SYSCALLS.3

PSIONICS FILE - SYSCALLS.3

==========

System calls (part 3)

Last modified 1998-07-22

==========

See part 1 for general notes and explanations.

Fn $8D is used for the Window Server, and is described in the Psionics file WSERVER.

Fn $8E Sub $00 to $10 control various parts of the hardware, and should notbe used by OPL programs.

Fn $8E Sub $11

HwGetSupplyStatus

BX: 6 byte bufferGets information about the power supply. The buffer is filled with the following data:

Offset 0 (word): main battery voltage in mV

Offset 2 (word): backup battery voltage in mV Offset 4 (word): positive if external power is available, zero if not available, and negative if the detector is disabled

because an SSD door is open.The returned voltages should be taken with a grain of salt: I have observed

the following values:

1800: no main batteries

1800: low main batteries

2056: new main batteries

2300: no lithium battery

2556: lithium battery fitted

Fn $8E Sub $12

HwLcdContrastDelta

AL: step directionAlters the LCD contrast by one step, upwards if AL is between 0 and 127, and

downwards if it is between 128 and 255 (all inclusive).

Fn $8E Sub $13

HwReadLcdContrast

AL: -> settingGets the current LCD contrast setting. On a Series 3, only the bottom 4 bits are significant.

Fn $8E Sub $14

HwSwitchOff

CX: delay in quarter secondsSwitches the machine off for the specified time, then back on again ($FFFF

means never turn back on). The machine will also be turned on by externalevents, alarms, and timers. The delay must be at least 9 (2.25 seconds).

Fn $8E Sub $15 should only be used by device drivers.

Fn $8E Sub $16

HwExit

Exits the emulation on PC systems; has no effect on actual Psion machines.

Fn $8E Sub $17 to $1A should only be used by device drivers.

Fn $8E Sub $1B

HwGetPsuType

AL: -> PSU type

Gets the PSU type: 0 = old MC, 1 = MC "Maxim", 2 = Series 3t, 3 = Series 3a

Fn $8E Sub $1C

HwSupplyWarnings

BX: 8 byte bufferGets information about the power supply. The buffer is filled with the following data:

Offset 0 (word): main battery good voltage threshold in mV Offset 2 (word): backup battery good voltage threshold in mV

Offset 4 (word): main battery nominal maximum voltage in mV Offset 6 (word): backup battery nominal maximum voltage in mV

The values will depend on the battery type set by GenSetBatteryType.

Fn $8E Sub $1D

HwForceSupplyReading

@No documentation available at present@

Fn $8E Sub $1E

HwGetBackLight

AX: -> current valueOn systems fitted with a backlight, this value indicates control of the backlight function. If the top bit is set, the operating system will ignorethe backlight toggle key. The remaining bits give the auto-light-off time inticks (1/32 second); zero means that auto-light-off is disabled.

Fn $8E Sub $1F

HwSetBackLight

BX: new value

Sets the backlight control value (see HwGetBackLight).

Fn $8E Sub $20

HwBackLight fails

AL: action -> status before callThis call fails if no backlight is fitted. Otherwise, the actions are:

0 = switch backlight off

1 = switch backlight on

2 = toggle backlight

3 = no action

The status returned is that of the backlight before the call: 0 = off, 1 = on.

Fn $8E Sub $21 controls various parts of the hardware, and should not be used by OPL programs.

Fn $8E Sub $22

HwSupplyInfo v3

BX: 22 byte buffer

Fills the buffer with additional information about the power supply: Offset 0 (byte): main battery status:

0 = not present

1 = very low voltage

2 = low voltage

3 = good voltage

Offset 1 (byte): worst main battery status since batteries last inserted Offset 2 (byte): non-zero if backup battery voltage is good Offset 3 (byte): non-zero if external supply is present

Offset 4 (word): warning flags

Bit 0: set if power supply too low for sound Bit 1: set if power supply too low to use flash

Bit 2: set if offset 6 changed because system clock changed clear if offset 6 changed because the batteries were changed

Offset 6 (long): abstime when batteries inserted

Offset 10 (long): ticks running on battery Offset 14 (long): ticks running on external supply

Offset 18 (long): mA-ticks (i.e. mAhours * 60 * 60 * 32)

Fn $8E Sub $28

HwGetScanCodes v3

BX: 20 byte bufferThe buffer is filled with information describing the state of each key on the keyboard. For the Series 3a, the offset and bit for each key are givenin the following table ("2:4" means offset 2, bit 4).

System	9:1	Esc	15:0	Delete	4:0	1	14:1	A	12:2	N	0:6								
Data	7:1	Tab	0:2	Enter	0:0	2	14:2	B	8:6	O	4:5	Word	11:1	Control	4:7	ShiftR	6:7	3	10:6
C	10:3	P	2:5	Agenda	3:1	ShiftL	2:7	Up	14:5	4	8:2	D	10:4	Q	12:1				
Time	1:1	Psion	0:7	Left	0:4	5	8:3	E	10:5	R	8:1	World	5:1	Menu	10:7	Down	0:5	6	14:3
F	10:1	S	12:4	Calc	3:0	Diamond	8:7	Right	0:1	7	6:6	G	8:5	T	8:4				
Sheet	1:0	Space	8:0	Help	6:2	8	4:3	H	14:6	U	6:5	+ and =	2:3	- and _	2:2	9	4:4	I	4:2
V	10:2	* and :	2:6	/ and ;	2:1	0	2:4	J	6:4	W	12:5	, and <	6:1	. and >	14:4	K	4:1	X	12:6
L	4:6	Y	0:3	M	6:3	Z	12:3												

Fn $8E Sub $29

HwGetSsdData

Nothing is known about this call except its name. @@@@@@

Fn $8E Sub $2A

HwResetBatteryStatus v3.9

The battery status is reset, exactly as if the main batteries had been removed and then replaced.

Fn $8E Sub $2B

HwEnableAutoBatReset v3.9

AX: -> (if querying) 1 = enabled, 0 = disabled BX: 1 = enable, 0 = disable, -1 = query

Enables, disables, or queries the current setting of the automatic resetof the battery status. This is intended for use on the Workabout, where the batteries can be recharged in situ.

Fn $8E Sub $2C

HwGetBatData v3.9

AX: -> address of a data structure in kernel data spaceThe data is described in the KERNEL file.

Fn $8E Sub $2E

HwReLogPacks v3.9 fails

This has the same effect as opening and then closing the SSD doors.

Fn $8E Sub $2F

```
HwSetIRPowerLevel v3.9
    AX: -> previous level (0 = low, 1 = high)
    BX: new level (0 = low, 1 = high)
Sets the IR power level, returning the previous level.
Fn $8E Sub $30
HwReturnTickCount v3.9
    AX: tick count
Returns a tick count that is incremented 32 times per second.
Fn $8E Sub $31
HwReturnExpansionPortState v3.9
    AX: -> port type and state
    BX: -> zero
Returns the type and state of the expansion port:  Bits 0 to 7:  non-zero if SSD doors are open,
or something has just been                plugged into or removed from the Honda expansion
connector
  Bits 8 to 14: connector type: 0 = Series 3t/3a, 1 = Workabout LIF,                2 = Siena
Honda, 3 = Series 3c Honda, 4 = HC
  Bit 15:       1 = contains Condor chip, 0 = no Condor chip
Fn $8F
GenDataSegment
This call is only useful in assembler code; it sets ES to point to the startof the kernel data
space (accessible in OPL using GenGetOsData).
Fn $90
ProcPanic
    AL: reason
Panic the current process; this call never returns.
Fn $91
ProcCopyFromById fails
    BX: process ID
    CX: number of bytes to copy
    SI: remote address of first byte to copy
   eDI: local address of first byte to copyCopy a number of bytes from the indicated process to
the current process.
Fn $92
ProcCopyToById fails
    BX: process ID
    CX: number of bytes to copy
```

```
    SI: local address of first byte to copy
    DI: remote address of first byte to copyCopies a number of bytes from the current process to
the indicated process.
Fn $93
CharIsDigit
Fn $94
CharIsHexDigit
Fn $95
CharIsPrintable
Fn $96
CharIsAlphabetic
Fn $97
CharIsAlphaNumeric
Fn $98
CharIsUpperCase
Fn $99
CharIsLowerCase
Fn $9A
CharIsSpace
Fn $9B
CharIsPunctuation
Fn $9C
CharIsGraphic
Fn $9D
CharIsControl
    AL: character to test
    EQ: -> set if test fails, clear if test succeedsTests to see whether the character has the
indicated property. These functions
are language dependent.
Fn $9E
CharToUpperChar
Fn $9F
CharToLowerChar
Fn $A0
CharToFoldedChar
    AL: character 1 -> converted character 1
    AH: character 2 -> converted character 2Converts two characters to uppercase, lowercase, or
```

folded (uppercase with no
accents). These functions are language dependent.
Fn $A1
BufferCopy
    CX: length to be copied
    SI: address of "from" buffer
   eDI: address of "to" bufferCopies a number of bytes from one buffer to another. The case of the buffers
overlapping is handled correctly.
Fn $A2
BufferSwap
    CX: length to be swapped
    SI: address of buffer 1
   eDI: address of buffer 2Swaps the contents of two buffers. The case of the buffers overlapping is
handled correctly.
Fn $A3
BufferCompare
Fn $A4
BufferCompareFolded
    BX: length of buffer 2
    CX: length of buffer 1
    SI: address of buffer 1
   eDI: address of buffer 2
    UL: -> set if buffer 1 less than buffer 2
    EQ: -> set if buffers are identicalThe contents of the two buffers are compared byte-for-byte, using unsignedcomparisons, and the result flags set accordingly. BufferCompareFolded acts as if each character had been passed to CharToFoldedChar before comparison.
Fn $A5
BufferMatch fails
Fn $A6
BufferMatchFolded fails
    CX: length of buffer
    DX: length of pattern
    SI: address of buffer
   eDI: address of patternThe buffer is examined to determine whether it matches the pattern (using

the usual wildcards); the call fails if it does not. BufferMatchFolded actsas if each character had been passed to CharToFoldedChar before comparison.

Fn $A7

BufferLocate fails

Fn $A8

BufferLocateFolded fails

    AH: character -> [undefined]

    AX: -> offset of character

    CX: length of buffer

    SI: address of bufferThe buffer is searched to determine if the character occurs within it; the

call fails if it does not. If it does, the offset from the start of the bufferof the first occurence is returned. BufferLocateFolded acts as if eachcharacter had been passed to CharToFoldedChar before comparison.

Fn $A9

BufferSubBuffer fails

Fn $AA

BufferSubBufferFolded fails

    AX: -> offset

    BX: length of buffer 2

    CX: length of buffer 1

    SI: address of buffer 1

   eDI: address of buffer 2Buffer 1 is searched to determine if buffer 2 occurs within it; the call failsif it does not. If it does, the offset from the start of buffer 1 of the first occurence of buffer 2 is returned. BufferSubBufferFolded acts as if eachcharacter had been passed to CharToFoldedChar before comparison.

Fn $AB

BufferJustify

    AX: -> address of first uncopied character

    BX: length of buffer 2

    CX: length of buffer 1

    DX: fill character * 256 + control code

    SI: address of buffer 1

   eDI: address of buffer 2Buffer 1 is copied into buffer 2, and the remaining space filled with the fill

character according to the control code (0 = fill at end, 1 = fill at start,2 = centre the copied data). If buffer 1 is longer than buffer 2, then thecontents will be truncated to fit. If

the length of buffer 2 is negative, thenit is assumed to be the same as that of buffer 1 (and the data is just copied).

Fn $AC

StringCopy

Fn $AD

StringCopyFolded

SI: cstr

eDI: large enough bufferThe cstr is copied into the buffer. StringCopyFolded passes each character

through CharToFoldedChar during the copy.

Fn $AE

StringConvertToFolded

SI: cstr

Each character of the cstr is passed through CharToFoldedChar.

Fn $AF

StringCompare

Fn $B0

StringCompareFolded

SI: cstr 1

eDI: cstr 2

UL: -> set if cstr 1 less than cstr 2

EQ: -> set if cstrs are identicalThe contents of the two cstrs are compared byte-for-byte, using unsigned

comparisons, and the result flags set accordingly. StringCompareFolded actsas if each character had been passed to CharToFoldedChar before comparison.

Fn $B1

StringMatch fails

Fn $B2

StringMatchFolded fails

SI: cstr to be searched

eDI: pattern (cstr)The cstr is searched to determine if the pattern occurs within it (using the usual wildcards); the call fails if it does not. StringMatchFolded actsas if each character had been passed to CharToFoldedChar before comparison.

Fn $B3

StringLocate fails

Fn $B4

StringLocateFolded fails

```
Fn $B5
StringLocateInReverse fails
Fn $B6
StringLocateInReverseFolded fails
    AH: character -> [undefined]
    AX: -> offset of character
    SI: cstr
The cstr is searched to determine if the character occurs within it; thecall fails if it does
not. If it does, the offset from the start of the cstrof the first (for StringLocate and
StringLocateFolded) or last (for the twoReverse calls) occurence is returned. The two Folded
calls act as if each
character had been passed to CharToFoldedChar before comparison.
Fn $B7
StringSubString fails
Fn $B8
StringSubStringFolded fails
    AX: -> offset
    SI: cstr 1
   eDI: cstr 2
Cstr 1 is searched to determine if cstr 2 occurs within it; the call failsif it does not. If it
does, the offset from the start of cstr 1 of the firstoccurence of cstr 2 is returned.
StringSubStringFolded acts as if eachcharacter had been passed to CharToFoldedChar before
comparison.
Fn $B9
StringLength
    AX: -> length of the cstr
   eDI: cstr
Returns the length of a cstr, excluding the terminating zero byte.
Fn $BA
StringValidateName fails
    AL: maximum number of characters permitted
    AH: non-zero if an extension is permitted
   eDI: cstr
The cstr is checked to see if it a valid name, and the call fails if it isnot. A name is valid
if it:
* begins with a letter;* contains only letters, digits, dollar signs ($), underscores (_), and
at
```

```
  most one dot;
* if AH is zero, does not contain a dot;* contains no more than the specified number of
characters before the dot
  (if any); and
* contains no more than 4 characters after the dot, if any.@The manual says 3 characters, but my
tests accept 4.@
Fn $BB
LongIntCompare
    AX: high half of P
    BX: low half of P
    CX: high half of Q
    DX: low half of Q
    SL: -> set if P less than Q
    EQ: -> set if P equals QCompares P and Q (both signed longs) and sets the result flags
accordingly.
Fn $BC
LongIntMultiply fails
    AX: high half of P -> high half of product
    BX: low half of P  -> low half of product
    CX: high half of Q
    DX: low half of QMultiples P and Q (both signed longs); the call fails if the result cannot
be represented as a signed long.
Fn $BD
LongIntDivide fails
    AX: high half of P -> high half of quotient
    BX: low half of P  -> low half of quotient    CX: high half of Q -> high half of
remainder
    DX: low half of Q  -> low half of remainderDivides P by Q (both signed longs); the call
fails if Q is zero. The remainder
will have the same sign as P.
Fn $BE
LongUnsignedIntCompare
    AX: high half of P
    BX: low half of P
    CX: high half of Q
    DX: low half of Q
    UL: -> set if P less than Q
```

```
    EQ: -> set if P equals QCompares P and Q (both unsigned longs) and sets the result flags
accordingly.
Fn $BF
LongUnsignedIntMultiply fails
    AX: high half of P -> high half of product
    BX: low half of P  -> low half of product
    CX: high half of Q
    DX: low half of QMultiples P and Q (both unsigned longs); the call fails if the result cannot
be represented as a unsigned long.
Fn $C0
LongUnsignedIntDivide fails
    AX: high half of P -> high half of quotient
    BX: low half of P  -> low half of quotient    CX: high half of Q -> high half of
remainder
    DX: low half of Q  -> low half of remainderDivides P by Q (both unsigned longs); the call
fails if Q is zero.
Fn $C1
FloatAdd fails
Fn $C2
FloatSubtract fails
Fn $C3
FloatMultiply fails
Fn $C4
FloatDivide fails
    SI: address of Q
    DI: address of PCalculates the specified one of P+Q, P-Q, P*Q, or P/Q, and places the result
in P. Both P and Q are reals.
Fn $C5
FloatCompare
    SI: address of Q
    DI: address of P
    SL: -> set if P is less than Q
    EQ: -> set if P equals Q
Compares two reals and sets the result flags appropriately.
Fn $C6
FloatNegate
    DI: address of P
```

```
Negates a real in-situ.
Fn $C7
FloatToInt fails
Fn $C8
FloatToUnsignedInt fails
    AX: -> result
    SI: address of realConverts a real to a signed or unsigned int; the call fails if the result
is
out of range. FloatToUnsignedInt ignores the sign of the real.
Fn $C9
FloatToLong fails
Fn $CA
FloatToUnsignedLong fails
    AX: -> high half of result
    BX: -> low half of result
    SI: address of realConverts a real to a signed or unsigned long; the call fails if the
result is
out of range. FloatToUnsignedLong ignores the sign of the real.
Fn $CB
IntToFloat
Fn $CC
UnsignedIntToFloat
    AX: value
    DI: address of real
Converts a signed or unsigned int to a real.
Fn $CD
LongToFloat
Fn $CE
UnsignedLongToFloat
    AX: high half of value
    BX: low half of value
    DI: address of real
Converts a signed or unsigned long to a real.
Fn $CF
LibSend
Fn $D0
LibSendSuper
```

```
    AX: -> method result
    BX: handle of object to receive the message
    CX: message number
    DX: argument 1
    SI: argument 2
    DI: argument 3These functions send a message to an object and invoke a method on that object.
LibSend starts searching for the method in the class of the object, andLibSendSuper in the
superclass.
Fn $D1 to $D3 can only be used from assembler.
Fn $D4
Dummy
This function has no effect.
Fn $D5
GenIntByNumber
    AL: Fn -> error and result flags
    SI: argument block
    DI: result blockThis call is equivalent to the OS keyword; it calls another system call with
the arguments and results stored in 12 byte blocks.
Fn $D6 is used for the Window Server, and is described in the Psionics file
WSERVER.
Fn $D7 is normally only used by the C library.
Fn $D8 is used for accessing DBF files. See the Psionics file DBF.FMT.
Fn $D9
LibEnterSend
    AX: -> method result
    BX: handle of object to receive the message
    CX: message number
    DX: argument 1
    SI: argument 2
    DI: argument 3This is identical to LibSend, except that it also starts a new "entry-exit
region". The method description will state when this is needed.
Fn $DA
IoKeyAndMouseStatus
@No documentation available at present@
Fn $DB
StringCapitalise
    SI: cstr
```

The first character of the cstr is passed through CharToUpperChar, and theremaining characters through CharToLowerChar.

Fn $DC

ProcIndStringCopyFromById fails

    BX: process ID

    CX: maximum length to copy

    SI: address of location holding address of cstr

   eDI: buffer

This copies a cstr from the indicated process to the current process. Thecstr is found via a pointer also in the memory of the indicated process, andthe address of this pointer is specified. This call is equivalent to makingtwo calls to ProcCopyFromById, the first to find the location of the cstr, andthe second to fetch it, except that the contents of the buffer beyond the

terminating zero are unspecified.

Fn $DD is reserved for the window server.

Fn $DE

IoSerManager@No documentation available at present@

Revision #1

Created Thu, Jan 24, 2019 10:34 AM by Alex

Updated Thu, Jan 24, 2019 10:34 AM by Alex