uOS

Thomas M. Doylend

Last revision: Jan 30, 3014

uOS is a complete operating system for the Propeller chip. It provides a standardized way to access extension cards, extension RAM, pheripherals, and graphics devices.

Each application in uOS is provided with a set of 4-long areas, called *slots*. Each slot is used to interface with a specific portion of the OS.

The slots allocations are:
Slot 0: Process and utility manager (launches and kills apps and sets up utilities)
Slot 1: Pipe manager (handles inter-process connections)
Slot 2: Pheripheral manager (handles PS/2 keyboards and mice)
Slot 3: Lock manager (provides additional locks)
Slot 4 – 7: XT Card socket managers (extension card socket managers)
Slot 8: EEPROM manager (may not be present)
Slot 9: Flash manager (may not be present)
Slot 10: SD or microSD card manager (always present)
Slot 11: ROM manager (usually not present)
Slot 12 – 15: XRAM managers
Slot 16: Primary graphics feeder
Slot 17: Secondary graphics feeder

Each application obtains a set of slots to communicate with the OS, and an amount of program memory. All programs begin by running high-level assembly called uOS Assembly (or uASM) which is interpreted from the HUB.

When a uASM program is executed, these things occur:

1) 4KB of data specified by the launching application are loaded into the first 4KB of the new program's RAM space.
2) The first 8 longs of the program's RAM space are written with system data for the program to use.
3) A cog is allocated to an interpreter, with the PC set to 8 (so as not to execute the system data.)

All memory addresses in a uASM program are long addresses provided relative to the start of the program's RAM space. All memory addresses provided in slots are in the same format, so an address of $10 specifies the 17th long from the start of the program's memory (which from now on will be referred to as the PRAM.)

Let us study the functions provided by each slot.

Slot 0: The Process and Utility Manager

The process and utility manager takes a command in long 0, a parameter in long 1, and returns data to long 2 and 3. The commands are:

0: Idle.
1: Get the cog data for utility N. Returns the CPUID (ID of processor the process is executing on) << 16 | PID (ID of process within CPU.)
2: Launch the 4KB addressed by N to a new cog. Returns (CPUID << 16 | PID.)
3: The same as 2, but executes PASM code.
4: The same as 2, but replaces the current process.
5: The same as 3, but replaces the current process.
6: Kill the process specified in N (CPUID << 16 | PID.)
7: Set the current process as utility N. (See the section on Utilities for more info.)
8: Get an OS parameter N to long 2. (See the section on OS Parameters for more info.)

If a process could not be allocated or no utility is available for a certain function, the manager will return -1.

NOTE: When launching PASM, PAR contains a Propeller HUB byte pointer (not a uASM pointer) to the beginning of the application's PRAM

Slot 1: The Pipe Manager

While it is expected that in the future the pipe manager will provide full bidirectional pipes, it is at the moment merely a PEEK/POKE type interface. It takes a CPUID<<16|PID parameter in long 0, plus bit 31 indicates whether a write operation should occur. It takes a long PRAM address in long 1.

Then it reads a long from the specified process' PRAM at the specified address into long 2, and replaces it with long 3 if bit 31 of long 0 is set. Note that this operationg is NOT atomic; it is possible that the caller could read a long, the other application could write data into the slot, and the data could be overwritten. Care must be taken when designing to prevent this.

The pipe manager indicates inactivity by setting long 1 to -1.

Slot 2: The Pheripheral Manager

This manager is responsible for communicating with all pheripherals connected to a device. A 'pheripheral' in the instance means an extension device which is not connected to the device via an expansion slot. PS/2 keyboards, PS/2 mice, and RTCs fall under this category.

Long 0 selects the pheripheral to interface to. Not all pheripherals are available on all systems:

Pheripheral -1: Nothing. Used when the manager is idle.
Pheripheral 0: PS/2 Keyboard
Pheripheral 1: PS/2 Mouse
Pheripheral 2: RTC
Pheripheral 3: ADC

Long 1 is the 'function' long. With the keyboard, it performs this function:
If 0, next keypress is sent (0 if no keypress). Anything else tests whether that key is down.
Long 1 has no function for the mouse, which returns:
(delta_x) | (delta_y << 9) | (button_states << 18)

On an RTC, Long 1 controls which date/time component is returned:
0 – Microseconds
1 – Seconds
2 – Minutes
3 – Hours
4 – Day of week
5 – Day of month
6 – Month
7 – Year

On an ADC, long 0 controls which channel is returned.

Long 2 holds the return value.

Slot 3: Lock Manager

Long 0 provides a command. Bit 31 indicates whether the lock is to be obtained or released. Bit 30 is 0 when the manager is idle. Bit 29 MUST be set to 0. Bits 23-0 select a lock.
Long 1 returns -1 on success.

The locks are:
0 – 3: XT channels 0 – 3
4 – 7: Storage Devices 0 – 3 (EEPROM, Flash, microSD, ROM)
8 – 11: XRAM devices
12 & 13: Graphics devices 0 and 1
256 – 511: XT 0 devices
512 – 767: XT 1 devices
768 – 1023: XT 2 devices
1024 – 1279: XT 3 devices

When a process is killed, all its locks are automatically released.

Slots 4 – 7: XT Managers

A single device can provide up to four XT channels. This is to facilitate speed, as only one application at a time can access a channel. For example, if a sound card and a printer are connected to Channel 0, and App 0 is printing, App 1 will have to wait to use the sound card. If the printer and the sound card are kept on separate channels, both devices can be used simultaneously.

Long 0 selects a device.
Long 1 points to a block of data to send
Long 2 indicates the size of the block, in bytes.
Long 3 is written with -1 when operation is complete.

After each long, long 2 is decremented by 4, and long 1 is incremented by 1.

Note that the bytes in a long are shifted out left, or least significant byte first. Each byte is replaced by the response byte from the XT device.

Slots 8 – 11: Storage Device Managers

Storage Device managers (SDMs) device all storage devices into 4KB sectors. (See 'PFS(3) .pdf ' for an explanation of why this is.)

Long 0 provides a command:
0 – Idle.
1 – Read 4KB.
2 – Write 4KB.

Long 1 provides the address of a sector on the storage device. For example, address 6 writes the area from 0x6000 to 0x6FFF.

Long 2 provides a pointer to a block of PRAM.

NOTE: SDM #2 (The SD card manager) may not directly access the data on the card. The data may be in a pseudo-drive on a differently formatted card (for example, a 256MB drive file 'drive.bin' on a FAT16 card.)

Slots 12 – 15: XRAM Managers

A single processor can have up to 4 XRAM (external RAMs) connected to it. Each RAM can be interfaced separately, providing good performance on multi-RAM systems (note that the C3 has only one XRAM manager because due to its design, only one RAM can be interfaced at a time anyway.)

Long 0 provides a command

0 – Idle.
2 – Read 1 long
3 – Read 4KB
4 – Write 1 long
5 – Write 4KB
6 – Allocate a 4KB sector of RAM for the program.
7 – Deallocate a 4KB sector.

Long 1 provides an address in XRAM, except in the case of command 6, where it stores the return address.

Long 2 provides an address to a long or block of PRAM.

When a process is killed, all of its XRAM is automatically released.

Slots 16 and 17: Graphics Feeders

The graphics feeders are identical in function to the XT feeders, except that long 0 is unused.

The System Data

When a program begins executing, longs 0 – 7 contain the following data:

Long 0: Storage Device on which program can be found.
Long 1: Pointer to sector on storage device containing the first 4KB of the program.
Long 2: Pointer to the program's parent directory's first sector.
Long 3: Pointer to ASMDAT
Long 4: Amount of PRAM (in longs)
Long 5: CPUID << 16 | PID of current process

Longs 6 & 7 are currently unused.

The pointer to ASMDAT is NOT a long pointer offset by the PRAM address. It is a Propeller RAM byte pointer.

ASMDAT
ASMDAT is a section of data for PASM programs. All pointers in ASMDAT are Propeller RAM byte pointers. Currently it only contains one long:

Long 0: Pointer to the applications slots.

OS Parameters

OS parameters are a set of longs that may be useful for certain programs. They can be accessed with the Process Manager and they are:

Long 0: Machine type
Long 1: Machine version (8.8.16)
Long 2: OS version (8.8.16)
Long 3: Number of XRAM banks on this processor
Long 4: Flags
Long 5: Graphics Device 0 Type
Long 6: Graphics Device 1 Type

The flags are:
bit 0: Is the SD card sharing an SPI bus with the RAM?
bit 1: Are there two graphics devices?
bit 2: Is a VGA signal generated by this processor?
bit 3: reserved
bit 4: reserved
bit 5: Does this build/machine support PASM (no for PC Emulations.)
bit 6: Is application pre-emption enabled?

Graphics Device Types

The available types of graphics devices are:

0 – Nothing.
1 – Parallax Serial Terminal
2 – Extended Serial Terminal (see the EST documentation for details)
3 – Serial LCD
4 – Graphics/Text Multi-mode device
5 – Pure Graphics Device
6 – Hybrid Text/Graphics device

The interface standards for devices 4-6 are currently undefined.

Utilities

Utilities are small programs that perform specific tasks. There are four possible utilities:

Utility 0: Multiple filesystem manager
Utility 1: Universal pheripheral/XT manager
Utility 2: Windowing/desktop manager
Utility 3: reserved

Interface standards for utilities are currently undefined. Feel free to post any interface standard ideas at the forum! (Note that utilities can only be interfaced with by using the Pipe Manager.)

uOS Assembly

uOS Assembly is the assembly language interpreted from the first 4KB of a program when it is launched from the shell.

A command in uOS Assembly takes the following form:

bit 31: Source literal flag
bits 30-24: Instruction
bits 23-12: Destination
bits 11-0: Source

Like PASM, the destination and source are LONG pointers. Since each is twelve bits in size, up to 16KB (4096 longs * 4 bytes per long) can be addressed at a time.

IMPORTANT: Nothing in uOS is error-checked, and this applies to uASM addresses as well! If a program is provided with 8KB of PRAM and still tries to access long $FFF, the interpreter will silently overwrite whatever is stored there – system data, another application, ANYTHING!

All arithmetic instructions are signed unless otherwise noted.

The instructions are:
```
$00: End self
$01: MOV
$02: ADD
$03: SUB
$04: reserved for MUL
$05: reserved for MULX
$06: reserved for DIV
$07: reserved for MOD
$08: ROL
$09: ROR
$0A: SHL
$0B: SHR
$0C: AND
$0D: OR
$0E: XOR
$0F: NOT
$10: NEG
$11: ABS
$12: RCL
```

```
$13: reserved for ONES
$14: REV
$15: ANDN
$16: reserved for SET_BIT
$17: reserved for CLR_BIT
$18: IF_EQ (If D != S, skip next instruction.)
$19: IF_NE (If D == S, skip next instruction.)
$1A: IF_GT (If D <= S, skip next instruction.)
$1B: IF_LT (If D >= S, skip next instruction.)
$1C: reserved for IF_GE
$1D: reserved for IF_LE
$1E: reserved for IF_BT
$1F: reserved for IF_NB
$20: JMP (Jump to S.)
$21: JMPRET (Jump to S, placing PC+1 into D.)
$22: WSLOT (Write D into long S[1:0] of slot (S>>2).)
$23: RSLOT (Read  D from long S[1:0] of slot (S>>2).)
$24: W_PTR (Write D into address pointed to by S.)
$25: R_PTR (Read  D from address pointed to by S.)
```

Please enjoy uOS! And if you create something cool, don’t hesitate to post to the forum!