MiniOS Developer Manual

If you so choose, you can develop programs for MiniOS. This document attempts to offer some guidance for the programmer by providing build instructions and an overview of available syscalls.

BUILDING

MiniOS must be built with a cross compiler; this can be done in two ways. The user can either rebuild GCC for the i386 system personally or use a pre-built cross compiler. Detailed instructions for setting up the cross compiler are in the `/doc` directory. Follow `INSTALL.md` for a manual GCC setup, or `INSTALL_PREBUILT.md` for a prebuilt cross compiler.

We've developed MiniOS as a multiboot kernel, so it must be launched by a bootloader that supports multiboot. A prebuilt ISO file is included with the project, which includes GRUB to boot the kernel; it can be run with a hardware emulator such as QEMU or on straight x86 hardware.

MiniOS is built with Makefiles, with each directory containing a Makefile specific to that portion of code.

PROGRAMMER SYSCALL INTERFACE

User programs can make syscalls to use kernel functions. These syscalls mainly focus around IPC communication and spawning processes. We point the programmer to `include/syscalls/syscalls.h` for the most up to date information. **NOTE:** MiniOS is filled with bugs; don't expect much of anything to work.

`uint32_t send(msg_t* msg, uint32_t comm_channel);`

`send` can send an arbitrary binary message to another process or to `STDOUT` to print to the screen. The `msg_t` structure includes a pointer to the `data`, a `length` field, and a `sender` field. To send a message, first create a `msg_t` structure and set its `data` pointer, `length` field, and `sender` field to the desired values, and pass a pointer to the `msg_t` structure to send. The `comm_channel` argument is used to specify which process to send the message to, or use the `STDOUT` macro to print to the screen as ASCII text. `send` returns 0 on success, 1 on failure.

`uint32_t recv(msg_t* msg_dest, uint32_t comm_channel);`

`recv` is the sister call of `send` with functionality mirroring it. To use `recv`, create a `msg_t` structure with the `data` pointer set to the base address of a buffer big enough to hold the incoming data, and the `length` set to the size of the buffer. The `sender` field will be populated by the kernel. To get input from the keyboard, pass the `STDIN` macro in the `comm_channel` argument. `recv` returns 0 on success, 1 on failure.

`uint32_t exit();`

`exit` can be called at any time in a user program's execution to immediately terminate the process and remove it from the process list. If the program's entry point returns, `exit` will be called automatically to remove it from the system. `exit` will return the PID of the process killed.

`uint32_t spawn(uint32_t eip, uint32_t argc, char** argv);`

`spawn` is used to start a new process. Simply pass the entry point of the new process in the `eip` argument and `argc/argv`, if desired. `spawn` will block the caller until the new process exits. `spawn` will return the PID of the process created.

`uint32_t spawn_bg(uint32_t eip, uint32_t argc, char** argv);`

`spawn_bg` has identical usage as spawn, the only difference being that the caller is not blocked when making a call to spawn_bg. `spawn_bg` will return the PID of the process created.

`uint32_t pid();`

`pid` will return the PID of the current running process.

ADDING PROGRAMS

As of now, loading programs from files is not supported, so programs must be compiled into the kernel. User programs should be placed in the `/src/program` directory, have their entry point included in `include/program/program.h`, and then added to the Makefile. A few example programs, including a shell, are already included. In order to run a program, use the `spawn` or `spawn_bg` syscall with the program's entry point as the `eip` argument.

Examples: `src/program/echo.c`, `src/program/overflow.c`, `src/program/shell.c`.

ACKNOWLEDGEMENTS

MiniOS was developed for Dr. Schrimpsher's CS499 class, which took place at UAH in Fall 2024. Developers include Ethan Jones, Britton Pearce, Logan Cagle, and Lane Wright.

MiniOS makes use of code from OSDev, as well as PDCLib (which is released under public domain).