New Operating System
Version 1.0

For the
DSE VZ200/300 Computer’s

Contents

Introduction

What are the aims of this Project?

The main aim of this project is to allow easier access to any peripheral devices attached to the VZ by any piece of software already written for the VZ with little or no modification.

The main assumptions before you commence this project are:-

1) You have **reasonable** knowledge of Z80 Assembly Language Programming.
2) You own a VZ with at least a 16K RAM Expansion.
3) You have time to do the work required.

The Vision

A VZ Computer with a maximum of 4 Megabytes of memory (available to all programs - even BASIC Programs), Multiple Disk Drives attached (including a 720K 3.5 inch Drive), Enhanced Keyboard, optional RAM Disk Software, A Real-Time Clock keeping track of the Date and Time allowing Alarm Features etc, a Mouse or other pointing device, an 80 column display for better viewing of data, and to round off the set-up - a Sound Blaster Card allowing you to record and play back sounds from various input devices (TV, Video, Radio, Stereo etc).

We are aiming High!

The BASIC's

All components in **NOS** are designed to be replaceable, so that you may configure your system in any way you would like. The system has been designed to emulate the DOS System of the popular IBM Personal Computers. That is, it will provide similar functionality but will **not** neccessarily be compatible. **NOS** will consist of nine basic components:-

B:	**BOOT.EXE**	This is the **NOS** Initialisation Program.
B:	**MEMORY**	This is the Memory Manager and will look after all memory allocation regardless of its location.
B:	**PROCMAN**	This is the Process Manager and looks after the loading/executing of all Programs.
B:	**SCRNMAN**	This is the Screen/Display Manager and looks after the different Graphics Modes etc.
B:	**DISKMAN**	This is the Disk Manager and shields all applications from the native Disk Drive Commands.
B:	**PRINTMAN**	This is the Print Manager and looks after the translation of control codes and provides a Printer Spooler.
T:	**CONFIG**	This file contains the Peripheral Device Driver installation/configuration information.
B:	**COMMAND**	This is the Command processor and contains the basic reserved words allowed at the command line.
T:	**AUTOEXEC**	This file contains any other Programs the User may wish to load when starting **NOS**.

Notes/Assumptions:

1) The files listed above **must** all live on the same diskette - which will from this point forth be referred to as the **NOS Boot Disk.**
2) The **NOS Boot Disk** is a standard VZ Diskette and as such can only be read from a VZ Disk Drive. This means that we will have two limitations: 80K Maximum Boot Disk size and also we cannot boot from a different type of Disk Drive yet. These are not major hurdles and will be resolved in the future.

The Initialisation Process (BOOT.EXE)

As mentioned above, the entry point into **NOS** will be via **BOOT.EXE** which, as with standard VZ disk software, will be started with the usual BRUN"BOOT.EXE". Once loaded, the Boot program will perform some minor tasks such as setting up internal pointers and reserving permanent memory areas for itself. It will then commence to load the other programs from the boot disk in the order listed above. The loading order is essential as some of the processes reply upon the previous components being there to operate correctly. Below is the pseudo code for the Boot process, see **Appendix A** for the executable version.

```
BeginProgram 'Boot'
Origin: 40960                                              ; Origin = A000H

    Disable Interrupts                                     ; Prevent the System from having problems
                                                           ; whilst initialisation occurs.

    Reserve 25 Bytes for the System Information Frame (SIF)  ; See later for further details
    at locations 31465 to 31489

    Initialise SIF:                                        ; Values are for NOS 1.0 only.

        Store NOS Version Description Pointer              ; = 31490
        Store High Order NOS Version                       ; = 1
        Store Low Order NOS Version                        ; = 0
        Initialise the System Flags                        ; = 0
        Initialise the Max Process ID                      ; = 255
        Set the Base System Stack Pointer                  ; = 31597 + 40 (Default Stack Size)
        Set the Top System Stack Pointer                   ; = 31597
        Set the Current System Stack Pointer               ; = Base System Stack Pointer
        Store IM1 Table Pointer                            ; = 31507
        Initialise IM1 Function Count                      ; = 0
        Store the Critical Error Handler Address           ; = 31636

    ; All following addresses are known to the Boot Program only. Any Process wishing to use the following
    ; information should go via the SIF to retain compatibility with future versions of NOS.

    Store Version Description String at location 31490 (16 Bytes)   ; "NOS Version 1.0"

    Initialise Interrupt Device Table at location 31507    ; 31507 - 31596 = 0
    (max 30 Functions - 3 Bytes each = 90 Bytes)

    Reserve 40 Bytes for the NOS System Stack at location 31597

    Install the Critical Error Handler at location 31636   ; See later for further details

    Install the Interrupt Processing Subroutine for IM1    ; See later for further details
    executed via an RST 38H instruction.

    ; Load the remaining NOS System Components
    EXECUTE[ MEMORY ]                                      ; Install the Memory Manager
    EXECUTE[ PROCMAN ]                                     ; Install the Process Manager
    EXECUTE[ SCRNMAN ]                                     ; Install the Screen Manager
    EXECUTE[ DISKMAN ]                                     ; Install the Disk Manager
    EXECUTE[ PRINTMAN ]                                    ; Install the Printer Manager

    Call SYS Interpreter [ CONFIG ]                        ; Read the CONFIG file line by line and
                                                           ; perform the specified commands

    EXECUTE[ COMMAND ]                                     ; Install the Command Line Processor

    Call Batch Interpreter[ AUTOEXEC ]                     ; Read the Autoexec file line by line and
                                                           ; perform the specified commands.
    Enable Interrupts

End Program 'Boot'                                         ; Return control to the User (NOS Installed)
```

The System Information Frame (SIF)

This is an area of memory which contains all the important parameters (offsets) needed by the system to handle the multi-processing and the communication between processes and **NOS**. The definition of **SIF** will continue to change as we add further information as **NOS** expands. Once an offset has been declared in **SIF**, it will not be removed. Also, the location of **SIF** must never relocate and is fixed at location 31465 decimal (7AE9H). This way backwards compatibility will be retained so that software written for **NOS** version 1.0 will still run under a future version without change.

The current definition of SIF is as follows:-

Offset	# of Bytes	Description
0	2	Size of SIF
2	2	NOS Version Description Pointer
4	1	High Order NOS Version
5	1	Low Order NOS Version
6	2	Base System Stack Pointer
8	2	Top System Stack Pointer
10	2	Current System Stack Pointer
12	2	Reserved
14	1	Reserved
15	2	IM1 Interrupt Device Table
17	1	No. entries in IM1 Interrupt Device Table
18	2	Reserved
20	1	Reserved
21	1	System Flags: Bit 0 = Stack In Use Indicator 0 - SP is the Current Process Stack 1 - SP is the System Stack Bits 1 & 2 = Interrupt Processing Indicator 00 - No Interrupt is being Processed 01 - Reserved 10 - IM1 Interrupt is being Processed 11 - Reserved Bits 3 & 4 = Stack Request after Interrupt 00 - No Request 01 - Request System Stack after Interrupt 10 - Request Current Process Stack after Interrupt 11 - Reserved Bits 5 - 7 = Reserved for future expansion Set to Zero (000)
22	1	Last Process ID. (Total No. of Processes currently loaded)
23	2	Address of the Critical Error Handler

The Critical Error Handler

The Critical Error Handler is just that, a centralised error checking routine which given an error code produces a message and then either returns control to allow the calling routine to cleanup as best it can, or it terminates the parent process if a major error has occured. For a full list of error messages refer to **Appendix ??**.

The IM1 Interrupt Processing Subroutine

The standard VZ hardware configuration is such that only the IM1 type interrupts of the Z80 CPU have been enabled. This is the simplest of the three types of interrupts the Z80 permits. When an interrupt occurs an **RST 38H** instruction is executed which in turn calls a small service routine. Currently, the only interrupting device on a standard VZ is the Graphics Controller Chip which interrupts 50 times every second. The current interrupt service routine only looks after screen updates, such as flashing the cursor, displaying messages and looking after the background colours.

Over the years a lot of software and hardware enhancements have meant that the standard service routine was replaced with a customised device specific version. This was a good solution until the time came where more than one add-on was required at any one time. The Interrupt Service routine installed by **NOS** will attempt to overcome this problem.

For the initial version of **NOS** we will only be considering IM1 type interrupts. It is most likely that this will change in a future version, but hardware enhancements will also be required.

In **NOS** there is a table which contains a list of functions, in priority order, which are to be called every time an interrupt ocurrs. The address of this table is stored at offset 15 of **SIF** and is known as the IM1 Device Table. For this version of NOS there is a limit of 30 functions which can be called every time an interrupt occurs. Each function looks after a specific device and whenever called polls the device to see if any events occurred since the last interrupt. The one major drawback to this method is that if we do not poll a device often enough we will loose information. The only way to resolve this is via some form of hardware enhancement which is beyond the scope of the initial version.

Questions:

How do we handle interrupts of devices which require different polling priorities
there is a limit on the amount of time we can spend servicing interrupts before the system slows down.

The Memory Manager

The Memory Manager controls the allocation/distribution of memory for use by all programs.
Memory on the VZ can be defined as one of four types:-

1. System Memory
2. Extended Memory
3. Expanded Memory
4. Screen Memory

All but the last type of memory is looked after by the Memory Manager. The minimum expected configuration is System memory only, although this will severely restrict the user and negate any of the benefits of **NOS**.

System Memory

This is the area of memory between 7AE9H and BFFFH, and is mandatory. Applications should not use this memory unless absolutely necessary. This memory is used by NOS for loading Device Drivers, lookup tables, file control blocks etc.

To access System Memory, the following primitives are available:

i)	Allocate_System	- This marks a block of system memory as being used and returns a pointer to it.
ii)	ReAllocate_System	- This increases the size of an allocated block of system memory and returns a pointer to it.
iii)	Release_System	- This releases a block of allocated system memory.
iv)	TotalFree_System	- The returns the amount of unused system memory.
v)	LargestFree_System	- The returns the size of the largest unused block of system memory.
vi)	SmallestFree_System	- The returns the size of the smallest unused block of system memory.
vii)	FirstFree_System	- The returns the size of the first unused block of system memory.
viii)	NextFree_System	- The returns the size of the next unused block of system memory.

Extended Memory

This is the area of memory starting at C000H. Depending upon the hardware configuration, the end address can vary. The memory in this region may also be bank switched.

To access extended memory, the following primitives have been provided:

i)	Allocate_Global	- This marks a block of extended memory as being used and returns a pointer to it.
ii)	ReAllocate_Global	- This increases the size of an allocated block of extended memory and returns a pointer to it.
iii)	Release_Global	- This releases a block of allocated extended memory.
iv)	TotalFree_Global	- This returns the amount of unused extended memory.
v)	LargestFree_Global	- This returns the size of the largest unused block of extended memory.
vi)	SmallestFree_Global	- This returns the size of the smallest unused block of extended memory.
vii)	FirstFree_Global	- This returns the size of the first unused block of extended memory.
viii)	NextFree_Global	- This returns the size of the next unused block of extended memory.
ix)	NoBanks_Global	- This returns the number of extended memory banks.
x)	TotalMem_Global	- This returns the total amount of extended memory.

Please note that these primitives return a three byte (24 bit) pointer. The following additional primitives have been provided to store and retrieve data from extended memory:

i)	StoreByte_Global	- This stores a byte of data in global extended memory.
ii)	ReadByte_Global	- This reads a byte of data from global extended memory.
iii)	StoreBlock_Global	- This transfers a block of data to global extended memory.
iv)	ReadBlock_Global	- This reads a block of data from global extended memory.
v)	TransferBlock_Global	- This transfers a block of data from one area of global extended memory to another.

The Process Manager

The Process Information Frame (PIF)

The current definition of PIF is as follows:-

Offset	# of Bytes	Description
0	2	Size of PIF
2	2	Start Address
4	1	Bank No
5	1	Process ID
6	2	Base Process Stack Pointer
8	2	Top Process Stack Pointer
10	2	Current Process Stack Pointer
12	2	Function Table Pointer
14	1	Number of External Functions
; Inter-Process Communication Call/Results		
15	1	Called Process No
16	1	Sub Function No
17	1	ES: Call Error Status/Single Return Value
18	2	RV1: Double Byte Return Value
20	2	RV2: Double Byte Return Value

Appendix ? - Error Messages produced by the Critical Error Handler

Error Code	Terminate/Suppress	Reason for Error	Error Message
0	-	No Error	-
1	Suppress	Invalid Process ID	A Requested Process is not available
2	Suppress	Invalid Function Call	A Requested Function is not available
3	Terminate	Stack Underflow Error	A Stack Error has Occurred.Application is terminating
4	Terminate	Stack Overflow Error	A Stack Error has Occurred.Application is terminating

NB: The second column indicates that the parent process can suppress these error messages and resolve the problem programatically.

Appendix ? - Standard Process Identifier's

Process ID	Description
; NOS Services	
0	Reserved
1	Memory Manager
2	Process Manager
3	Screen Manager (Video I/O)
4	Disk Manager
5	Print Manager
; Device Drivers - Standard Hardware	
6	Keyboard Input/Output
7	Video Input/Output
8	Cassette Input/Output
9	Speaker Output
; Device Drivers - Standard Hardware Extensions	
10	Parallel Printer Output
11	Joystick Input
12	Disk Input/Output
; Device Drivers - Non-standard Hardware Extensions Support	
13	Mouse Input
14	Real-Time Clock Input/Output
15	Serial Input/Output
; Device Drivers - Logical Devices	
16	Standard Input
17	Standard Output
18	Standard Printer
19	Standard Error
;Reserved	
20 - 24	Reserved for Future Expansion.
; Process Manager allocates from 25 to 255	

Appendix ? - Useful Definitions

To make the code easier to read, the following definitions have be created:

```
SIF                      EQU   31465
SYSTEM_FLAGS             EQU   SIF + 19
PROCESS_LOOKUP_TABLE     EQU   SIF + 23
SYSTEM_STACK             EQU   SIF + 10
LAST_PROCESS_ID          EQU   SIF + 22
```

Appendix ?? - The IM1 Interrupt Handler

Upon Entry: Stack contains all registers except IY & IX.

```
INTR:   LD      A, (SYSTEM_FLAGS)       ; Indicate that we are servicing IM1 Interrupts.
        AND     11111101b               ; Preserve all bits other than bits 1 & 2.
        LD      (SYSTEM_FLAGS), A

        POP     HL                      ; Remove the return address to the ROM routine from the stack.

        PUSH    IY                      ; Save the Current Process Frame Pointer.
        PUSH    IX                      ; Save the IX Register.

        BIT     0,A                     ; Check if System Stack is already in use.
        JR      NZ, INT1

        LD      IY, SIF                 ; Set Frame Pointer to SIF.

INT1:   LD      HL, -20                 ; Recalculate the Current Stack Pointer allowing for the
        ADD     HL, SP                  ; registers already placed there by the ROM routine.
        LD      (IY + 10), L
        LD      (IY + 11), H            ; HL = Current Stack Pointer, could be the Process Stack.
                                        ; SP is unaffected.

        CALL    VSTACK                  ; Validate that the stack integrity is OK.

                |
                |
        SERVICE INTERRUPTS              ; Refer Appendix ?? for further Information.
                |
                |

        POP     IX                      ; Restore the IX Register.
        POP     IY                      ; Restore the Current Process Frame Pointer.

        POP     HL                      ; Restore registers placed on the stack by the ROM routine.
        POP     DE
        POP     BC

        LD      A, (SYSTEM_FLAGS)       ; Reload the System flags.
        BIT     3, A                    ; Test for System Stack request.
        JR      NZ, INT3
        BIT     4, A                    ; Test for Process Stack request.
        JR      NZ, INT4

INT2:   AND     11100001b               ; Reset: Interrupt Indicator & Stack request.
        LD      (SYSTEM_FLAGS), A
        POP     AF
        EI
        RETI

INT3:   BIT     0, A                    ; Test if System Stack already in use
        JR      NZ, INT2                ; Goto INT2 if so.
        AND     11100001b               ; Reset: Interrupt Indicator & Stack request.
        SET     0, A                    ; Indicate SP = System Stack.
        LD      (SYSTEM_FLAGS), A

        POP     AF                      ; Restore Registers placed there by ROM routine.
```

```
                ; Because we are messing around with the stack pointer we cannot rely upon it.

                LD      (SAVE_REG1), HL         ; Save the current value of HL
                POP     HL                      ; HL = Commencement Address when returning from interrupt.
                                                ; This is not the return address to the ROM routine as this has
                                                ; already been removed.
                LD      SP, (SIF + 10)          ; SP = Current System Stack Pointer.
                PUSH    HL                      ; Restore the Commencement Address
                LD      HL, (SAVE_REG1)         ; Restore value of HL
                EI
                RETI

INT4:           BIT     0, A                    ; Test if System Stack in use
                JR      Z, INT2                 ; Goto INT2 if not.

                AND     11100001b               ; Reset: Interrupt Indicator & Stack request.
                RES     0, A                    ; Indicate SP = Current Process Stack.
                LD      (SYSTEM_FLAGS), A

                POP     AF                      ; Restore Registers placed there by ROM routine.


                ; Because we are messing around with the stack pointer we cannot rely upon it.

                LD      (SAVE_REG1), HL         ; Save the current value of HL

                POP     HL                      ; HL = Commencement Address when returning from interrupt.
                                                ; This is not the return address to the ROM routine as this has
                                                ; already been removed.
                LD      (SAVE_REG2), HL         ; Save it

                LD      L, (IY + 10)            ; HL = Current Process Stack Pointer
                LD      H, (IY + 11)

                LD      SP, HL                  ; SP = Current Process Stack Pointer
                LD      HL, (SAVE_REG2)
                PUSH    HL                      ; Restore Commencement Address
                LD      HL, (SAVE_REG1)         ; Restore value of HL
                EI
                RETI


                ; These storage areas are known only to the Interrupt Service Routine.

SAVE_REG1:      DEFS    2
SAVE_REG2:      DEFS    2
```

Appendix ?? - Stack Validation Routine

Upon Entry: HL = Stack Pointer.

Upon Exit: HL register altered,
Flags register altered,
All other registers preverved.

```
VSTACK:         PUSH  DE
                LD    E, (IY + 6)
                LD    D, (IY + 7)             ; DE = Base of Stack.
                EX    DE, HL                  ; DE = Current Stack Pointer.
                                              ; HL = Base of Stack.
                OR    A                       ; Clear carry flag.
                SBC   HL, DE
                JR    ?, VSTACK_UFLOW         ; Stack Underflow Error.

                LD    L, (IY + 8)
                LD    H, (IY + 9)             ; HL = Top of Stack.
                OR    A
                EX    DE, HL                  ; HL = Current Stack Pointer.
                                              ; DE = Top of Stack.
                SBC   HL, DE
                JR    ?, VSTACK_OFLOW         ; Stack Overflow Error.

                OR    A                       ; Clear carry flag.
                POP   DE
                RET

VSTACK_UFLOW:   LD    E, A                    ; Preserve the A register.
                LD    A, 3
                CALL  CRITICAL_ERRORHANDLER
                LD    A, E                    ; Restore the A register.
                POP   DE
                RET

VSTACK_OFLOW:   LD    E, A                    ; Preserve the A register.
                LD    A, 4
                CALL  CRITICAL_ERRORHANDLER
                LD    A, E                    ; Restore the A register.
                POP   DE
                RET
```

Appendix ?? - Inter-Process Function Call Routine (RST 30H)

Upon Entry: IY = Current Process **PIF**.

Upon Exit: IY = Current Process **PIF**.
All other registers **destroyed**.

```
IP_CALL:     LD    A, (SYSTEM_FLAGS)           ; Get the current System Flags.
             BIT   1, A                        ; Is the call occurring during interrupt processing.
             JR    NZ, IP_CALL1                ; Goto IP_CALL1 if so.

             HALT                              ; Wait for an interrupt. This serves two purposes:
                                               ;
                                               ; i)  Yields to O/S allowing background
                                               ;     processes to occur.
                                               ; ii) Ensures stack integrity.

IP_CALL1:    LD    H, 0
             LD    L, (IY + 13)                ; HL = Called Process No.

             LD    A, (LAST_PROCESS_ID)        ; A = ID of last Process loaded.
             CP    L                           ; Check if Called Process No is valid
             LD    A, 1                        ; Set Errorflag.
             JP    ?, CRITICAL_ERRORHANDLER    ; "Invalid Process ID".

             ADD   HL, HL                      ; *2
             PUSH  HL
             ADD   HL, HL                      ; *2     (*4)
             ADD   HL, HL                      ; *2     (*8)
             POP   DE
             ADD   HL, DE                      ;        (*10)
             ADD   HL, HL                      ; *2     (*20)

             EX    DE, HL                      ; DE = Table Offset
             LD    IX, (TABLE_BASE_ADDRESS)
             ADD   IX, DE                      ; IX = Pointer to Called Process PIF.

             LD    E, (IX + 10)
             LD    D, (IX + 11)                ; DE = Function Table Pointer.

             LD    A, D                        ; Check that the Process ID is valid.
             OR    E
             JP    Z, CRITICAL_ERRORHANDLER

             LD    A, (IX + 12)                ; A = No of available functions.
             LD    L, (IY + 14)                ; L = Function No.

             CP    L                           ; Compare them.
             LD    A, 2
             JP    ?, CRITICAL_ERRORHANDLER    ; "Invalid Function Call".

             XOR   A                           ; A = 0, carry flag cleared.
             SLA   L
             RLA
             LD    H, A
             ADD   HL, DE                      ; HL = Table Address of function.

             LD    A, (SYSTEM_FLAGS)
             BIT   0, A                        ; Check if System stack in use.
             JR    NZ, IP_CALL3                ; Goto IP_CALL3 if not.
```

```
                ; System Stack is not in use.

                BIT     1, A                    ; Check if interrupt being processed.
                JR      Z, IP_CALL2             ; Goto IP_CALL2 if not.


                ; Currently Processing Interrupts.

                EX      DE, HL                  ; Save Table Address in DE
                LD      HL, 0
                ADD     HL, SP                  ; HL = Current Stack Pointer

                LD      (IY + 8), L             ; Save current Stack pointer.
                LD      (IY + 9), H

                CALL    VSTACK                  ; Validate that the stack integrity is OK.

                EX      DE, HL                  ; HL = Table address of Function.
                LD      SP, (SYSTEM_STACK)      ; Set the Stack Pointer to the System Stack.

                SET     0, A
                LD      (SYSTEM_FLAGS), A       ; Indicate System Stack in use.
                JR      IP_CALL3                ; GOTO IP_CALL3

IP_CALL2:       SET     2, A
                LD      (SYSTEM_FLAGS), A       ; Request System stack on next interrupt.
                HALT                            ; Wait for an interrupt.

IP_CALL3:       LD      A, (IX + 2)             ; A = Bank of Called Process.
                CALL    BANK_SWITCH             ; = OUT  (7FH), A

                LD      E, (HL)
                INC     HL
                LD      D, (HL)                 ; DE = Function Address.

                PUSH    IX                      ; Save Called Process PIF.
                EX      (SP), IY                ; Exchange it with current Process PIF.

                LD      A, (SYSTEM_FLAGS)
                BIT     1, A                    ; Check if interrupt being processed.
                JR      Z, IP_CALL4             ; Goto IP_CALL4 if not.

                ; Currently Processing Interrupts.

                SET     0, A                    ; Indicate Process Stack in use.
                LD      (SYSTEM_FLAGS), A

                LD      L, (IY + 8)
                LD      H, (IY + 9)             ; HL = Called Process Stack Pointer.

                LD      SP, HL                  ; SP = Called Process Stack Pointer.

                JR      IP_CALL5

IP_CALL4:       SET     3, A
                LD      (SYSTEM_FLAGS), A       ; Request current Process Stack on next interrupt.

                HALT                            ; Wait for an interrupt.

IP_CALL5:       LD      HL, (CLEANUP_ADDRESS)
                PUSH    HL
                EX      DE, HL
                JP      (HL)                    ; Execute Function.
```