BORSAT OS System Calls

Version: 3
Created: Dec 25, 2024

Table of Contents

Intended Audience

It is assumed the person reading this knows Intel Assembly.

Background

BORSAT OS uses one system call (i.e. interrupt call) for all user functions; 0xFF (e.g. int 0xff). The sections below gives a brief description of the intent of the system function and required parameters. Each system function is identified using register RDX.

The system functions described below use the following format:

Description of function

Parameters: *<registers used to call the function. Note, RDX is always required>*
Returns: *<return values and which registers the values are returned to>*

INT 0xFF
Parameters: RDX = function number
RAX – R15 = parameters for specific function

System Functions

Stop the running program

Parameters: RDX = 0x0
Returns: n/a

Return interrupt ticker count

Parameters: RDX = 0x1
Returns: RAX = current ticker count

Become active STDIN process for the keyboard

Used for taking keyboard control
Parameters: RDX = 0x3
Returns: n/a

Get active STDIN process for the keyboard

Used to determine which program has control of the keyboard.
Parameters: RDX = 0x4
Returns: n/a

Release STDIN for the keyboard

Used for giving back control of the keyboard to the OS.
Parameters: RDX = 0x5

Returns:	n/a

Hide Cursor

Parameters:	RDX = 0x6
Returns:	n/a

Show Cursor

Parameters:	RDX = 0x7
Returns:	n/a

Reset cursor to start position

Parameters:	RDX = 0x8
Returns:	n/a

Returns the process ID (PID) of the running program

Parameters:	RDX = 0x9
Returns:	EAX = PID

Network transmit

System call for transmitting a frame. Low level function, use at your own risk. Tells the OS to send whatever is in the transmit system memory buffer.

Parameters:	RDX = 0xA
	RSI = NIC memory location
	RAX = memory location of frame
	RCX = size of frame
Returns:	<nothing>

Network receive

System call for receiving a frame. Low level function, use at your own risk. Tells the OS to poll the Receive Descriptor and Receive system memory buffer to network frames.

Parameters:	RDX = 0xB
	RSI = NIC memory location
	RDI = Receive descriptor memory location
	RAX = memory location of frame
Returns:	RCX = size of frame

Current time in millisecond and fractions of a millisecond

Use this system call for precise time keeping.

Parameters:	RDX = 0xC
Returns:	RAX = milliseconds
	RBX = fractions of a millisecond

Current time in milliseconds only

Parameters:	RDX = 0xD
Returns:	RAX = milliseconds

Clear screen

Clears the screen.

Parameters:	RDX = 0xE
Returns:	<nothing>

Switch to another process

Use this to save CPU time. Rather than waiting for another process to do something, force a process switch from the current process so that the other processes run.

Parameters:	RDX = 0xF
Returns:	<nothing>

Deallocates memory from a process

This will give memory back to the system, to be reused again.

Parameters:	RDX = 0x10
	RAX = process' virtual memory location
	RCX = size (in bytes) to deallocate
Returns:	<nothing>

Generate a random number

Parameters: RDX = 0x11

CL = size of random number to generate

- 1 = generate a number between 0 - 15
- 2 = generate a number between 0 - 255
- 3 = generate a number between 0 - 65535
- 4 = generate a number between 0 - 16,772,215
- 5 = generate a number between 0 - 4,294,967,295
- 6 = generate a number between 0 - 18,446,744,073,709,551,615

Returns: RCX = random number

Return current time in seconds

Parameters:	RDX = 0x1D
Returns:	RAX = current time in seconds

Prints to X/Y coordinates

Text mode or GUI mode print to X,Y location. X will use character width * X. And Y will use character height * Y.

NOTE: This routine will not clear the screen.

Parameters:	RDX = 0x400
	AL = X
	AH = Y

	RSI = pointer to string location
Returns:	<nothing>

Prints to X/Y coordinates, at a specified width and specified number of characters.

Parameters:	RDX = 0x401
	AL = X
	AH = Y
	BX = width
	CX = number of characters to print
	RSI = pointer to string location
Returns:	<nothing>

Return Process ID number for a given program name string

Parameters:	RDX = 0x403
Parameters:	RSI = pointer to program name string location (virtual address)
Returns:	EBX = process ID number (PID). 0xFFFFF (5 f's) is a not found error.

Send an IPC message to a specified process

Parameters:	RDX = 0x404
	RSI = pointer to message to send
	CX = size of message to send (min 32)
	BX = PID to send message to
Returns:	BX = Return code;

0 = success
1 = PID not found
2 = Destination PID's header does not have enough space

Share memory with another process

Parameters:	RDX = 0x405
	RSI = pointer to memory to share
	CX = size of memory to share (in 4K blocks)
	BX = PID to share memory with
Returns:	BX = Return code;

0 = success
1 = PID not found

Read currently running IPC message header

Reads the IPC message header for a currently running process. Then returns the first 32 bytes. This system call would be used if you need to read a message while the running process is in an interrupt and cannot easily read the IPC memory.

Parameters:	RDX = 0x406
Returns:	RAX = Returns message (bytes 0 – 7)
	RBX = Returns message (bytes 8 – 15)
	RCX = Returns message (bytes 16 – 23)
	RDX = Returns message (bytes 24 - 32)

Read currently running IPC message header – brief

Reads the IPC message header for a currently running process. However, only returns the first 8 bytes. This system call would be used if you need to read a message while the running process is in an interrupt and cannot easily read the IPC memory.

Parameters:	RDX = 0x407
Returns:	RAX = Returns message (bytes 0 – 7)

Read currently running IPC message header and return data

Reads the IPC message header for a currently running process. Then skips the CMD field and returns the first 32 bytes of data. This system call would be used if you need to read a message while the running process is in an interrupt and cannot easily read the IPC memory.

Parameters:	RDX = 0x408
Returns:	RAX = Returns message (bytes 0 – 7) of data
	RBX = Returns message (bytes 8 – 15) of data
	RCX = Returns message (bytes 16 – 23) of data
	RDX = Returns message (bytes 24 - 32) of data

Clears the IPC header for the currently running process

Parameters:	RDX = 0x409
Returns:	<nothing>

Check for queued IPC message

Checks if the currently running process has an IPC message queued. If so, populates the process' IPC header with the message.

Parameters:	RDX = 0x40A
Returns:	BX = Return code;
	0 = no messages found
	1 = queued message found

Check for pending sent IPC message(s)

Checks if the currently running process has IPC message(s) queued to a specified process. If so, returns the number of messages queued.

Parameters:	RDX = 0x40B

Returns: BX = Return code;
0 = no messages found
1 = number of messages found

Print

Prints a string to the screen. This will do a raw print and will not do any type of word wrapping. NOTE: This routine will not clear the screen.

Parameters: RDX = 0x40E
RSI = pointer to string location
Returns: <nothing>

Release or deallocate shared memory

Releases or deallocates shared memory from a currently running process. Depending on conditions, memory is released, or, deallocated and given back to the OS.

Parameters: RDX = 0x422
RAX = virtual memory that is shared
CX = size of memory shared
Returns: BX = Return code;
0 = success
1 = error

Release and deallocate shared memory

Releases and deallocates shared memory from a currently running process. NOTE: This call will not check if there is another program using the memory. This routine is meant for use cases that memory should be given back to the OS (e.g. TCP connection closes, give the OS back the receive buffer).

Parameters: RDX = 0x423
RAX = virtual memory that is shared
CX = size of memory shared
Returns: BX = Return code;
0 = success
1 = error

Sound Blaster 16 – Memory Allocation

Gets a request from a user program to allocate memory for the SB16 card. Because memory for SB16 DMA has specific things, I have created an IRQ system call and routine to do it.

Parameters: RDX = 0x424
RCX = Size (bytes) to allocate. If zero use maximum (65535), if more then maximum, use maximum.

Returns:	RAX = Memory address. Zero if none available.
	RCX = Size (bytes) of memory allocated

Register Interrupt Handler

Connects a function to an interrupt. Also known as an Interrupt Service Routine (ISR) handler.
NOTE: Use with extreme caution!!

Parameters:	RDX = 0x425
	AL = IRQ number
	AH = ISR flags, 0 for default values
	RBX = memory location of ISR routine that will run when IRQ fires
Returns:	N/A

File System Functions

Start a file in storage (HDD)

Start a new file. This WILL overwrite an existing file.

Parameters:	RDX = 0x500
	RSI = Memory pointer to filename string.
Returns:	RAX = cluster number (if FAT32)
	BL = return code
	0: success
	-1: Error, finding file
	-2: Error, ran out of memory
	-3: Error, creating directory
	-4: Error, no space for new file
	-5: Error with storage
	-6: Error deleting file

Append a file in storage (HDD)

Append data to an existing file.

Parameters:	RDX = 0x501
	RAX = memory pointer to location of data
	RCX = number of bytes to append
	RSI = Memory pointer to filename string.
Returns:	n/a

File Exists

Check if a file or directory already exists.

NOTE: Use DL to determine if a file is found or not. There is a case that a cluster number can be zero (when the parent directory is a root directory, 0 will be returned).

Parameters:	RDX = 0x504
	RSI = Pointer to NULL terminated filename.
Returns:	RAX = cluster number.
	DL = 0 nothing found
	1 found a file
	2 found a directory
	90 ERROR - no memory available

Check for, and return a queued message

Checks the Send Message queue to see if the currently running process has a message to be sent to it. If so, then the message will be copied to the currently running process' IPC header.

Parameters:	RDX = 0x40A
Returns:	BL = Return code;
	0 = no messages waiting
	1 = found a message (IPC header populated).

Check for pending sent messages

Checks if the currently running process has any messages still waiting to be sent to another process.

Parameters:	RDX = 0x40B
	BX = Process ID (PID) sending the message(s) to.
Returns:	BL = Return code;
	0 = no messages waiting
	1 = message(s) waiting in the queue

Network Functions

DHCP request

Run a DHCP request for a specified NIC.

Parameters:	RDX = 0x12
	RSI = NIC memory address
Returns:	n/a

DHCP release

Release DHCP address for a specified NIC.

Parameters:	RDX = 0x13
	RSI = NIC memory address
Returns:	n/a

DHCP renew

Renews a DHCP address for a specified NIC.

Parameters:	RDX = 0x14
	RSI = NIC memory address
Returns:	n/a

Get network return code

Get a response code from the Network Module.

Parameters:	RDX = 0x16
	RCX = network transaction ID
Returns:	RAX = return code

0 = transaction ID matches (good time to get the values in function 0x17).
-1 = no match (should keep waiting or time out).

Get network values

Get values from the Network Module.

Parameters:	RDX = 0x17
	RCX = network transaction ID
Returns:	R8 = Response field 1 (8 bytes)
	R9 = Response field 2 (8 bytes)
	R10 = Response field 3 (up to 1512 bytes)

Get IP addresses for all NICs

Returns the IP addresses and network masks for all of the NICs. If a NIC does not have an IP, then it will return zero. The high DWORD is the mask, the low DWORD is the IP address.

Parameters:	RDX = 0x1A
Returns:	RAX = NIC1 mask / IP address
	RBX = NIC2 mask / IP address
	RCX = NIC3 mask / IP address
	RDX = NIC4 mask / IP address

Get IP addresses for a specific NIC

Returns the IP address and network mask for one of the NICs. If the NIC does not have an IP, then it will return zero. The high DWORD is the mask, the low DWORD is the IP address.

Parameters:	RDX = 0x1B
	CL = NIC number <1 to 4>, returns zero if invalid entry
Returns:	EAX = mask / IP address

Send a raw packet

Parameters:	RDX = 0x20
	RSI = NIC memory location
	RDI = memory location of packet
	Packet information / payload structure is as follows:
	DWORD - Dst IP
	DWORD - Src Port / Dst Port
	DWORD - Protocol number (TCP, UDP, etc) / payload size (bytes)
	BYTES - payload
Returns:	<nothing>

Request to open a TCP connection

Parameters:	RDX = 0x25
	RAX = Destination port (2 bytes, high DWORD) / destination IP address (4 bytes, low DWORD).
	ECX = Size of receive buffer. Maximum size is 1,073,725,440 (0x3FFFC000) bytes. Should be an even number. Otherwise will be rounded down to the next even number (eg. 65535 will become 65534).
	RDI = Receive buffer location.
Returns:	RAX = return code; 0=success, -1 error starting request, refer to Return Codes for more information.
	RCX = connection ID

Close a TCP connection

Parameters:	RDX = 0x27	
	RCX = connection ID	
Returns:	RAX =	return code; 0x3FF=close completed,
		-1 could not send close to network module

Send data over an existing TCP connection

Parameters:	RDX = 0x28	
	EAX = Send buffer size.	
	RCX = connection ID	
	RSI = virtual address of send buffer	
Returns:	RAX =	return code; network response code,
		-1 could not send close to network module

Signal OS that the receive buffer can be cleared and used again.

Parameters:	RDX = 0x29	
	RCX = connection ID	
Returns:	RAX =	return code; 0 = completed,
		-1 could not send to network module

Send data over an existing TCP connection using the send queue

Parameters:	RDX = 0x2B	
	EAX = Send buffer size.	
	RCX = connection ID	
	RSI = virtual address of send buffer	
Returns:	RAX =	return code; network response code,
		-1 could not send close to network module

Opens a TCP listener port.

Parameters:	RDX = 0x30
	AX = Port
	ECX = size of receive buffer
	RDI = receive buffer location
Returns:	RAX = return code; 0=success, refer to NET_RTN_ codes
	RCX = connection ID

DNS Query.

Parameters: RDX = 0x40

AX = Type number, valid numbers are:

A	equ 1	; IPv4 address
NS	equ 2	; name server lookup
CNAME	equ 5	; alias name
PTR	equ 12	; reverse record lookup
MX	equ 15	; mail exchange
AAAA	equ 28	; IPv6 address

RSI = address pointer to query string

Returns: RAX = return code; 0=success, refer to NET_RTN_ codes

RSI = using the same address space as the query string, returns the results

List DNS Servers configured for a NIC.

Parameters: RDX = 0x41

CL = NIC number; valid values are 1 to 4

Returns: RAX = DNS server 1; 32bit IP address, zero if not configured

RBX = DNS server 2; 32bit IP address, zero if not configured

RCX = DNS server 3; 32bit IP address, zero if not configured

GUI Functions

Get linear address

Parameters:	RDX = 0x100
	AX = X
	BX = Y
Returns:	RDI = linear address

Get character size

Parameters:	RDX = 0x101
	R10 = character code (aka ASCII value)
Returns:	AX = width in pixels
	BX = height in pixels
	If character code not found, then AX & BX will be zero

Get Y pitch

Parameters:	RDX = 0x102
Returns:	AX = pitch value

Get number of bytes per pixel

Parameters:	RDX = 0x103
Returns:	AX = bytes per pixel

Get the X resolution of the screen

Parameters:	RDX = 0x104
Returns:	AX = number of pixels

Get the Y resolution of the screen

Parameters:	RDX = 0x105
Returns:	AX = number of pixels

Get number of bits per pixel

Parameters:	RDX = 0x106
Returns:	AX = bits

Get GUI ON

Parameters:	RDX = 0x10D
Returns:	AL = mode; 0=text mode, 1=graphics mode

Clears GUI input box

Parameters:	RDX = 0x10E
Returns:	<nothing>

Draw Functions

Draw a character

Parameters:	RDX = 0x120
	AX = X coordinate
	BX = Y coordinate
	ECX = color
	R10 = character code
Returns:	<nothing>

Draws a NULL terminated string

Parameters:	RDX = 0x121
	AX = X coordinate
	BX = Y coordinate
	ECX = color
	RSI = memory location of NULL terminated string
Returns:	<nothing>

Draws a full box

Parameters:	RDX = 0x126
	R8 = starting X (left)
	R9 = starting Y (top)
	R10 = width
	R11 = height
	ECX = color
Returns:	<nothing>

'Types' (i.e. cat) a large amount of data to the MAINSCR

Parameters:	RDX = 0x129
	RCX = number of bytes to display
	RSI = memory location of data
Returns:	<nothing>

Process Functions

Get Process ID

Get the process ID of the currently running program. Used to find out the calling program’s ID.

Parameters:	RDX = 0x200
Returns:	AX = process ID

Run program from a user program

Wrapper routine to run a process from a user mode process. This allows the OS to put in checks and security so that a user process does not go out of control.

Parameters:	RDX = 0x220
	RAX = virtual address of memory pointer to program name
	BL = number of parameters to pass; maximum 3
	R8 = parameter 1
	R9 = parameter 2
	R10 = parameter 3
	R14 = process' page directory address
Returns:	AL = return status; 0=success, otherwise error code
	0=success
	1=file not found
	2=no memory available
	5=invalid parameter stack pointer
	RBX = memory location of process
	RCX = size of process, in bytes
	DX = process number Parameters:

Allocate memory for a user program

Parameters:	RDX = 0x221
	RAX = number of bytes to allocate
Returns:	RAX = virtual address where memory begins, 0 if error
	BL = return code; 0=success, 1=not enough memory

Deallocate memory from a user program and gives it back to the OS to be used again.

Parameters:	RDX = 0x222
	RAX = start of virtual memory location
	RCX = number of bytes to deallocate
Returns:	n/a

Releases memory from a user program.

Releases memory from a user program but does not release memory back to the OS. Typical use case for this is when memory is shared among 2 or more user programs. One user program must own the responsibility of giving the memory back to the OS.

Parameters:	RDX = 0x223
	RAX = start of virtual memory location
	RCX = number of bytes to release
Returns:	n/a