GEISA IPC Timings

Norm McEntire
ncmcentire@ucsd.edu

License: CC BY-SA

- NOTE: CC BY-SA is to documentation what GNU GPL is to software

- CC: Creative Commons

- BY: Give credit to the original author

- SA: ShareAlike: When you remix, transform, or build upon material, redistribute your contributions under the same license

About These Slides

- These slides are from an upcoming **UCSD Extended Studies** course taught by Norman (Norm) McEntire on **Embedded Linux IPC**
  - IPC = Interprocess Communication

- Other UCSD Extended Studies Courses Taught by Norman McEntire
  - https://extendedstudies.ucsd.edu/courses/embedded-linux-ece-40105
  - https://extendedstudies.ucsd.edu/courses/embedded-real-time-operating-system-(rtos)-ece-40290

Contents

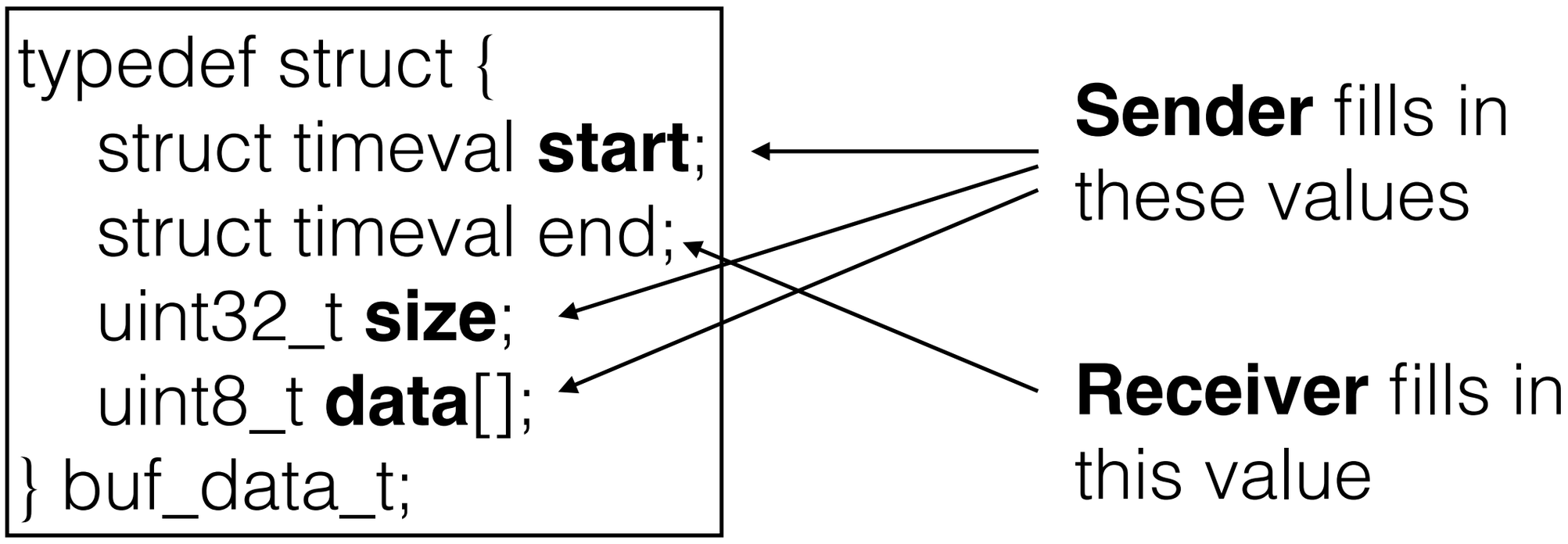
- Introduction
- Memory Copy (baseline)
- POSIX Shared Memory Copy (two processes)
- UDP Memory Copy (two processes)
- TCP Memory Copy (two processes)
- Dbus Memory Copy (two processes)
- ZeroMQ Memory Copy (two processes)
- MQTT Memory Copy (two processes)

Introduction

Introduction

- The following is a study of doing timings of how long it takes a process (e.g. a publisher) to send a block of data to another process (e.g. a subscriber)
- All code written in Embedded Linux C and run on a Raspberry Pi
    - All code shown at end of the slides
- Command-line parameter to set the number of bytes transferred
    - --size NUMBER
    - Example: ./memcpy -- size $((1024*1024))
- To establish a **baseline**, we being with a process that copies a buffer from a source to a designation in the same process
    - memcpy()

Buffer Data Type



Example: gettimeofday(&start, NULL);

Reference Hardware
x86_64/64GB RAM

```
$ arch
x86_64
```

```
$ uname -a
Linux system76-pc 6.8.0-76060800daily20240311-generic #202403110203~1715181801~22.04~aba43ee~dev-Ubunt
SMP PREEMPT_DY x86_64 x86_64 x86_64 GNU/Linux
```

```
$ cat /etc/os-release
cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.4 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="24.04.4 LTS (Jammy Jellyfish)"
. . .
UBUNTU_CODENAME=jammy
```

```
$ free
               total        used        free      shared  buff/cache   available
Mem:        65034456    11057712     1350816      158300    52625928    53093052
Swap:             0           0           0
```

Reference Hardware
RPi Model 4B/2GB RAM

```
$ arch
aarch64
```

```
$ uname -a
Linux raspberrypi 6.1.0-rpi6-rpi-v8 #1 SMP PREEMPT Debian 1:6.1.58-1+rpt2 (2023-10-27) aarch64
GNU/Linux
```

```
$ cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 12 (bookworm)"
NAME="Debian GNU/Linux"
VERSION_ID="12"
VERSION="12 (bookworm)"
VERSION_CODENAME=bookworm
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```

```
$ free
               total        used        free      shared  buff/cache   available
Mem:         1892440      367672      213940       13656     1393924     1524768
Swap:         102396        2304      100092
```

Reference Hardware
RPi Zero/512MB RAM

```
$ arch
armv6l
```

```
$ uname -a
Linux raspberrypi 6.6.51+rpt-rpi-v6 #1 Raspbian 1:6.6.51-1+rpt3 (2024-10-08) armv6l GNU/Linux
```

```
$ cat /etc/os-release
cat /etc/os-release
PRETTY_NAME="Raspbian GNU/Linux 12 (bookworm)"
NAME="Raspbian GNU/Linux"
VERSION_ID="12"
VERSION="12 (bookworm)"
VERSION_CODENAME=bookworm
ID=raspbian
ID_LIKE=debian
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
```

```
$ free
               total        used        free      shared  buff/cache   available
Mem:          438164      115192      246520        1116      126144      322972
Swap:         524284           0      524284
```

High-Level Summary (Measured on RPi Zero)

- **Name => Low MB/s => High MB/s**

- memcpy => 99 MB/s => 120 MB/s
- shmemcpy => 97 MBs => 111 MB/s
- tcpmemcpy => 56 MB/s => 62 MB/s

- udpmemcpy => 21 MB/s => 25 MB/s
- zmqmemcpy => 14 MB/s => 25 MB/s
- dbusmemcpy => 7 MB/s => 8 MB/s

High-Level Summary (Measured on RPi Model 4B)

- **Name => Low MB/s => High MB/s**

- memcpy => 192 MB/s => 290 MB/s
- tcpmemcpy => 141 MB/s => 233 MB/s
- shmem => 137 MB/s => 161 MB/s

- udpmemcpy => 57 MB/s => 73 MB/s
- zmqmemcpy => 40 MB/s => 80 MB/s
- dbusmemcpy => 18 MB/s => 51 MB/s

High-Level Summary (Measured on x86_64)

- **Name => Low MB/s => High MB/s**

- memcpy => 2.4 GB/s => 2.5 GB/s
- tcpmemcpy => 141 MB/s => 233 MB/s
- shmem => 1.6 GB/s => 1.9 GB/s

- udpmemcpy => 350 MB/s => 372 MB/s
- zmqmemcpy => 40 MB/s => 80 MB/s
- dbusmemcpy => 18 MB/s => 51 MB/s

Memory Copy

Demo 1: memcpy Overview

- This base line demo records the fastest possible way to copy a source buffer to a destination buffer
- All other demos will take longer to copy data than this one

Demo 1: memcpy
Key Design Features

- memcpy()

- No additional libraries required (built into libc)

Demo 1 - memcpy.c

```
$ gcc -Wall memcpy.c -o memcpy

$ file memcpy
memcpy: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV),
dynamically linked, interpreter /lib/ld-linux-aarch64.so.1,
BuildID[sha1]=b39c9c15878fdcdfa29f66ca10bf35bed9a2a363,
for GNU/Linux 3.7.0, not stripped

$ ldd memcpy
   linux-vdso.so.1 (0x0000007fae943000)
   libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000007fae730000)
   /lib/ld-linux-aarch64.so.1 (0x0000007fae906000)
```

Demo 1 - memcpy.c Running on RPi Zero

```
$ ./memcpy --size $((1024*1024))
Start Time: 1747746828.917924 seconds
End Time:   1747746828.928413 seconds
Elapsed Time: 0.010489 seconds
Transferred:  1048576 bytes
Throughput:   99969110.50 bytes/second
              99.97 MB/second

$ ./memcpy --size $((1024*1024))
Start Time: 1747746830.746359 seconds
End Time:   1747746830.756665 seconds
Elapsed Time: 0.010306 seconds
Transferred:  1048576 bytes
Throughput:   101744226.66 bytes/second
              101.74 MB/second

$ ./memcpy --size $((1024*1024))
Start Time: 1747746834.735617 seconds
End Time:   1747746834.744288 seconds
Elapsed Time: 0.008671 seconds
Transferred:  1048576 bytes
Throughput:   120929073.92 bytes/second
              120.93 MB/second
```

100 MB/s

Demo 1 - memcpy.c Running on RPi Model 4B

```
./memcpy --size $((1024*1024))
Start Time: 1747751963.924807 seconds
End Time:   1747751963.929527 seconds
Elapsed Time: 0.004720 seconds
Transferred:  1048576 bytes
Throughput:   222155932.20 bytes/second
              222.16 MB/second

$ ./memcpy --size $((1024*1024))
Start Time: 1747751965.402242 seconds
End Time:   1747751965.407261 seconds
Elapsed Time: 0.005019 seconds
Transferred:  1048576 bytes
Throughput:   208921299.06 bytes/second
              208.92 MB/second

$ ./memcpy --size $((1024*1024))
Start Time: 1747751966.843833 seconds
End Time:   1747751966.848634 seconds
Elapsed Time: 0.004801 seconds
Transferred:  1048576 bytes
Throughput:   218407831.70 bytes/second
              218.41 MB/second
```

200 MB/s

Demo 1 - memcpy.c Running on x86_64

```
./memcpy --size $((1024*1024))
Start Time: 1747751670.896839 seconds
End Time:   1747751670.897243 seconds
Elapsed Time: 0.000404 seconds
Transferred:  1048576 bytes
Throughput:   2595485148.51 bytes/second
              2595.49 MB/second


$ ./memcpy --size $((1024*1024))
Start Time: 1747751684.525879 seconds
End Time:   1747751684.526291 seconds
Elapsed Time: 0.000412 seconds
Transferred:  1048576 bytes
Throughput:   2545087378.64 bytes/second
              2545.09 MB/second

$ ./memcpy --size $((1024*1024))
Start Time: 1747751691.849670 seconds
End Time:   1747751691.850104 seconds
Elapsed Time: 0.000434 seconds
Transferred:  1048576 bytes
Throughput:   2416073732.72 bytes/second
```

2.4 GB/s

POSIX Shared Memory

Demo 2: POSIX shmem

- This modification replaces the memcpy with POSIX shared memory (shmem, mmap)

- The parent starts a child and copies data from the parent to the child via the POSIX shared memory

Demo 2: shmemcpy Key Design Features

- shmem(), mmap()
- fork()
- No additional libraries required (built into libc)

Demo 2 - shmemcpy.c

```
$ gcc -Wall shmemcpy.c -o shmemcpy

$ file shmemcpy
shmemcpy: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV),
dynamically linked, interpreter /lib/ld-linux-aarch64.so.1,
BuildID[sha1]=702e50743c6b7b68e3e122b472374a3c0f9c21a6,
for GNU/Linux 3.7.0, not stripped

$ ldd shmemcpy
   linux-vdso.so.1 (0x0000007fae943000)
   libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000007fae730000)
   /lib/ld-linux-aarch64.so.1 (0x0000007fae906000)
```

Demo 2 - shmemcpy.c Running on RPi Zero

```
./shmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.010406 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   100766480.88 bytes/sec (100.77 MB/sec)

$ ./shmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.009398 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   111574377.53 bytes/sec (111.57 MB/sec)

$ ./shmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.010748 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   97560104.21 bytes/sec (97.56 MB/sec)
```

100 MB/s

Demo 2 - shmemcpy.c Running on RPi Model 4B

```
$ ./shmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.007507 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   139679765.55 bytes/sec (139.68 MB/sec)

$ ./shmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.005269 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   199008540.52 bytes/sec (199.01 MB/sec)

$ ./shmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.007404 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   141622906.54 bytes/sec (141.62 MB/sec)
```

150 MB/s

Demo 2 - shmemcpy.c Running on x86_64

```
$ ./shmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.000607 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   1727472817.13 bytes/sec (1727.47 MB/sec)

$ ./shmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.000547 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   1916957952.47 bytes/sec (1916.96 MB/sec)

$ ./shmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.000618 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   1696724919.09 bytes/sec (1696.72 MB/sec)
```

1.8 GB/s

UDP Sockets

Demo 3: UDP Sockets

- This modification replaces the POSIX Shared Memory Communication with with **UDP socket communication** between the **parent** and **child** processes.

- The parent sends the data over a UDP socket, and the child receives it, recording the end time and calculating throughput.

Demo 2: udpmemcpy Key Design Features

- socket(), recvfrom(), sendto()
- fork()
- No additional libraries required (built into libc)

Demo 3 - udpmemcpy.c

```
$ gcc -Wall udpmemcpy.c -o udpmemcpy

$ ./udpmemcpy
Invalid size specified.

$ ./udpmemcpy --size $((1024*1024))
Parent sendto: Message too long

$ ./udpmemcpy --size $((1024*512))
Parent sendto: Message too long
^C
$ ./udpmemcpy --size $((1024*256))
Parent sendto: Message too long
^C
$ ./udpmemcpy --size $((1024*128))
Parent sendto: Message too long
^C
$ ./udpmemcpy --size $((1024*64))
Parent sendto: Message too long
^C
$ ./udpmemcpy --size $((1024*32))
[Child] Elapsed Time: 0.000656 seconds
[Child] Transferred:  32768 bytes
[Child] Throughput:   49951219.51 bytes/sec (49.95 MB/sec)
```

Key Point!

Notice “**Message too long**”

Demo 3 - udpmemcpy.c Running on RPi Zero

```
$ ./udpmemcpy --size $((1024*32))
[Child] Elapsed Time: 0.001285 seconds
[Child] Transferred:  32768 bytes
[Child] Throughput:   25500389.11 bytes/sec (25.50 MB/sec)

$ ./udpmemcpy --size $((1024*32))
[Child] Elapsed Time: 0.001384 seconds
[Child] Transferred:  32768 bytes
[Child] Throughput:   23676300.58 bytes/sec (23.68 MB/sec)

$ ./udpmemcpy --size $((1024*32))
[Child] Elapsed Time: 0.001526 seconds
[Child] Transferred:  32768 bytes
[Child] Throughput:   21473132.37 bytes/sec (21.47 MB/sec)
```

24 MB/s

Demo 3 - udpmemcpy.c Running on x86/64

```
$ ./udpmemcpy --size $((1024*32))
[Child] Elapsed Time: 0.000088 seconds
[Child] Transferred:  32768 bytes
[Child] Throughput:   372363636.36 bytes/sec (372.36 MB/sec)

$ ./udpmemcpy --size $((1024*32))
[Child] Elapsed Time: 0.000092 seconds
[Child] Transferred:  32768 bytes
[Child] Throughput:   356173913.04 bytes/sec (356.17 MB/sec)

$ ./udpmemcpy --size $((1024*32))
[Child] Elapsed Time: 0.000088 seconds
[Child] Transferred:  32768 bytes
[Child] Throughput:   372363636.36 bytes/sec (372.36 MB/sec)
```

350 MB/s

TCP Sockets

Demo 4: TCP Sockets

- This modification replaces the UDP socket communication with **TCP socket communication** between the **parent** and **child** processes.

- The parent sends the data over a TCP socket, and the child receives it, recording the end time and calculating throughput.

- NOTE: This solves the “message too long” issue related to use of UDP

Demo 2: tcpmemcpy Key Design Features

- socket(), recvfrom(), sendto()
- fork()
- No additional libraries required (built into libc)

Demo 4 - tcpmemcpy.c Build

```
$ gcc -Wall tcpmemcpy.c -o tcpmemcpy

$ ldd tcpmemcpy
   linux-vdso.so.1 (0x0000007f816e2000)
   libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000007f814d0000)
   /lib/ld-linux-aarch64.so.1 (0x0000007f816a5000)
```

Demo 4 - tcpmemcpy.c Run Multiple Times (RPi Zero)

```
./tcpmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.016877 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   62130473.43 bytes/sec (62.13 MB/sec)

$ ./tcpmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.018364 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   57099542.58 bytes/sec (57.10 MB/sec)

$ ./tcpmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.018669 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   56166693.45 bytes/sec (56.17 MB/sec)
```

60 MB/s

Demo 4 - tcpmemcpy.c
Run Multiple Times (RPi Model 4B)

```
$ ./tcpmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.007323 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   143189403.25 bytes/sec (143.19 MB/sec)

$ ./tcpmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.007451 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   140729566.50 bytes/sec (140.73 MB/sec)

$ ./tcpmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.005799 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   180820141.40 bytes/sec (180.82 MB/sec)

$ ./tcpmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.006202 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   169070622.38 bytes/sec (169.07 MB/sec)
```

150 MB/s

Demo 4 - tcpmemcpy.c Run Multiple Times (x86_64)

```
$ ./tcpmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.000602 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   1741820598.01 bytes/sec (1741.82 MB/sec)

$ ./tcpmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.000489 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   2144327198.36 bytes/sec (2144.33 MB/sec)

$ ./tcpmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.000454 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   2309638766.52 bytes/sec (2309.64 MB/sec)

$ ./tcpmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.000449 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   2335358574.61 bytes/sec (2335.36 MB/sec)
```

2.0 GB/s

Dbus

Demo 5: Dbus Overview

- This modification replaces the TCP socket communication with **Dbus communication** between the **parent** and **child** processes.

- The parent sends the data over Dbus, and the child receives it, recording the end time and calculating throughput.

Demo 5: Dbus
Key Design Features

- Dbus Messages Replace TCP Sockets
- Parent process is Dbus client (sends data to child process)
- Child process is Dbus server (receives data from the parent)
- Additional Library Required: libdbus-1

Demo 5 - dbusmemcpy.c Build

```
$ sudo apt install libdbus-1-dev
```

```
$ gcc -Wall dbusmemcpy.c -o dbusmemcpy $(pkg-config --cflags --libs dbus-1)
```

```
$ ldd dbusmemcpy
   linux-vdso.so.1 (0x0000007fa2283000)
   libdbus-1.so.3 => /lib/aarch64-linux-gnu/libdbus-1.so.3 (0x0000007fa21a0000)
   libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000007fa1ff0000)
   libsystemd.so.0 => /lib/aarch64-linux-gnu/libsystemd.so.0 (0x0000007fa1f00000)
   /lib/ld-linux-aarch64.so.1 (0x0000007fa2246000)
   libcap.so.2 => /lib/aarch64-linux-gnu/libcap.so.2 (0x0000007fa1ed0000)
   libgcrypt.so.20 => /lib/aarch64-linux-gnu/libgcrypt.so.20 (0x0000007fa1dc0000)
   liblzma.so.5 => /lib/aarch64-linux-gnu/liblzma.so.5 (0x0000007fa1d70000)
   libzstd.so.1 => /lib/aarch64-linux-gnu/libzstd.so.1 (0x0000007fa1cb0000)
   liblz4.so.1 => /lib/aarch64-linux-gnu/liblz4.so.1 (0x0000007fa1c60000)
   libgpg-error.so.0 => /lib/aarch64-linux-gnu/libgpg-error.so.0 (0x0000007fa1c10000)
```

Demo 5 - dbusmemcpy.c
Run Multiple Times (RPi Model 4B)

```
$ ./dbusmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.037618 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   27874315.49 bytes/sec (27.87 MB/sec)

$ ./dbusmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.032547 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   32217285.77 bytes/sec (32.22 MB/sec)

$ ./dbusmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.054043 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   19402623.84 bytes/sec (19.40 MB/sec)

$ ./dbusmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.022623 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   46349997.79 bytes/sec (46.35 MB/sec)
```

Demo 5 - dbusmemcpy.c Run Multiple Times (RPi Zero)

```
./dbusmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.127306 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   8236658.13 bytes/sec (8.24 MB/sec)

$ ./dbusmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.137478 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   7627227.63 bytes/sec (7.63 MB/sec)

$ ./dbusmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.149565 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   7010838.10 bytes/sec (7.01 MB/sec)
```

ZeroMQ

Demo 6: ZeroMQ Overview

- This modification replaces the Dbus Communication with **ZeroMQ communication** between the **parent** and **child** processes.

- The parent sends the data over ZeroMQ, and the child receives it, recording the end time and calculating throughput.

Demo 6: ZeroMQ
Key Design Features

- ZeroMQ Messages Replace Dbus Messages
- Additional Library Required: libzmq

Demo 6 - zmqmemcpy.c Build

```
$ sudo apt-get install libzmq3-dev
```

```
$ gcc -Wall zmqmemcpy.c -o zmqmemcpy $(pkg-config --cflags --libs dbus-1)
```

```
$ ldd zmqmemcpy
   linux-vdso.so.1 (0x0000007f86d99000)
   libzmq.so.5 => /lib/aarch64-linux-gnu/libzmq.so.5 (0x0000007f86c60000)
   libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000007f86ab0000)
   libbsd.so.0 => /lib/aarch64-linux-gnu/libbsd.so.0 (0x0000007f86a70000)
   libsodium.so.23 => /lib/aarch64-linux-gnu/libsodium.so.23 (0x0000007f86a20000)
   libpgm-5.3.so.0 => /lib/aarch64-linux-gnu/libpgm-5.3.so.0 (0x0000007f869b0000)
   libnorm.so.1 => /lib/aarch64-linux-gnu/libnorm.so.1 (0x0000007f86870000)
   libgssapi_krb5.so.2 => /lib/aarch64-linux-gnu/libgssapi_krb5.so.2 (0x0000007f86800000)
   libstdc++.so.6 => /lib/aarch64-linux-gnu/libstdc++.so.6 (0x0000007f865e0000)
   /lib/ld-linux-aarch64.so.1 (0x0000007f86d5c000)
   libgcc_s.so.1 => /lib/aarch64-linux-gnu/libgcc_s.so.1 (0x0000007f865a0000)
   libmd.so.0 => /lib/aarch64-linux-gnu/libmd.so.0 (0x0000007f86570000)
   libpthread.so.0 => /lib/aarch64-linux-gnu/libpthread.so.0 (0x0000007f86540000)
   libm.so.6 => /lib/aarch64-linux-gnu/libm.so.6 (0x0000007f864a0000)
   libkrb5.so.3 => /lib/aarch64-linux-gnu/libkrb5.so.3 (0x0000007f863b0000)
   libk5crypto.so.3 => /lib/aarch64-linux-gnu/libk5crypto.so.3 (0x0000007f86360000)
   libcom_err.so.2 => /lib/aarch64-linux-gnu/libcom_err.so.2 (0x0000007f86330000)
   libkrb5support.so.0 => /lib/aarch64-linux-gnu/libkrb5support.so.0 (0x0000007f86300000)
   libkeyutils.so.1 => /lib/aarch64-linux-gnu/libkeyutils.so.1 (0x0000007f862d0000)
   libresolv.so.2 => /lib/aarch64-linux-gnu/libresolv.so.2 (0x0000007f862a0000)
```

Demo 5 - zmqmemcpy.c
Run Multiple Times (RPi Model 4B)

```
$ ./zmqmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.019845 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   52838296.80 bytes/sec (52.84 MB/sec)

$ ./zmqmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.014469 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   72470523.19 bytes/sec (72.47 MB/sec)

$ ./zmqmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.013222 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   79305400.09 bytes/sec (79.31 MB/sec)

$ ./zmqmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.016509 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   63515415.83 bytes/sec (63.52 MB/sec)

$ ./zmqmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.025597 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   40964800.56 bytes/sec (40.96 MB/sec)
```

Demo 5 - zmqmemcpy.c Run Multiple Times (RPi Zero)

```
./zmqmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.072017 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   14560117.75 bytes/sec (14.56 MB/sec)

$ ./zmqmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.048771 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   21499989.75 bytes/sec (21.50 MB/sec)

$ ./zmqmemcpy --size $((1024*1024))
[Child] Elapsed Time: 0.040727 seconds
[Child] Transferred:  1048576 bytes
[Child] Throughput:   25746458.12 bytes/sec (25.75 MB/sec)
```

MQTT

Demo 7: MQTT Overview

- This modification replaces the ZeroMQ Communication with **MQTT communication** between the **parent** and **child** processes.

- The parent sends the data over MQTT, and the child receives it, recording the end time and calculating throughput.

Demo 7: MQTT Key Design Features

- MQTT Messages Replace ZeroMQ Messages
- Additional MQTT Broker Required
  - We use mosquitto
- Additional Library Required: paho-mqtt3c

Demo 7 - mqttmemcpy.c Build

```
$ sudo apt install mosquitto
```

```
$ sudo apt install mosquitto-clients
```

```
$ sudo apt install libpaho-mqtt-dev
```

```
$  gcc -Wall mqttmemcpy.c -o mqttmemcpy -lpaho-mqtt3c
```

```
$ ldd mqttmemcpy
   linux-vdso.so.1 (0x0000007f8eadd000)
   libpaho-mqtt3c.so.1 => /lib/aarch64-linux-gnu/libpaho-mqtt3c.so.1 (0x0000007f8e990000)
   libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000007f8e7e0000)
   /lib/ld-linux-aarch64.so.1 (0x0000007f8eaa0000)
```

Demo 7 - mqttmemcpy.c Run

In Console 1

```
$ mosquitto_sub -h localhost -t ipc/data
```

In Console 1

```
$ mosquitto_pub -h localhost -t ipc/data -m "hello"
```

```
$ systemctl status mosquitto
* mosquitto.service - Mosquitto MQTT Broker
     Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-05-20 04:49:55 PDT; 8min ago
. . .
```

```
$ systemctl status mosquitto
* mosquitto.service - Mosquitto MQTT Broker
     Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-05-20 04:49:55 PDT; 8min ago
. . .
```

Backup Slides

memcpy.c

memcpy.c - Part 1

```
// memcpy.c
//
// For questions/support: norman.mcentire@gmail.com
//
// To build: gcc -Wall memcpy.c -o memcpy
//
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <sys/time.h>
#include <getopt.h>

int main(int argc, char *argv[]) {

    int size = 0;

    // Parse command-line arguments
    static struct option long_options[] = {
        {"size", required_argument, 0, 's'},
        {0, 0, 0, 0}
    };
```

memcpy.c - Part 2

```
    int option_index = 0;
    int c;

    while ((c = getopt_long(argc, argv, "s:", long_options, &option_index)) != -1) {
        switch (c) {
            case 's':
                size = atoi(optarg);
                break;
            default:
                fprintf(stderr, "Usage: %s --size NUMBER\n", argv[0]);
                return EXIT_FAILURE;
        }
    }

    if (size <= 0) {
        fprintf(stderr, "Invalid size specified.\n");
        return EXIT_FAILURE;
    }

    // Allocate source and destination buf_data_t buffers
    buf_data_t *src = malloc(sizeof(buf_data_t) + size);
    buf_data_t *dst = malloc(sizeof(buf_data_t) + size);

    if (!src || !dst) {
        fprintf(stderr, "Memory allocation failed.\n");
        free(src);
        free(dst);
        return EXIT_FAILURE;
    }
```

memcpy.c - Part 3

```
    // Store size in the src buffer metadata
    src->size = size;

    // Fill the source data buffer with values 0, 1, 2, ...
    for (int i = 0; i < size; i++) {
        src->data[i] = (uint8_t)i;
    }

    gettimeofday(&src->start, NULL);
    printf("Start Time: %ld.%06ld seconds\n", src->start.tv_sec, src->start.tv_usec);

    // Copy the entire source buffer into destination buffer
    memcpy(dst, src, sizeof(buf_data_t) + size);

    gettimeofday(&dst->end, NULL);
    printf("End Time:   %ld.%06ld seconds\n", dst->end.tv_sec, dst->end.tv_usec);

    // Calculate elapsed time
    long seconds = dst->end.tv_sec - src->start.tv_sec;
    long microseconds = dst->end.tv_usec - src->start.tv_usec;
    if (microseconds < 0) {
        seconds--;
        microseconds += 1000000;
    }
```

memcpy.c - Part 4

```
    double elapsed_time_sec = seconds + microseconds / 1e6;
    size_t bytes_copied = size;
    double bytes_per_sec = elapsed_time_sec > 0 ? (bytes_copied / elapsed_time_sec) : 0;
    double megabytes_per_sec = bytes_per_sec / 1000000.0;

    printf("Elapsed Time: %.6f seconds\n", elapsed_time_sec);
    printf("Transferred:  %zu bytes\n", bytes_copied);
    printf("Throughput:   %.2f bytes/second\n", bytes_per_sec);
    printf("              %.2f MB/second\n", megabytes_per_sec);

    // Clean up
    free(src);
    free(dst);

    return EXIT_SUCCESS;
}
```

shmemcpy.c

shmemcpy.c - Part 1

```
// shmemcpy.c
//
// For questions/support: norman.mcentire@gmail.com
//
// To build: gcc -Wall shmemcpy.c -o shmemcpy
//
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <getopt.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <errno.h>

#define SHM_NAME "/my_shared_buf"

typedef struct {
    struct timeval start;
    struct timeval end;
    uint32_t size;
    uint8_t data[];
} buf_data_t;
```

shmemcpy.c - Part 2

```
volatile sig_atomic_t sigusr1_received = 0;
volatile sig_atomic_t sigio_received = 0;

void handle_sigusr1(int sig) {
    sigusr1_received = 1;
}

void handle_sigio(int sig) {
    sigio_received = 1;
}
```

shmemcpy.c - Part 3

```
int main(int argc, char *argv[]) {
    int size = 0;

    static struct option long_options[] = {
        {"size", required_argument, 0, 's'},
        {0, 0, 0, 0}
    };

    while (1) {
        int option_index = 0;
        int c = getopt_long(argc, argv, "s:", long_options, &option_index);
        if (c == -1) break;

        switch (c) {
            case 's':
                size = atoi(optarg);
                break;
            default:
                fprintf(stderr, "Usage: %s --size NUMBER\n", argv[0]);
                return EXIT_FAILURE;
        }
    }
```

shmemcpy.c - Part 4

```
    if (size <= 0) {
        fprintf(stderr, "Invalid size specified.\n");
        return EXIT_FAILURE;
    }

    size_t total_size = sizeof(buf_data_t) + size;

    // Allocate src in heap
    buf_data_t *src = malloc(total_size);
    if (!src) {
        perror("malloc");
        return EXIT_FAILURE;
    }
    src->size = size;
    for (int i = 0; i < size; i++) {
        src->data[i] = (uint8_t)i;
    }

    // Create and set up shared memory
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd < 0) {
        perror("shm_open");
        free(src);
        return EXIT_FAILURE;
    }
```

shmemcpy.c - Part 5

```
    if (ftruncate(shm_fd, total_size) < 0) {
        perror("ftruncate");
        shm_unlink(SHM_NAME);
        free(src);
        return EXIT_FAILURE;
    }

    pid_t child_pid = fork();
    if (child_pid < 0) {
        perror("fork");
        shm_unlink(SHM_NAME);
        free(src);
        return EXIT_FAILURE;
    }
```

shmemcpy.c - Part 6

```
    if (child_pid == 0) {
        // --- Child Process ---
        signal(SIGIO, handle_sigio);

        // Open and map shared memory
        int fd = shm_open(SHM_NAME, O_RDWR, 0666);
        if (fd < 0) {
            perror("child shm_open");
            exit(EXIT_FAILURE);
        }

        buf_data_t *dst = mmap(NULL, total_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
        if (dst == MAP_FAILED) {
            perror("child mmap");
            exit(EXIT_FAILURE);
        }

        // Notify parent process that we're ready
        kill(getppid(), SIGUSR1);

        // Wait for SIGIO from parent
        while (!sigio_received) pause();
```

shmemcpy.c - Part 7

```
    // Record end time
    gettimeofday(&dst->end, NULL);

    // Compute and display metrics
    long sec = dst->end.tv_sec - dst->start.tv_sec;
    long usec = dst->end.tv_usec - dst->start.tv_usec;
    if (usec < 0) {
        sec--;
        usec += 1000000;
    }
    double elapsed = sec + usec / 1e6;
    size_t bytes = dst->size;
    double bps = elapsed > 0 ? (bytes / elapsed) : 0;
    double mbps = bps / 1000000.0;

    printf("[Child] Elapsed Time: %.6f seconds\n", elapsed);
    printf("[Child] Transferred:  %zu bytes\n", bytes);
    printf("[Child] Throughput:   %.2f bytes/sec (%.2f MB/sec)\n", bps, mbps);

    munmap(dst, total_size);
    close(fd);
    exit(EXIT_SUCCESS);
```

shmemcpy.c - Part 8

```
    } else {
        // --- Parent Process ---
        signal(SIGUSR1, handle_sigusr1);

        // Wait for SIGUSR1 from child
        while (!sigusr1_received) pause();

        // Map the shared memory
        buf_data_t *dst = mmap(NULL, total_size, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
        if (dst == MAP_FAILED) {
            perror("parent mmap");
            shm_unlink(SHM_NAME);
            free(src);
            return EXIT_FAILURE;
        }

        // Record start time and copy to shared memory
        gettimeofday(&src->start, NULL);
        memcpy(dst, src, total_size);

        // Notify child
        kill(child_pid, SIGIO);
```

shmemcpy.c - Part 9

```
        // Cleanup
        wait(NULL);
        munmap(dst, total_size);
        close(shm_fd);
        shm_unlink(SHM_NAME);
        free(src);
    }

    return EXIT_SUCCESS;
}
```

udpmemcpy.c

udpmemcpy.c - Part 1

```
// udpmemcpy.c
//
// For questions/support: norman.mcentire@gmail.com
//
// To build: gcc -Wall udpmemcpy.c -o udpmemcpy
//
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <getopt.h>
#include <signal.h>
#include <errno.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define UDP_PORT 54321
#define LOCALHOST "127.0.0.1"

typedef struct {
    struct timeval start;
    struct timeval end;
    uint32_t size;
    uint8_t data[];
} buf_data_t;
```

udpmemcpy.c - Part 2

```
volatile sig_atomic_t sigusr1_received = 0;

void handle_sigusr1(int sig) {
    sigusr1_received = 1;
}

int main(int argc, char *argv[]) {
    int size = 0;

    static struct option long_options[] = {
        {"size", required_argument, 0, 's'},
        {0, 0, 0, 0}
    };
```

udpmemcpy.c - Part 3

```
    while (1) {
        int option_index = 0;
        int c = getopt_long(argc, argv, "s:", long_options, &option_index);
        if (c == -1) break;

        switch (c) {
            case 's':
                size = atoi(optarg);
                break;
            default:
                fprintf(stderr, "Usage: %s --size NUMBER\n", argv[0]);
                return EXIT_FAILURE;
        }
    }

    if (size <= 0) {
        fprintf(stderr, "Invalid size specified.\n");
        return EXIT_FAILURE;
    }

    size_t total_size = sizeof(buf_data_t) + size;
```

udpmemcpy.c - Part 4

```
    // Allocate and prepare source buffer
    buf_data_t *src = malloc(total_size);
    if (!src) {
        perror("malloc");
        return EXIT_FAILURE;
    }
    src->size = size;
    for (int i = 0; i < size; i++) {
        src->data[i] = (uint8_t)i;
    }

    pid_t child_pid = fork();
    if (child_pid < 0) {
        perror("fork");
        free(src);
        return EXIT_FAILURE;
    }
```

udpmemcpy.c - Part 4

```
    if (child_pid == 0) {
        // --- Child Process ---
        struct sockaddr_in addr;
        int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
        if (sockfd < 0) {
            perror("Child socket");
            exit(EXIT_FAILURE);
        }

        memset(&addr, 0, sizeof(addr));
        addr.sin_family = AF_INET;
        addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
        addr.sin_port = htons(UDP_PORT);

        if (bind(sockfd, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
            perror("Child bind");
            close(sockfd);
            exit(EXIT_FAILURE);
        }

        // Notify parent
        kill(getppid(), SIGUSR1);
```

udpmemcpy.c - Part 5

```
        // Allocate destination buffer
        buf_data_t *dst = malloc(total_size);
        if (!dst) {
            perror("Child malloc");
            close(sockfd);
            exit(EXIT_FAILURE);
        }

        ssize_t received = recvfrom(sockfd, dst, total_size, 0, NULL, NULL);
        if (received < 0) {
            perror("Child recvfrom");
            free(dst);
            close(sockfd);
            exit(EXIT_FAILURE);
        }
```

udpmemcpy.c - Part 6

```
        gettimeofday(&dst->end, NULL);

        // Calculate elapsed time
        long sec = dst->end.tv_sec - dst->start.tv_sec;
        long usec = dst->end.tv_usec - dst->start.tv_usec;
        if (usec < 0) {
            sec--;
            usec += 1000000;
        }
        double elapsed = sec + usec / 1e6;
        size_t bytes = dst->size;
        double bps = elapsed > 0 ? (bytes / elapsed) : 0;
        double mbps = bps / 1000000.0;

        printf("[Child] Elapsed Time: %.6f seconds\n", elapsed);
        printf("[Child] Transferred:  %zu bytes\n", bytes);
        printf("[Child] Throughput:   %.2f bytes/sec (%.2f MB/sec)\n", bps, mbps);

        free(dst);
        close(sockfd);
        exit(EXIT_SUCCESS);
```

udpmemcpy.c - Part 7

```
    } else {
        // --- Parent Process ---
        signal(SIGUSR1, handle_sigusr1);

        while (!sigusr1_received) pause();

        // Create socket for sending
        int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
        if (sockfd < 0) {
            perror("Parent socket");
            free(src);
            return EXIT_FAILURE;
        }

        struct sockaddr_in addr;
        memset(&addr, 0, sizeof(addr));
        addr.sin_family = AF_INET;
        addr.sin_port = htons(UDP_PORT);
        addr.sin_addr.s_addr = inet_addr(LOCALHOST);
```

udpmemcpy.c - Part 8

```
        // Get start time and send buffer
        gettimeofday(&src->start, NULL);

        ssize_t sent = sendto(sockfd, src, total_size, 0,
                              (struct sockaddr *)&addr, sizeof(addr));
        if (sent < 0) {
            perror("Parent sendto");
        }

        close(sockfd);
        wait(NULL);
        free(src);
    }

    return EXIT_SUCCESS;
}
```

tcpmemcpy.c

tcpmemcpy.c - Part 1

```
// tcpmemcpy.c
//
// For questions/support: norman.mcentire@gmail.com
//
// To build: gcc -Wall tcpmemcpy.c -o tcpmemcpy
//
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <getopt.h>
#include <signal.h>
#include <errno.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define TCP_PORT 54321
#define LOCALHOST "127.0.0.1"

typedef struct {
    struct timeval start;
    struct timeval end;
    uint32_t size;
    uint8_t data[];
} buf_data_t;
```

tcpmemcpy.c - Part 2

```
volatile sig_atomic_t sigusr1_received = 0;

void handle_sigusr1(int sig) {
    sigusr1_received = 1;
}

ssize_t full_write(int fd, const void *buf, size_t count) {
    size_t written = 0;
    while (written < count) {
        ssize_t res = write(fd, (char *)buf + written, count - written);
        if (res <= 0) return res;
        written += res;
    }
    return written;
}

ssize_t full_read(int fd, void *buf, size_t count) {
    size_t read_bytes = 0;
    while (read_bytes < count) {
        ssize_t res = read(fd, (char *)buf + read_bytes, count - read_bytes);
        if (res <= 0) return res;
        read_bytes += res;
    }
    return read_bytes;
}
```

tcpmemcpy.c - Part 3

```
int main(int argc, char *argv[]) {
    int size = 0;

    static struct option long_options[] = {
        {"size", required_argument, 0, 's'},
        {0, 0, 0, 0}
    };

    while (1) {
        int option_index = 0;
        int c = getopt_long(argc, argv, "s:", long_options, &option_index);
        if (c == -1) break;

        switch (c) {
            case 's':
                size = atoi(optarg);
                break;
            default:
                fprintf(stderr, "Usage: %s --size NUMBER\n", argv[0]);
                return EXIT_FAILURE;
        }
    }
```

tcpmemcpy.c - Part 4

```
    if (size <= 0) {
        fprintf(stderr, "Invalid size specified.\n");
        return EXIT_FAILURE;
    }

    size_t total_size = sizeof(buf_data_t) + size;

    buf_data_t *src = malloc(total_size);
    if (!src) {
        perror("malloc");
        return EXIT_FAILURE;
    }

    src->size = size;
    for (int i = 0; i < size; i++) {
        src->data[i] = (uint8_t)i;
    }

    pid_t child_pid = fork();
    if (child_pid < 0) {
        perror("fork");
        free(src);
        return EXIT_FAILURE;
    }
```

tcpmemcpy.c - Part 5

```
    if (child_pid == 0) {
        // --- Child Process (TCP Server) ---
        int server_fd = socket(AF_INET, SOCK_STREAM, 0);
        if (server_fd < 0) {
            perror("Child socket");
            exit(EXIT_FAILURE);
        }

        int opt = 1;
        setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

        struct sockaddr_in addr;
        memset(&addr, 0, sizeof(addr));
        addr.sin_family = AF_INET;
        addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
        addr.sin_port = htons(TCP_PORT);

        if (bind(server_fd, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
            perror("Child bind");
            close(server_fd);
            exit(EXIT_FAILURE);
        }
```

tcpmemcpy.c - Part 6

```
        if (listen(server_fd, 1) < 0) {
            perror("Child listen");
            close(server_fd);
            exit(EXIT_FAILURE);
        }

        kill(getppid(), SIGUSR1); // Notify parent

        int client_fd = accept(server_fd, NULL, NULL);
        if (client_fd < 0) {
            perror("Child accept");
            close(server_fd);
            exit(EXIT_FAILURE);
        }

        buf_data_t *dst = malloc(total_size);
        if (!dst) {
            perror("Child malloc");
            close(client_fd);
            close(server_fd);
            exit(EXIT_FAILURE);
        }
```

tcpmemcpy.c - Part 7

```
    ssize_t n = full_read(client_fd, dst, total_size);
    if (n != total_size) {
        fprintf(stderr, "Child: Failed to read complete buffer\n");
        free(dst);
        close(client_fd);
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    gettimeofday(&dst->end, NULL);

    long sec = dst->end.tv_sec - dst->start.tv_sec;
    long usec = dst->end.tv_usec - dst->start.tv_usec;
    if (usec < 0) {
        sec--;
        usec += 1000000;
    }

    double elapsed = sec + usec / 1e6;
    double bps = elapsed > 0 ? (dst->size / elapsed) : 0;
    double mbps = bps / 1e6;

    printf("[Child] Elapsed Time: %.6f seconds\n", elapsed);
    printf("[Child] Transferred:  %u bytes\n", dst->size);
    printf("[Child] Throughput:   %.2f bytes/sec (%.2f MB/sec)\n", bps, mbps);

    free(dst);
    close(client_fd);
    close(server_fd);
    exit(EXIT_SUCCESS);
```

tcpmemcpy.c - Part 8

```
} else {
   // --- Parent Process (TCP Client) ---
   signal(SIGUSR1, handle_sigusr1);

   while (!sigusr1_received) pause(); // Wait for child to bind

   int sockfd = socket(AF_INET, SOCK_STREAM, 0);
   if (sockfd < 0) {
       perror("Parent socket");
       free(src);
       return EXIT_FAILURE;
   }

   struct sockaddr_in serv_addr;
   memset(&serv_addr, 0, sizeof(serv_addr));
   serv_addr.sin_family = AF_INET;
   serv_addr.sin_port = htons(TCP_PORT);
   inet_pton(AF_INET, LOCALHOST, &serv_addr.sin_addr);

   if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
       perror("Parent connect");
       close(sockfd);
       free(src);
       return EXIT_FAILURE;
   }
```

tcpmemcpy.c - Part 9

```
        gettimeofday(&src->start, NULL);

        ssize_t sent = full_write(sockfd, src, total_size);
        if (sent != total_size) {
            fprintf(stderr, "Parent: Failed to send complete buffer\n");
        }

        close(sockfd);
        wait(NULL);
        free(src);
    }

    return EXIT_SUCCESS;
}
```

dbusmemcpy.c

dbusmemcpy.c - Part 1

```
// dbusmemcpy.c
//
// For questions/support: norman.mcentire@gmail.com
//
// To build: gcc -Wall dbusmemcpy.c -o dbusmemcpy $(pkg-config --cflags --libs dbus-1)
//

#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <getopt.h>
#include <signal.h>
#include <errno.h>
#include <sys/wait.h>
#include <dbus/dbus.h>

typedef struct {
    struct timeval start;
    struct timeval end;
    uint32_t size;
    uint8_t data[];
} buf_data_t;
```

dbusmemcpy.c - Part 2

```
volatile sig_atomic_t sigusr1_received = 0;

void handle_sigusr1(int sig) {
    sigusr1_received = 1;
}

int main(int argc, char *argv[]) {
    int size = 0;

    static struct option long_options[] = {
        {"size", required_argument, 0, 's'},
        {0, 0, 0, 0}
    };

    while (1) {
        int option_index = 0;
        int c = getopt_long(argc, argv, "s:", long_options, &option_index);
        if (c == -1) break;

        switch (c) {
            case 's':
                size = atoi(optarg);
                break;
            default:
                fprintf(stderr, "Usage: %s --size NUMBER\n", argv[0]);
                return EXIT_FAILURE;
        }
    }
```

dbusmemcpy.c - Part 3

```
    if (size <= 0) {
        fprintf(stderr, "Invalid size specified.\n");
        return EXIT_FAILURE;
    }

    size_t total_size = sizeof(buf_data_t) + size;

    buf_data_t *src = malloc(total_size);
    if (!src) {
        perror("malloc");
        return EXIT_FAILURE;
    }

    src->size = size;
    for (int i = 0; i < size; i++) {
        src->data[i] = (uint8_t)i;
    }

    pid_t child_pid = fork();
    if (child_pid < 0) {
        perror("fork");
        free(src);
        return EXIT_FAILURE;
    }
```

dbusmemcpy.c - Part 4

```
    if (child_pid == 0) {
        // --- Child Process (D-Bus Server) ---
        DBusError err;
        dbus_error_init(&err);

        DBusConnection *conn = dbus_bus_get(DBUS_BUS_SESSION, &err);
        if (!conn) {
            fprintf(stderr, "Failed to connect to the D-Bus session bus: %s\n", err.message);
            dbus_error_free(&err);
            exit(EXIT_FAILURE);
        }

        dbus_bus_request_name(conn, "org.example.DBusTransfer", DBUS_NAME_FLAG_REPLACE_EXISTING,
&err);
        if (dbus_error_is_set(&err)) {
            fprintf(stderr, "Failed to request name on D-Bus: %s\n", err.message);
            dbus_error_free(&err);
            dbus_connection_unref(conn);
            exit(EXIT_FAILURE);
        }

        // Signal parent that we are ready
        kill(getppid(), SIGUSR1);
```

dbusmemcpy.c - Part 5

```
while (1) {
    dbus_connection_read_write(conn, 100);
    DBusMessage *msg = dbus_connection_pop_message(conn);
    if (!msg) continue;

    if (dbus_message_is_method_call(msg, "org.example.DBusTransfer", "TransferData")) {
        DBusMessageIter args;
        dbus_message_iter_init(msg, &args);

        uint32_t received_size = 0;
        const uint8_t *data_ptr;
        int array_len = 0;

        if (dbus_message_iter_get_arg_type(&args) != DBUS_TYPE_UINT32) {
            fprintf(stderr, "Child: Expected uint32_t\n");
            dbus_message_unref(msg);
            continue;
        }

        dbus_message_iter_get_basic(&args, &received_size);
        dbus_message_iter_next(&args);

        if (dbus_message_iter_get_arg_type(&args) != DBUS_TYPE_ARRAY) {
            fprintf(stderr, "Child: Expected byte array\n");
            dbus_message_unref(msg);
            continue;
        }
```

dbusmemcpy.c - Part 6

```
                DBusMessageIter sub_iter;
                dbus_message_iter_recurse(&args, &sub_iter);
                dbus_message_iter_get_fixed_array(&sub_iter, &data_ptr, &array_len);

                if (array_len < sizeof(struct timeval)) {
                    fprintf(stderr, "Child: Incomplete timing info\n");
                    dbus_message_unref(msg);
                    continue;
                }

                struct timeval start;
                memcpy(&start, data_ptr, sizeof(struct timeval));

                struct timeval end;
                gettimeofday(&end, NULL);

                long sec = end.tv_sec - start.tv_sec;
                long usec = end.tv_usec - start.tv_usec;
                if (usec < 0) {
                    sec--;
                    usec += 1000000;
                }
```

dbusmemcpy.c - Part 7

```
            double elapsed = sec + usec / 1e6;
            double bps = elapsed > 0 ? (received_size / elapsed) : 0;
            double mbps = bps / 1e6;

            printf("[Child] Elapsed Time: %.6f seconds\n", elapsed);
            printf("[Child] Transferred:  %u bytes\n", received_size);
            printf("[Child] Throughput:   %.2f bytes/sec (%.2f MB/sec)\n", bps, mbps);

            dbus_message_unref(msg);
            break; // One-shot transfer; exit after report
        }

        dbus_message_unref(msg);
    }

    dbus_connection_unref(conn);
    exit(EXIT_SUCCESS);
```

dbusmemcpy.c - Part 8

```
    } else {
        // --- Parent Process (D-Bus Client) ---
        signal(SIGUSR1, handle_sigusr1);
        while (!sigusr1_received) pause();

        DBusError err;
        dbus_error_init(&err);

        DBusConnection *conn = dbus_bus_get(DBUS_BUS_SESSION, &err);
        if (!conn) {
            fprintf(stderr, "Parent: D-Bus connection failed: %s\n", err.message);
            dbus_error_free(&err);
            free(src);
            return EXIT_FAILURE;
        }

        DBusMessage *msg = dbus_message_new_method_call(
            "org.example.DBusTransfer",
            "/org/example/DBusTransfer",
            "org.example.DBusTransfer",
            "TransferData"
        );
```

dbusmemcpy.c - Part 9

```
    if (!msg) {
        fprintf(stderr, "Parent: Failed to create message\n");
        dbus_connection_unref(conn);
        free(src);
        return EXIT_FAILURE;
    }

    // Prepare payload: [start_time | data[]]
    gettimeofday(&src->start, NULL);
    size_t payload_size = sizeof(struct timeval) + size;
    uint8_t *payload = malloc(payload_size);
    if (!payload) {
        perror("malloc");
        dbus_message_unref(msg);
        dbus_connection_unref(conn);
        free(src);
        return EXIT_FAILURE;
    }

    memcpy(payload, &src->start, sizeof(struct timeval));
    memcpy(payload + sizeof(struct timeval), src->data, size);
```

dbusmemcpy.c - Part 10

```
        DBusMessageIter args;
        dbus_message_iter_init_append(msg, &args);
        dbus_message_iter_append_basic(&args, DBUS_TYPE_UINT32, &src->size);

    DBusMessageIter array_iter;
    dbus_message_iter_open_container(&args, DBUS_TYPE_ARRAY, "y", &array_iter);
    dbus_message_iter_append_fixed_array(&array_iter, DBUS_TYPE_BYTE, &payload, payload_size);
    dbus_message_iter_close_container(&args, &array_iter);

        if (!dbus_connection_send(conn, msg, NULL)) {
            fprintf(stderr, "Parent: Failed to send message\n");
        }

        dbus_connection_flush(conn);
        dbus_message_unref(msg);
        dbus_connection_unref(conn);

        free(payload);
        free(src);
        wait(NULL);
    }

    return EXIT_SUCCESS;
}
```

zmqmemcpy.c

zmqmemcpy.c - Part 1

```
// memcpy.c
//
// For questions/support: norman.mcentire@gmail.com
//
// To build: gcc -Wall zmqmemcpy.c -o zmqmemcpy -lzmq
//
#define _GNU_SOURCE
#include <zmq.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <getopt.h>
#include <signal.h>
#include <errno.h>
#include <sys/wait.h>

typedef struct {
    struct timeval start;
    struct timeval end;
    uint32_t size;
    uint8_t data[];
} buf_data_t;
```

zmqmemcpy.c - Part 2

```
volatile sig_atomic_t sigusr1_received = 0;

void handle_sigusr1(int sig) {
    sigusr1_received = 1;
}

int main(int argc, char *argv[]) {
    int size = 0;

    static struct option long_options[] = {
        {"size", required_argument, 0, 's'},
        {0, 0, 0, 0}
    };

    while (1) {
        int option_index = 0;
        int c = getopt_long(argc, argv, "s:", long_options, &option_index);
        if (c == -1) break;

        switch (c) {
            case 's':
                size = atoi(optarg);
                break;
            default:
                fprintf(stderr, "Usage: %s --size NUMBER\n", argv[0]);
                return EXIT_FAILURE;
        }
    }
```

zmqmemcpy.c - Part 3

```
    if (size <= 0) {
        fprintf(stderr, "Invalid size specified.\n");
        return EXIT_FAILURE;
    }

    size_t total_size = sizeof(buf_data_t) + size;

    buf_data_t *src = malloc(total_size);
    if (!src) {
        perror("malloc");
        return EXIT_FAILURE;
    }

    src->size = size;
    for (int i = 0; i < size; i++) {
        src->data[i] = (uint8_t)i;
    }

    pid_t child_pid = fork();
    if (child_pid < 0) {
        perror("fork");
        free(src);
        return EXIT_FAILURE;
    }
```

zmqmemcpy.c - Part 4

```
    if (child_pid == 0) {
        // --- Child Process (Receiver) ---
        void *context = zmq_ctx_new();
        void *receiver = zmq_socket(context, ZMQ_PULL);
        int rc = zmq_bind(receiver, "tcp://127.0.0.1:5555");
        if (rc != 0) {
            perror("zmq_bind");
            exit(EXIT_FAILURE);
        }

        kill(getppid(), SIGUSR1);  // Notify parent

        // Allocate buffer
        size_t max_recv = sizeof(struct timeval) + size;
        uint8_t *recv_buf = malloc(max_recv);
        if (!recv_buf) {
            perror("malloc");
            exit(EXIT_FAILURE);
        }

        int received = zmq_recv(receiver, recv_buf, max_recv, 0);
        if (received < (int)sizeof(struct timeval)) {
            fprintf(stderr, "[Child] Incomplete data received\n");
            exit(EXIT_FAILURE);
        }
```

zmqmemcpy.c - Part 5

```
        struct timeval start, end;
        memcpy(&start, recv_buf, sizeof(struct timeval));
        gettimeofday(&end, NULL);

        long sec = end.tv_sec - start.tv_sec;
        long usec = end.tv_usec - start.tv_usec;
        if (usec < 0) {
            sec--;
            usec += 1000000;
        }

        double elapsed = sec + usec / 1e6;
        double bps = elapsed > 0 ? (size / elapsed) : 0;
        double mbps = bps / 1e6;

        printf("[Child] Elapsed Time: %.6f seconds\n", elapsed);
        printf("[Child] Transferred:  %d bytes\n", size);
        printf("[Child] Throughput:   %.2f bytes/sec (%.2f MB/sec)\n", bps, mbps);

        free(recv_buf);
        zmq_close(receiver);
        zmq_ctx_term(context);
        exit(EXIT_SUCCESS);
```

zmqmemcpy.c - Part 6

```
    } else {
        // --- Parent Process (Sender) ---
        signal(SIGUSR1, handle_sigusr1);
        while (!sigusr1_received) pause();

        void *context = zmq_ctx_new();
        void *sender = zmq_socket(context, ZMQ_PUSH);
        if (zmq_connect(sender, "tcp://127.0.0.1:5555") != 0) {
            perror("zmq_connect");
            free(src);
            return EXIT_FAILURE;
        }

        // Create payload: [start_time][data]
        gettimeofday(&src->start, NULL);
        size_t payload_size = sizeof(struct timeval) + size;
        uint8_t *payload = malloc(payload_size);
        memcpy(payload, &src->start, sizeof(struct timeval));
        memcpy(payload + sizeof(struct timeval), src->data, size);
```

zmqmemcpy.c - Part 7

```
        zmq_send(sender, payload, payload_size, 0);

        free(payload);
        zmq_close(sender);
        zmq_ctx_term(context);
        free(src);
        wait(NULL);
    }

    return EXIT_SUCCESS;
}
```