PROCESS

```
PSIONICS FILE - PROCESS
=======================
Processes and their properties
Last modified 1997-09-11
==============================
The Series 3 operating system is a fully multitasking one; at any time it canbe running any
number of processes (up to some limit). Much useful informationabout processes can be displayed
with the SPY program.
Each process has a name, a task number, and a process ID. The task number runsfrom 1 upwards
(the Series 3 is limited to 24), and a task number can bereused whenever a process terminates.
The process ID consists of the addressof the control block for the process in kernel memory in
the bottom 12 bits,
and a simple counter in the top 4 bits; this starts at 0 for the firstprocess to use this
control block, and is incremented whenever the block is
reused.
The name of a process is normally the name of the program (1 to 8 characters),followed by ".$"
(or ".@" for subtasks), and the task number as 2 decimaldigits. The code for a process is placed
in a segment called by the name ofthe program followed by ".SC". Thus it is not possible to run
two programs
with the same name at the same time.
Each process has a priority. Non-operating system processes are limited topriority 192; the
normal priorities are 128 for foreground and 112 forbackground. Only the highest priority
process(es) able to run will do so; all
lower priority ones will wait.
System processes
----------------
There are two kinds of system processes: fixed ones and optional ones. Thefixed processes
are:
    SYS$NULL.$01   handles power off
    SYS$MANG.$02   the process and resource manager
    SYS$FSRV.$03   the file server
    SYS$WSRC.$04   the window server
```

```
    SYS$SHLL.$05   the process launcher and system screen    TIME.$06       the time and alarm
server
The optional processes are:
    SYS$NCP.$??    the remote link manager
Registers
---------
Processes may use the data registers (AX, BX, CX, DX, SI, and DI) withoutrestriction. The kernel
may move memory segments about in memory at any timewithout warning; when it does so, those of
the segment registers (CS, DS, ES,and SS) that appear to the kernel to be valid will be adjusted
automatically.
If it is desired to change these registers at any other time, it shouldtherefore be done with
interrupts disabled.
OPL uses CS, DS, and SS, and these should not be altered by assembler codeinvoked from with OPL.
ES is not used by the OPL interpreter, and is normallyleft the same as DS and SS (which are
always the same). Many system callsuse data registers to hold pointers; a segment register is
also used with
each such pointer. The Psionics files SYSCALLS.n indicate when an address isusing ES; they do
not distinguish DS and SS. Any assembler code called from
OPL should exit with ES equal to DS.
The only other values that can usefully be placed in ES are the kernel datasegment (using system
call GenDataSegment), and segments created with thesystem call SegCreate. To do the former, just
invoke INT $8F; ES will thenpoint to the kernel data segment, and accesses via ES will behave
just as
if done with the system call GenGetOsData (it is not possible to write inthis way). The latter
is done as follows:
    ; Assume the segment handle is in BX.
    INT $8F
    MOV ES,ES:[BX]
    ; ES now points to the start of the segment.    ; If the segment moves, ES will change
automatically.
    ; Before returning to OPL:
    CLI      ; with interrupts disabled
    PUSH DS  ; copy DS to ES
    POP ES
    STI      ; and allow interrupts again
Memory
------
```

Each process has a single segment of memory. From zero upwards, this contains: reserved statics

  stack (grows downwards)

  normal statics

  heap (grows upwards)

The heap is divided into allocated and free cells. The SPY program shows thenumber and total byte count of each kind of cell. The kernel will remove freecells from the end of the heap when it needs space.

The stack is initialized to all $FF; it must always hold at least 256 bytesto allow interrupts to be processed (otherwise panic 69 will occur). A typicalstack size is $A00. The SPY program allows the unused portion of the stack tobe refilled with $FF (to allow the high water mark to be reset).

The heap consists of consecutive cells:

  Offset 0 (word): length of data portion (L) Offset 2 to L+1: data portion (whose address is returned by allocators)

@Is that right, or is it 2 to L-1 ?@

Cells are always aligned to even addresses, and for this and other reasons maybe larger than requested.

Specific memory locations

The following table lists some specific use of memory locations.

  Offset 0 (word): initialized to $DEAD; should never change Offset 2 to 5: used by the kernel

  Offset 6 (word): handle of the segment holding the fold tables @@ Offset 8 to 15: used by the object oriented programming system

  Offset 16 (word): used by the kernel

  Offset 18 (word): address of block <P18> Offset 20 (word): handle of the application manager object

  Offset 22 to 25: used by the window server

  Offset 26 to 29: used by the OPL runtime system

  Offset 30 (word): used by the debugger Offset 32 (byte): zero when address trapping is turned off (do not alter)

  Offset 33 (byte): non-zero when the kernel is modifying the heap Offset 34 (word): address of cstr holding the program name; used to determine which list on the system screen a process appears in (@if this word is zero, the process will not be sent X messages) Offset 36 (word): address of a heap cell containing a cstr holding the full path name of the program being executed, followed by

a
qstr holding the program information (see system call
FilExecute).
Offset 38 (word): not used by the kernel or OPL; used by the debugger Offset 40 to 53: not used by the kernel or OPL
Offset 54 (word): address of current dialog structure Offset 56 (word): @@ DatGate object handle
Offset 58 (word): non-zero if locked (by OPL "LOCK ON" or equivalent) Offset 60 (word): address of cstr to appear in the status window Offset 62 (word): address of cstr holding current file being processed Offset 64 : maximum limit of stack (if floating point emulation is
in use, the emulator uses offsets 64 to 767).
Certain of these memory locations have names in the system documentation: 34 DatProcessNamePtr
36 DatCommandPtr
40 DatApp1
42 DatApp2
44 DatApp3
46 DatApp4
48 DatApp5
50 DatApp6
52 DatApp7
58 DatLocked
60 DatStatusNamePtr
62 DatUsedPathNamePtr
@Location "winHandle" is the channel number of the console device.
Block <P18> has the following contents:
@this is a "wserv" object@
Offset 32 (word): address of block <P18P32>
Block <P18P32> has the following contents:
Offset 12 (word): address of block <P18P32P12>
Block <P18P32P12> is created by the MINIT keyword and destroyed by the MENUkeyword. Outside this range, the block is invalid, and the pointer in the<P18P32> block is invalid. Each MCARD keyword can move this block, thus
altering that pointer. The block has the following contents: Offset 0 (byte): number of MCARD calls so far
Offset 1 onward: menu information blocks, in order of menu creation

Each menu information block has the following contents: Offset 0 (word): address of secondary menu information block

  Offset 2 (cstr): menu title

Each secondary menu information block has the following contents: Offset 0 (long): [Only ever seen $7392A1DF]

  Offset 4 (word): number of items in the menu

  Offset 6 (long): [Only ever seen 1 so far]

  Offset 10 (word): address of item table

Each item table has the following contents:

  Offset 0 (byte): number of items in the menu Offset 1 onward: item information blocks, in order of item

Each item information block has the following contents: Offset 0 (byte): number of bytes in the block, excluding this one

  Offset 1 (byte): Psion+ accelerator code (lowercased)

  Offset 2 (cstr): item text

Time process

The pending alarms can be located in the memory of the Time process. At somelocation between offset $0A00 and $0AFF is a block with the form:

  Offset 0 (long): $08040201

  Offset 4 (long): $00402010

  Offset 12 (word): first alarm control blockIt is reported that on the Series 3c this has moved to somewhere between

$0C00 and $0CFF. It is also reported that on some versions the controlblock is at offset 10 or 15.

The alarm control blocks are in a circular list, with the last pointing tooffset 12 (or 10 or 15) of the above block (which is not an alarm controlblock); if that location points to itself, there are no alarms. Each block

has the form:

  Offset 0 (word): next alarm control block Offset 2 (word): reported that this is the second next alarm control block

  Offset 6 (word): process ID of process owning alarm Offset 8 (byte): 0 = clock alarm, 1 = has time and date, 2 = has date

  Offset 10 (long): abstime alarm expires

  Offset 14 (long): abstime displayed in alarm message

  Offset 18 (qstr): name of alarm sound

  Offset 28 (cstr): text displayed in alarm message Offset 93 to 255: [unknown; alleged to be

```
part of the block]
The built-in alarm sounds have names consisting of a single byte (thusoffset 18 is 1 and offset
19 holds the name): 1=rings, 2=chimes, and
16=silence.
@System screen or Window Server (which?) places info about 14 applicationbuttons at DatApp1.@
```

Revision #1
Created Thu, Jan 24, 2019 10:30 AM by Alex
Updated Thu, Jan 24, 2019 10:30 AM by Alex