => increase/decrease pane size. Alt + <J> VERTICAL
```

# Microps Demonstration - TCP/IP Connection

# Dependency Graph





## First Port - Utils.c

Utils.c (approx 200 lines) is a collection of utility functions for network-related operations and data handling.

lprintf: This function is for logging and printing messages with a timestamp, log level, and source file information.

hexdump: This function is for printing hexadecimal and ASCII representations of binary data in a human-readable format

queue\_init, queue\_push, queue\_pop, queue\_peek, and queue\_foreach: These functions implement a simple queue data structure for managing a list of elements.

endian, byteorder, byteswap16, byteswap32, hton16, ntoh16, hton32, and ntoh32: These functions handle byte order conversion for big-endian and little-endian systems

cksum16: This function calculates a 16-bit checksum for a given data block. It's commonly used in networking to verify data integrity.



## lib.rs

We set up the Cargo library by creating a lib.rs file as the entry point for the Rust code.

One of the notable changes is the implementation of the queue data structure using a struct in rust with associated functions being implemented using 'impl'.

We encountered challenges in finding Rust equivalents for the C libraries and functions used in the original code.

We also faced several errors with type compatibility.



# Foreign Function Interface

**FFI (Foreign Function Interface)** is a way for functions written in one language to be called from another language. It allows Rust programs to interact with other programs and libraries, such as those written in C or C++.

```
Next chapter gle]
pub extern "C" fn hello_from_rust() {
    println!("Hello from Rust!");
}
```

```
[lib]
crate-type = ["cdylib"]
```

```
extern void hello_from_rust();

int main(void) {
    hello_from_rust();
    return 0;
}
```

```
gcc call_rust.c -o call_rust -lrust_from_c -L./target/debug
```





# Integration - Procedure

We have divided the task into 2 parts:

1. Creating a shared library using FFI standards
  - a. Creating a library module in rust
  - b. Declaring the shared functions using `pub extern "C"` and exporting them as modules
  - c. Building a `cargo.toml` file for dependency resolution
2. Importing the rust library in microps using header files
  - a. Creating a build file for microps and including rust shared library in compilation
  - b. Generating a header file to collect method definitions from rust
  - c. Utilizing the rust functions in C for testing

# Integration - Live code demonstration!

We will now demonstrate a working code for 2 functions from the util.c file:

1. dumphex - Memory Intensive
2. Std\_err - I/O Intensive

```
~/Desktop/College/Projects/JAAK Project $ cd rustcode
Desktop/College/Projects/JAAK Project/rustcode $ cargo build --release
    Finished release [optimized] target(s) in 0.09s
Desktop/College/Projects/JAAK Project/rustcode $ cd ../microps
Desktop/College/Projects/JAAK Project/microps $ ./build.sh
Desktop/College/Projects/JAAK Project/microps $ ./bin/myapp
This is a message from C written by rust.
Write to stderr successful.
hexdump completed successfully.
```

- ✓ microps
  - ✓ bin
    - myapp
    - output.txt
  - ✓ include
    - h mylibrary.h
  - ✓ src
    - C main.c
    - build.sh
- ✓ rustcode
  - ✓ src
    - utllib
    - lib.rs
  - ✓ target
    - debug
    - release
      - .rustc\_info.json
  - CACHEDIR.TAG
  - .gitignore
  - Cargo.lock
  - Cargo.toml



# Integration

Integrating with a C/C++ codebase has provided the following challenges. These concepts do not inherently exist in C/C++ which demands extra caution and vigilance

- Ownership model of Rust to leverage the borrow-checker to the full extent. Changing the structure of the code was often necessary for us to keep the ownership rules intact
- Lifetimes. Adding lifetimes required us going through the codebase thoroughly to make sure we didn't end the scope of a variable before it was supposed to
- Traits. Using traits provided a lot of freedom in terms of implementation of data types, which is significantly more robust and type-safe than in C/C++.



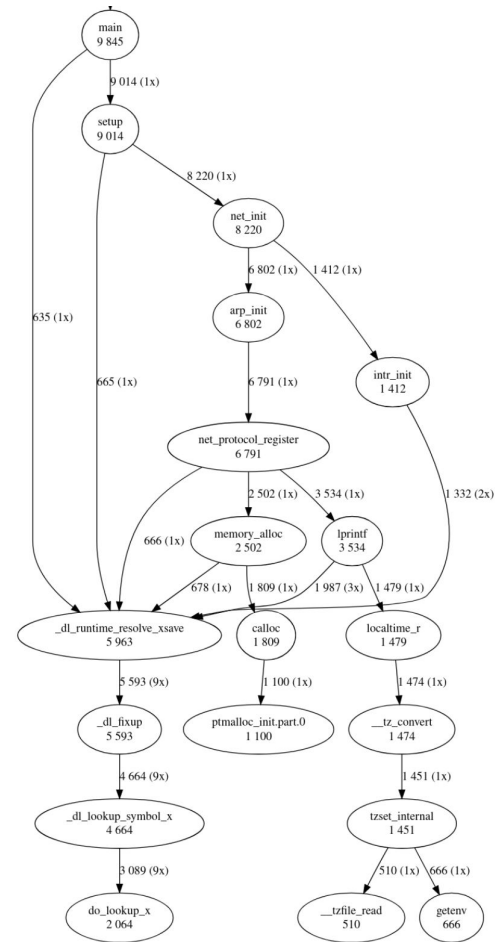
## Preliminary Benchmarking Tools

Callgrind - Captures function call hierarchy, cache misses, and CPU cycles to profile program execution.

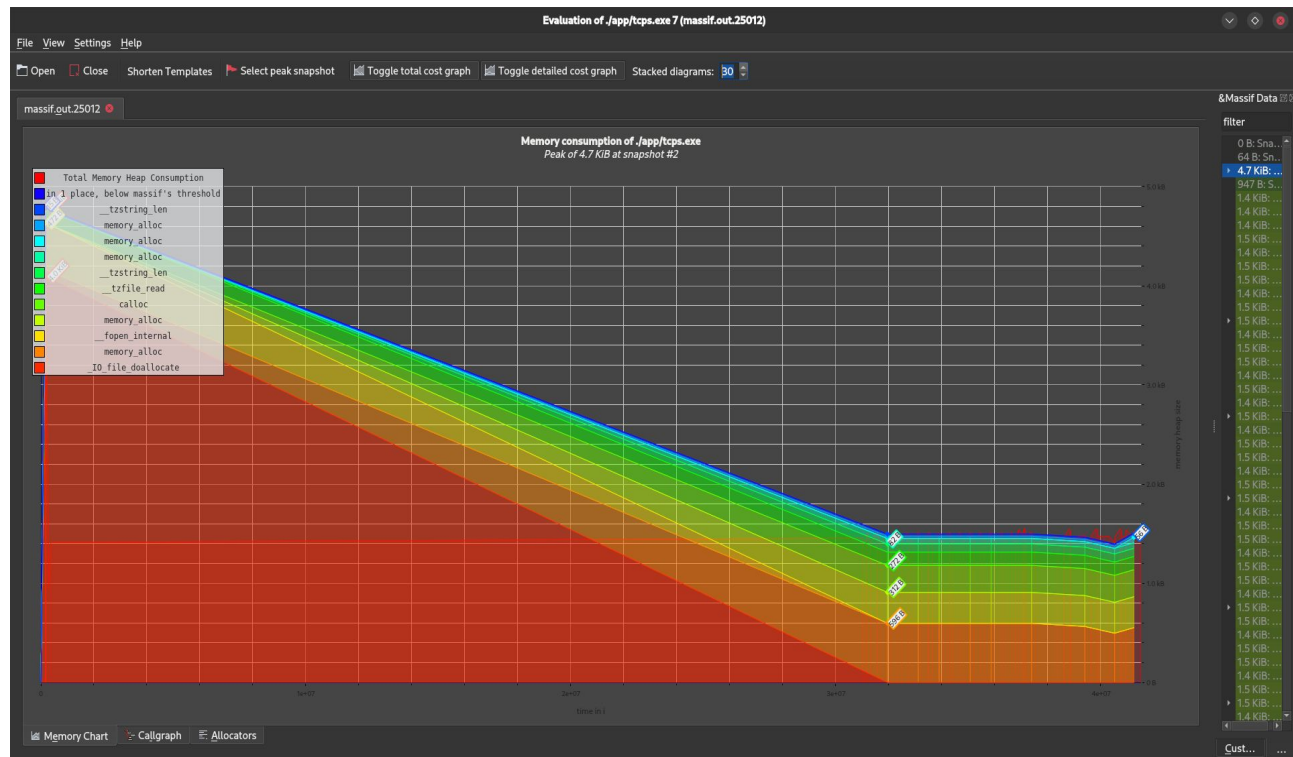
Massif is a heap profiler. It performs detailed heap profiling by taking regular snapshots of a program's heap. It produces a graph showing heap usage over time, including information about which parts of the program are responsible for the most memory allocations.

DHAT is a tool for examining how programs use their heap allocations. It tracks the allocated blocks, and inspects every memory access to find which block, if any, it is to.

# Callgrind



# Massif



# DHAT

```
Invocation {  
  Mode: heap  
  Command: ./app/tcps.exe 7  
  PID: 22341  
}
```

```
Times {  
  t-gmax: 1,98,530 instrs (1.37% of program duration)  
  t-end: 1,45,16,490 instrs  
}
```

```
▼ PP 1/1 (9 children) {  
  Total: 9,063 bytes (100%, 624.32/Minstr) in 93 blocks (100%, 6.41/Minstr), avg size 97.45 bytes, avg lifetime 40,59,478.35 instrs (27.96% of program duration)  
  At t-gmax: 4,781 bytes (100%) in 5 blocks (100%), avg size 956.2 bytes  
  At t-end: 1,160 bytes (100%) in 19 blocks (100%), avg size 61.05 bytes  
  Reads: 30,23,958 bytes (100%, 2,08,311.93/Minstr), 333.66/byte  
  Writes: 13,057 bytes (100%, 899.46/Minstr), 1.44/byte  
  Allocated at {  
    #0: [root]  
  }  
}  
PP 1.1/9 {  
  Total: 4,096 bytes (45.19%, 282.16/Minstr) in 1 blocks (1.08%, 0.07/Minstr), avg size 4,096 bytes, avg lifetime 3,414 instrs (0.02% of program duration)  
  Max: 4,096 bytes in 1 blocks, avg size 4,096 bytes  
  At t-gmax: 4,096 bytes (85.67%) in 1 blocks (20%), avg size 4,096 bytes  
  At t-end: 0 bytes (0%) in 0 blocks (0%), avg size 0 bytes  
  Reads: 214 bytes (0.01%, 14.74/Minstr), 0.05/byte  
  Writes: 454 bytes (3.48%, 31.27/Minstr), 0.11/byte  
  Allocated at {  
    #1: 0x48FE499: _IO_file_doallocate (filedoalloc.c:101)  
    #2: 0x490D278: _IO_doallocbuf (genops.c:347)  
    #3: 0x490D278: _IO_doallocbuf (genops.c:342)  
    #4: 0x490C18C: _IO_file_xsgetn (fileops.c:1288)  
    #5: 0x4909DB3: fread_unlocked (iofread_u.c:40)  
    #6: 0x4952C53: __tzfile_read (tzfile.c:187)  
    #7: 0x49526B4: tzset_internal (tzset.c:405)  
    #8: 0x49528B6: __tz_convert (tzset.c:577)  
    #9: 0x10AE35: lprintf (util.c:27)  
    #10: 0x10BFCD: net_protocol_register (net.c:218)  
    #11: 0x10DF90: arp_init (arp.c:331)  
    #12: 0x10C613: net_init (net.c:374)  
  }  
}
```



# Questions?