Brennan Mahoney

bmm1@bu.edu | (774) 280-0845 | https://www.linkedin.com/in/brennan-mahoney | https://github.com/bmahoney1

Education

Boston University College of Engineering Expected May 2025

Bachelor of Science in Computer Engineering (Concentration in Machine Learning) GPA: 3.79/4.00

Relevant Coursework

Deep Learning, Software Engineering, Programming for Engineers, Operating Systems, Machine Learning, Computer Organization, Computational Linear Algebra, Probability-Statistics-Data Science, Algorithms, Logic Design

Skills

Software: Pytorch, Python, C, C++, Java, HTML, JavaScript, CSS, MATLAB, Arduino Code, Onshape, SwiftUI, Verilog
Technical: Soldering, Wiring, Hardware imaging

Work Experience

Software Engineering Intern (Applications and IT Team) Summer 2023
EG America
- Deployed software using Tanium, software development, hardware imaging
- Utilized Python and .bat files for credit processor speed improvements and point of sales bugs
- Troubleshot technology with pin pads, credit processors and fuel technology

Software Engineering Intern (Applications and IT Team) December 2023 – January 2024
EG America
- Applied HTML, CSS and JavaScript for point of sales display and content
- Worked with git for source code management and collaboration

Engineering/Tech Substitute Teacher December 2022 – January 2023
Northbridge Public Schools
- Conducted projects with students, aiding with the structure and design of projects

Projects/Extra Curricular

HolyFit IOS App (Personal Developer App)
- Applied SwiftUI to write front-end user interface including the ability to make posts, comments and likes
- Programmed back-end using Firebase that stores user data for usernames, passwords and account information

Audio Style Transfer via Neural Network (Team Leader)
- Collaborated on a team to implement a Convolutional Neural Network (CNN) setup with a VGG19 network for style transfer in audio spectrograms, bridging the gap between image processing and audio data
- Employed machine learning models for audio synthesis, specifically using the WaveNet vocoder, to convert mel spectrograms back to audio, demonstrating proficiency in both audio analysis and synthesis

Simple Shell Project
- Created a Basic Shell written in C to accept and execute commands, redirect input/output, and conduct processes in the background

File System
- Implemented a file system in C with functions for creating, mounting, unmounting, opening, closing, block reading/writing, deleting, and listing files.
- Designed data structures for managing file metadata, file descriptors, and a simulated file allocation table (FAT) within a block-based virtual disk environment.
- Implemented additional file system operations like file seek (`fs_lseek`), file truncation (`fs_truncate`), and retrieving file size (`fs_get_filesize`), enhancing file manipulation capabilities.

Thread Local Storage
- Developed a thread-local storage (TLS) system using C to manage memory spaces unique to each thread, enhancing data isolation and thread safety in concurrent applications.
- Integrated system-level programming techniques including memory management and multi-threading using POSIX threads (pthread) and memory-mapped files.
- Designed a hash table mechanism for efficient mapping of threads to their corresponding TLS
- Created functions for TLS creation, destruction, read, write, and cloning operations, supporting fundamental operations like memory protection, thread-specific data isolation, and efficient memory usage with reference counting.

Pi Kappa Alpha Fraternity (Recruitment Chairman)
- Have taken part in philanthropy helping raise over $100,000 for inclusive foundations

Multithreading with Semaphores

- Engineered a custom threading library in C, including thread creation, execution, and synchronization mechanisms, compatible with POSIX standards.
- Implemented thread management functionalities using `setjmp` and `longjmp` for context switching, and `sigprocmask` for signal handling to control thread execution flow.
- Designed and managed a thread table for tracking up to 128 threads, each with individual state, stack, and execution context, enhancing the system's ability to handle concurrent operations.
- Developed semaphore mechanisms with initialization, wait, and post operations to enable inter-thread communication and synchronization, ensuring thread-safe interactions.
- Integrated system calls and assembly code for low-level operations including thread-specific data handling and pointer mangling for security purposes.
- Utilized UNIX signals to implement pre-emptive scheduling based on timer interrupts, ensuring fair CPU time allocation among threads.
- Implemented a robust thread scheduling algorithm that manages thread states (ready, running, blocked, exited) to facilitate smooth context switches.
- Ensured thread safety and synchronization using mutex locks and signal masks to prevent race conditions and maintain data integrity across concurrent executions.
- Managed memory for thread stacks dynamically, allowing threads to have isolated execution stacks for safety and stability.

File System

- Implemented a file system in C with functions for creating, mounting, unmounting, opening, closing, block reading/writing, deleting, and listing files.
- Designed data structures for managing file metadata, file descriptors, and a simulated file allocation table (FAT) within a block-based virtual disk environment.
- Implemented additional file system operations like file seek (`fs_lseek`), file truncation (`fs_truncate`), and retrieving file size (`fs_get_filesize`), enhancing file manipulation capabilities.

Thread Local Storage

- Developed a thread-local storage (TLS) system using C to manage memory spaces unique to each thread, enhancing data isolation and thread safety in concurrent applications.
- Integrated system-level programming techniques including memory management and multi-threading using POSIX threads (pthread) and memory-mapped files.
- Designed a hash table mechanism for efficient mapping of threads to their corresponding TLS, enabling quick retrieval and management.
- Implemented page fault handling to terminate threads that access unauthorized memory, using UNIX signal handling (SIGSEGV, SIGBUS) to manage segmentation and bus errors.
- Utilized `mmap` and `munmap` for dynamic memory allocation of TLS pages, ensuring protected memory access with `mprotect`.
- Created functions for TLS creation, destruction, read, write, and cloning operations, supporting fundamental operations like memory protection, thread-specific data isolation, and efficient memory usage with reference counting.
- Leveraged copy-on-write semantics during write operations to minimize memory usage and maximize performance.
- Designed a cloning feature allowing threads to duplicate TLS configurations, enabling efficient thread replication with shared memory pages.

Multithreading with Semaphores

- Engineered a custom threading library in C, including thread creation, execution, and synchronization mechanisms, compatible with POSIX standards.
- Implemented thread management functionalities using `setjmp` and `longjmp` for context switching, and `sigprocmask` for signal handling to control thread execution flow.
- Designed and managed a thread table for tracking up to 128 threads, each with individual state, stack, and execution context, enhancing the system's ability to handle concurrent operations.
- Developed semaphore mechanisms with initialization, wait, and post operations to enable inter-thread communication and synchronization, ensuring thread-safe interactions.
- Integrated system calls and assembly code for low-level operations including thread-specific data handling and pointer mangling for security purposes.
- Utilized UNIX signals to implement pre-emptive scheduling based on timer interrupts, ensuring fair CPU time allocation among threads.
- Implemented a robust thread scheduling algorithm that manages thread states (ready, running, blocked, exited) to facilitate smooth context switches.
- Ensured thread safety and synchronization using mutex locks and signal masks to prevent race conditions and maintain data integrity across concurrent executions.
- Managed memory for thread stacks dynamically, allowing threads to have isolated execution stacks for safety and stability.

Audio Style Transfer via Neural Network

- Developed audio processing pipelines using Python, including advanced audio manipulation and transformation techniques.
- Integrated and utilized several audio processing libraries such as Librosa, WebRTC VAD, and Pydub for tasks including waveform normalization, voice activity detection, and audio I/O operations.
- Employed machine learning models for audio synthesis, specifically using the WaveNet vocoder, to convert mel spectrograms back to audio, demonstrating proficiency in both audio analysis and synthesis.
- Managed audio data preprocessing, including dynamic resampling and silence trimming, to prepare data for neural network training, optimizing for both performance and accuracy.
- Implemented a Convolutional Neural Network (CNN) setup with a VGG19 network for style transfer in audio spectrograms, bridging the gap between traditional image processing techniques and audio data.
- Optimized neural network parameters and training procedures to achieve high-quality audio output, demonstrating deep understanding of neural architecture and hyperparameter tuning.

Information Technology Specialist January 2024 – May 2024
Boston University

- Worked with a team to fix software bugs and technical issues within classrooms

Truss Analysis Algorithm

- Truss was designed using Onshape to ensure it fell within size specifications and then collaborated with 3 other group members on MATLAB to determine failing member, max load, and total cost of the truss

FPGA Tennis Game

- Using Verilog, constructed behavioral modules in Vivado for the functionality of the game
- Developed testbenches for each module to test checkpoints and individual parts of the program

Temperature Sensor

- Designed Arduino code to send information from the switch to the LCD screen and buzzer
- Soldered wires and developed a circuit design for wires attached to Arduino, battery and LCD to display temperature

Utilized Onshape to format box and battery holder