; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)

; ----------------------------------------------------------------------------

; U6.ASM (include u6.asm) //// UNIX v1 -> u6.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)

; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)

; 1.44 MB Floppy Disk

; (11/03/2013)

;

; [ Last Modification: 23/07/2014 ] ;;; completed ;;;

;

; Derivation from UNIX Operating System (v1.0 for PDP-11)

; (Original) Source Code by Ken Thompson (1971-1972)

; <Bell Laboratories (17/3/1972)>

; <Preliminary Release of UNIX Implementation Document>

;

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; 23/07/2014 rtty

; 07/07/2014 wtty

; 27/06/2014 wtty (putc)

; 19/06/2014 rtty, wtty

; 03/06/2014 (rtty/wtty check is ok)

; 02/06/2014 wtty

; 26/05/2014 wtty

; 15/04/2014 rtty, wtty ('getc' and 'putc' error return modifications)

; 14/04/2014 wtty

; 23/02/2014 rtty

; 01/02/2014 rtty

; 13/01/2014 rtty, wtty

; 06/12/2013 rtty, wtty (major modification: p.ttyc, u.ttyp)

; 10/10/2013 rtty, wtty (tty read lock & tty write lock are removed)

; 05/10/2013 rtty, wtty

; 29/09/2013 rtty

; 20/09/2013 rtty & passc (tty read lock)

; wtty & cpass (tty write lock), dskw, rmem, wmem

; 13/09/2013 rtty

; 26/08/2013 wtty

; 14/08/2013 rtty, rcvt, wtty, xmtt, cpass

; 03/08/2013 dskr (namei\_r), dskw (mkdir\_w)

; 01/08/2013 dskw (mkdir\_w)

; 31/07/2013 dskr (namei\_r), writei

; 29/07/2013 rtty, idle

; 28/07/2013 rtty, rcvt, wtty, u.namei\_r

; 26/07/2013 readi

; 16/07/2013 rtty, rcvt, chk\_ttyp, rmem, wmem modifications

; 27/05/2013 chk\_ttyp

; 21/05/2013 chk\_ttyp, chk\_com\_o

; 20/05/2013 chk\_ttyp

; 15/05/2013 rcvt, xmtt, COM1, COM2

; 26/04/2013 readi, writei modifications

; 14/03/2013 -> writei

; 12/03/2013 -> writei, u.segment

; 11/03/2013

readi:

; 31/07/2013

; 26/07/2013 (namei\_r check in 'dskr')

; 15/05/2013 COM1, COM2 (serial ports) modification

; 26/04/2013 (modification depending on 'dsrkd' modification)

; 12/03/2013 -> u.segment

; 11/03/2013

; Reads from an inode whose number in R1

;

; INPUTS ->

; r1 - inode number

; u.count - byte count user desires

; u.base - points to user buffer

; u.fofp - points to word with current file offset

; OUTPUTS ->

; u.count - cleared

; u.nread - accumulates total bytes passed back

;

; ((AX = R1)) input/output

; (Retro UNIX Prototype : 01/03/2013 - 14/12/2012, UNIXCOPY.ASM)

; ((Modified registers: DX, BX, CX, SI, DI, BP))

xor dx, dx ; 0

mov word ptr [u.nread], dx ; 0

; clr u.nread / accumulates number of bytes transmitted

cmp word ptr [u.count], dx ; 0

; tst u.count / is number of bytes to be read greater than 0

ja short @f ; 1f

; bgt 1f / yes, branch

retn

; rts r0 / no, nothing to read; return to caller

@@: ; 1:

; mov r1,-(sp) / save i-number on stack

cmp ax, 40

; cmp r1,$40. / want to read a special file

; / (i-nodes 1,...,40 are for special files)

ja dskr

; ble 1f / yes, branch

; jmp dskr / no, jmp to dskr;

; / read file with i-node number (r1)

; / starting at byte ((u.fofp)), read in u.count bytes

push ax ; because subroutines will jump to 'ret\_'

@@: ; 1:

mov bx, ax

shl bx, 1

; asl r1 / multiply inode number by 2

add bx, offset @f - 2

jmp word ptr [BX]

; jmp \*1f-2(r1)

@@: ; 1:

dw offset rtty ; tty, AX = 1 (runix)

;rtty / tty; r1=2

;rppt / ppt; r1=4

dw offset rmem ; mem, AX = 2 (runix)

;rmem / mem; r1=6

;rrf0 / rf0

;rrk0 / rk0

;rtap / tap0

;rtap / tap1

;rtap / tap2

;rtap / tap3

;rtap / tap4

;rtap / tap5

;rtap / tap6

;rtap / tap7

dw offset rfd ; fd0, AX = 3 (runix only)

dw offset rfd ; fd1, AX = 4 (runix only)

dw offset rhd ; hd0, AX = 5 (runix only)

dw offset rhd ; hd1, AX = 6 (runix only)

dw offset rhd ; hd2, AX = 7 (runix only)

dw offset rhd ; hd3, AX = 8 (runix only)

dw offset rlpr ; lpr, AX = 9 (invalid, write only device !?)

dw offset rcvt ; tty0, AX = 10 (runix)

;rcvt / tty0

dw offset rcvt ; tty1, AX = 11 (runix)

;rcvt / tty1

dw offset rcvt ; tty2, AX = 12 (runix)

;rcvt / tty2

dw offset rcvt ; tty3, AX = 13 (runix)

;rcvt / tty3

dw offset rcvt ; tty4, AX = 14 (runix)

;rcvt / tty4

dw offset rcvt ; tty5, AX = 15 (runix)

;rcvt / tty5

dw offset rcvt ; tty6, AX = 16 (runix)

;rcvt / tty6

dw offset rcvt ; tty7, AX = 17 (runix)

;rcvt / tty7

dw offset rcvt ; COM1, AX = 18 (runix only)

;rcrd / crd

dw offset rcvt ; COM2, AX = 19 (runix only)

rtty: ; / read from console tty

; 19/06/2014

; 15/04/2014 ('getc' error return modifications)

; 23/02/2014

; 01/02/2014

; 13/01/2014

; 06/12/2013 (major modification: p.ttyc, u.ttyp)

; 10/10/2013

; 05/10/2013

; 29/09/2013

; 20/09/2013 (tty read lock)

; 13/09/2013

; 14/08/2013

; 28/07/2013 u.ttyn

; 16/07/2013

; 16/07/2013 'getc' modifications

; 20/05/2013

; 15/05/2013 'getc' error return for serial ports

; 14/05/2013 'getc' modifications instead of INT 16h

; 11/03/2013

; Console tty buffer is PC keyboard buffer

; and keyboard-keystroke handling is different than original

; unix (PDP-11) here. TTY/Keyboard procedures here are changed

; according to IBM PC compatible ROM BIOS keyboard functions.

;

; 06/12/2013

mov bl, byte ptr [u.uno] ; process number

xor bh, bh

mov al, byte ptr [BX]+p.ttyc-1 ; current/console tty

rttys:

; mov tty+[8\*ntty]-8+6,r5 / r5 is the address of the 4th word of

; / of the control and status block

; tst 2(r5) / for the console tty; this word points to the console

; / tty buffer

; 28/07/2013

mov byte ptr [u.ttyn], al

; 06/12/2013

;; 13/01/2014

;;cmp al, 7

;;ja short rtty\_nc

inc al

mov byte ptr [u.ttyp], al ; tty number + 1

rtty\_nc: ; 01/02/2014

; 29/09/2013

mov cx, 10

@@: ; 01/02/2014

push cx ; 29/09/2013

; byte ptr [u.ttyn] = tty number (0 to 9)

mov al, 1

call getc

pop cx ; 29/09/2013

; 28/07/2013

; byte ptr [u.ttyn] = tty number

;; 15/04/2014

;;jc error ; 15/05/2013 (COM1 or COM2 serial port error)

;mov ah, 01h ; Test for available key, ZF=1 if none, ZF=0 and

;int 16h ; AX contains next key code if key available.

jnz short @f

; bne 1f / 2nd word of console tty buffer contains number

; / of chars. Is this number non-zero?

;dec cx

;jnz short rtty\_idle

loop rtty\_idle ; 01/02/2014

; 05/10/2013

mov ah, byte ptr [u.ttyn]

; 29/09/2013

call sleep

; jsr r0,canon; ttych / if 0, call 'canon' to get a line

; / (120 chars.)

;byte ptr [u.ttyn] = tty number (0 to 9)

jmp short rtty\_nc ; 01/02/2014

rtty\_idle:

; 16/07/2013

;; mov cx, word ptr [s.idlet]+2 ;; 29/07/2013

call idle

; 29/09/2013

jmp short @b ; 01/02/2014

;1:

;rtty\_nc:

;mov al, 1

;call getc

;mov ah, 01h ; Test for available key, ZF=1 if none, ZF=0 and

;int 16h ; AX contains next key code if key available.

;jz short ret\_

; tst 2(r5) / is the number of characters zero

; beq ret1 / yes, return to caller via 'ret1'

; movb \*4(r5),r1 / no, put character in r1

; inc 4(r5) / 3rd word of console tty buffer points to byte which

; / contains the next char.

; dec 2(r5) / decrement the character count

@@:

xor al, al

call getc

;; 23/07/0014

;;jc error ; 15/05/2013 (COM1 or COM2 serial port error)

; AL = ascii code of the character

;xor ah, ah

;int 16h

;

call passc

; jsr r0,passc / move the character to core (user)

; 19/06/2014

jnz short rtty\_nc

; 23/07/2014

;jmp short ret\_

pop ax

retn

;ret1:

; jmp ret / return to caller via 'ret'

rcvt: ; < receive/read character from tty >

; 06/12/2013 (major modification: p.ttyc, u.ttyp)

; 28/07/2013 al = tty number (ah -> al)

; 16/07/2013 rttys

; 21/05/2013 owner checking for COM/serial ports

; 15/05/2013

;

; Retro UNIX 8086 v1 modification !

;

; In original UNIX v1, 'rcvt' routine

; (exactly different than this one)

; was in 'u9.s' file.

;

sub al, 10

; AL = tty number (0 to 9), (COM1=8, COM2=9)

; 16/07/2013

; 21/05/2013

jmp short rttys

;rppt: / read paper tape

; jsr r0,pptic / gets next character in clist for ppt input and

; / places

; br ret / it in r1; if there 1s no problem with reader, it

; / also enables read bit in prs

; jsr r0,passc / place character in users buffer area

; br rppt

rmem: ; / transfer characters from memory to a user area of core

mov si, word ptr [u.fofp]

@@:

mov bx, word ptr [SI]

; mov \*u.fofp,r1 / save file offset which points to the char

; / to be transferred to user

inc word ptr [BX] ; 16/07/2013

; inc \*u.fofp / increment file offset to point to 'next'

; / char in memory file

mov al, byte ptr [BX]

; movb (r1),r1 / get character from memory file,

; / put it in r1

call passc ; jsr r0,passc / move this character to

; / the next byte of the users core area

; 20/09/2013

;jmp short @b

; br rmem / continue

jnz short @b

;

ret\_:

pop ax

retn

rlpr:

;1:

;rcrd:

jmp error

;jmp error / see 'error' routine

dskr:

; 03/08/2013

; 31/07/2013

; 26/07/2013 (namei\_r check)

push ax ; 26/04/2013

; mov (sp),r1 / i-number in r1

; AX = i-number

call iget

; jsr r0,iget / get i-node (r1) into i-node section of core

mov dx, word ptr [i.size\_]

; mov i.size,r2 / file size in bytes in r2

mov bx, word ptr [u.fofp]

sub dx, word ptr [BX]

; sub \*u.fofp,r2 / subtract file offset

jna short ret\_

; blos ret

cmp dx, word ptr [u.count]

; cmp r2,u.count / are enough bytes left in file

; / to carry out read

jnb short dskr\_1

; bhis 1f

mov word ptr [u.count], dx

; mov r2,u.count / no, just read to end of file

dskr\_1: ; 1:

; AX = i-number

call mget

; jsr r0,mget / returns physical block number of block

; / in file where offset points

; AX = physical block number

call dskrd

; jsr r0,dskrd / read in block, r5 points to

; / 1st word of data in buffer

; BX (r5) = system (I/O) buffer address

call sioreg

; jsr r0,sioreg

xchg si, di

; DI = file (user data) offset

; SI = sector (I/O) buffer offset

; CX = byte count

; 03/08/2013

cmp byte ptr [namei\_r], 0

;;28/07/2013 namei\_r -> u.namei\_r

; 26/07/2013

;;dec byte ptr [u.namei\_r] ; the caller is 'namei' sign (=1)

jna short dskr\_2 ; zf=0 -> the caller is 'namei'

rep movsb

jmp short dskr\_3

dskr\_2:

;;28/07/2013

; 26/07/2013

;;inc byte ptr [u.namei\_r] ; (=0)

mov ax, word ptr [u.segmnt] ; Retro Unix 8086 v1 feature only !

mov es, ax ; Retro Unix 8086 v1 feature: ES = user segment !

; 2:

rep movsb

; movb (r2)+,(r1)+ / move data from buffer into working core

; / starting at u.base

; dec r3

; bne 2b / branch until proper number of bytes are transferred

mov ax, ds

mov es, ax

dskr\_3:

; 03/08/2013

pop ax

cmp word ptr [u.count], cx ; 0

; tst u.count / all bytes read off disk

; bne dskr

ja short dskr

mov byte ptr [namei\_r], cl ; 0

retn

;jna short ret\_

; br ret

;pop ax ; 26/04/2013 (i-node number)

;jmp short dskr

passc:

mov bx, word ptr [u.segmnt] ; Retro Unix 8086 v1 feature only !

mov es, bx ; Retro Unix 8086 v1 feature: ES = user segment !

mov bx, word ptr [u.base]

mov byte ptr ES:[BX], al

; movb r1,\*u.base / move a character to the next byte of the

; / users buffer

mov bx, ds ; Retro Unix 8086 v1 feature: DS = system segment !

mov es, bx ; Retro Unix 8086 v1 feature: ES = system segment !

inc word ptr [u.base]

; inc u.base / increment the pointer to point to

; / the next byte in users buffer

inc word ptr [u.nread]

; inc u.nread / increment the number of bytes read

dec word ptr [u.count]

; dec u.count / decrement the number of bytes to be read

; 20/09/2013 (;;)

retn

;;jnz short @f

; bne 1f / any more bytes to read?; yes, branch

;;pop ax

;; ; mov (sp)+,r0 / no, do a non-local return to the caller of

; / 'readi' by:

;;ret\_: ;/ (1) pop the return address off the stack into r0

;; pop ax

; mov (sp)+,r1 / (2) pop the i-number off the stack into r1

;;@@: ;1:

; clr \*$ps / clear processor status

;; retn

; rts r0 / return to address currently on top of stack

writei:

; 31/07/2013

; 15/05/2013 COM1, COM2 (serial ports) modification

; 26/04/2013

; 14/03/2013 wslot, sioreg

; 12/03/2013

; Write data to file with inode number in R1

;

; INPUTS ->

; r1 - inode number

; u.count - byte count to be written

; u.base - points to user buffer

; u.fofp - points to word with current file offset

; OUTPUTS ->

; u.count - cleared

; u.nread - accumulates total bytes passed back

; ((AX = R1))

; (Retro UNIX Prototype : 18/11/2012 - 11/11/2012, UNIXCOPY.ASM)

; ((Modified registers: DX, BX, CX, SI, DI, BP))

xor cx, cx

mov word ptr [u.nread], cx ; 0

; clr u.nread / clear the number of bytes transmitted during

; / read or write calls

cmp word ptr [u.count], cx

; ; tst u.count / test the byte count specified by the user

ja short @f ; 1f

; bgt 1f / any bytes to output; yes, branch

retn

; ; rts r0 / no, return - no writing to do

@@: ;1:

; mov r1 ,-(sp) / save the i-node number on the stack

cmp ax, 40

; cmp r1,$40.

; / does the i-node number indicate a special file?

ja dskw

; bgt dskw / no, branch to standard file output

;

push ax ; because subroutines will jump to 'ret\_'

mov bx, ax

shl bx, 1

; asl r1 / yes, calculate the index into the special file

add bx, offset @f - 2

jmp word ptr [BX]

; jmp \*1f-2(r1)

; / jump table and jump to the appropriate routine

@@: ;1:

dw offset wtty ; tty, AX = 1 (runix)

;wtty / tty; r1=2

;wppt / ppt; r1=4

dw offset wmem ; mem, AX = 2 (runix)

;wmem / mem; r1=6

;wrf0 / rf0

;wrk0 / rk0

;wtap / tap0

;wtap / tap1

;wtap / tap2

;wtap / tap3

;wtap / tap4

;wtap / tap5

;wtap / tap6

;wtap / tap7

dw offset wfd ; fd0, AX = 3 (runix only)

dw offset wfd ; fd1, AX = 4 (runix only)

dw offset whd ; hd0, AX = 5 (runix only)

dw offset whd ; hd1, AX = 6 (runix only)

dw offset whd ; hd2, AX = 7 (runix only)

dw offset whd ; hd3, AX = 8 (runix only)

dw offset wlpr ; lpr, AX = 9 (runix)

dw offset xmtt ; tty0, AX = 10 (runix)

;xmtt / tty0

dw offset xmtt ; tty1, AX = 11 (runix)

;xmtt / tty1

dw offset xmtt ; tty2, AX = 12 (runix)

;xmtt / tty2

dw offset xmtt ; tty3, AX = 13 (runix)

;xmtt / tty3

dw offset xmtt ; tty4, AX = 14 (runix)

;xmtt / tty4

dw offset xmtt ; tty5, AX = 15 (runix)

;xmtt / tty5

dw offset xmtt ; tty6, AX = 16 (runix)

;xmtt / tty6

dw offset xmtt ; tty7, AX = 17 (runix)

;xmtt / tty7

dw offset xmtt ; COM1, AX = 18 (runix only)

; / wlpr / lpr

dw offset xmtt ; COM2, AX = 19 (runix only)

wtty: ; write to console tty (write to screen)

; 07/07/2014

; 27/06/2014

; 19/06/2014

; 02/06/2014

; 26/05/2014 (putc\_eot, putc\_n, sleep bugfix)

; 15/04/2014 ('putc' error return modification)

; 14/04/2014 (serial port modification)

; 13/01/2014

; 06/12/2013 (major modification: p.ttyc, u.ttyp)

; 10/10/2013

; 05/10/2013

; 20/09/2013 (tty write lock)

; 13/09/2013

; 26/08/2013

; 14/08/2013

; 28/07/2013 u.ttyn

; 21/05/2013 owner checking

; 15/05/2013 'mov ah, byte ptr [ptty]', wtty\_nc

; 14/05/2013 'putc' modifications instead of INT 10h

; 12/03/2013

; Console tty output is on on current video page

; Console tty character output procedure is changed here

; acconding to IBM PC compatible ROM BIOS video (text mode) functions.

;

; 06/12/2013

mov bl, byte ptr [u.uno] ; process number

xor bh, bh

mov ah, byte ptr [BX]+p.ttyc-1 ; current/console tty

mov al, ah ; 07/07/2014

wttys: ;

; 10/10/2013

mov byte ptr [u.ttyn], ah

; 06/12/2013

;; 13/01/2014

;;cmp ah, 7

;;ja short @f

;mov al, ah

inc al

mov byte ptr [u.ttyp]+1, al ; tty number + 1

;;@@: ; 26/08/2013

wtty\_nc: ; 15/05/2013

; AH = [u.ttyn] = tty number ; 28/07/2013

call cpass

; jsr r0,cpass / get next character from user buffer area; if

; / none go to return address in syswrite

; tst r1 / is character = null

; beq wtty / yes, get next character

; 10/10/2013

jz short wret

;1 :

;mov $240,\*$ps / no, set processor priority to five

;cmpb cc+1,$20. / is character count for console tty greater

; / than 20

;bhis 2f / yes; branch to put process to sleep

; 27/06/2014

@@:

; AH = tty number

; AL = ASCII code of the character

; 15/04/2014

push ax

call putc ; 14/05/2013

jnc short @f

; 02/06/2014

mov ah, byte ptr [u.ttyn]

call sleep

pop ax

jmp short @b

; jc error ; 15/05/2013 (COM1 or COM2 serial port error)

; jsr r0,putc; 1 / find place in freelist to assign to

; / console tty and

; br 2f / place character in list; if none available

; / branch to put process to sleep

; jsr r0,startty / attempt to output character on tty

@@:

; 15/04/2014

pop ax

jmp short wtty\_nc

; br wtty

wret: ; 10/10/2013

pop ax

retn

;2:

;mov r1,-(sp) / place character on stack

;jsr r0,sleep; 1 / put process to sleep

;mov (sp)+,r1 / remove character from stack

;br 1b / try again to place character in clist and output

xmtt: ; < send/write character to tty >

; 06/12/2013 (major modification: p.ttyc, u.ttyp)

; 10/10/2013

; 14/08/2013

; 28/07/2013

; 21/05/2013 owner checking for COM/serial ports

; 15/05/2013

;

; Retro UNIX 8086 v1 modification !

;

; In original UNIX v1, 'xmtt' routine

; (exactly different than this one)

; was in 'u9.s' file.

;

sub al, 10

; AL = tty number (0 to 9), (COM1=8, COM2=9)

; 10/10/2013

mov ah, al

; 28/07/2013

jmp short wttys

;wppt:

; jsr r0,cpass / get next character from user buffer area,

; / if none return to writei's calling routine

; jsr r0,pptoc / output character on ppt

; br wppt

wlpr:

jmp error ; ... Printing procedure will be located here ...

;/ jsr r0,cpass

;/ cmp r0,$'a

;/ blo 1f

;/ cmp r1,$'z

;/ bhi 1f

;/ sub $40,r1

;/1:

;/ jsr r0,lptoc

;/ br wlpr

; br rmem / continue

wmem: ; / transfer characters from a user area of core to memory file

mov si, word ptr [u.fofp]

@@:

call cpass

; jsr r0,cpass / get next character from users area of

; / core and put it in r1

; mov r1,-(sp) / put character on the stack

; 20/09/2013

jz short wret ; @f

mov bx, word ptr [SI]

; mov \*u.fofp,r1 / save file offset in r1

inc word ptr [BX] ; 16/07/2013

; inc \*u.fofp / increment file offset to point to next

; / available location in file

mov byte ptr [BX], al

; movb (sp)+,(r1) / pop char off stack, put in memory loc

; / assigned to it

jmp short @b

; br wmem / continue

;1:

;jmp error / ?

;@@:

; ; 20/09/2013

; pop ax

; retn

dskw: ; / write routine for non-special files

; 20/09/2013

; 03/08/2013

; 01/08/2013 (mkdir\_w check)

push ax ; 26/04/2013

; mov (sp),r1 / get an i-node number from the stack into r1

; AX = inode number

call iget

; jsr r0,iget / write i-node out (if modified),

; / read i-node 'r1' into i-node area of core

mov bx, word ptr [u.fofp]

mov dx, word ptr [BX]

; mov \*u.fofp,r2 / put the file offset [(u.off) or the offset

; / in the fsp entry for this file] in r2

add dx, word ptr [u.count]

; add u.count,r2 / no. of bytes to be written

; / + file offset is put in r2

cmp dx, word ptr [i.size\_]

; cmp r2,i.size / is this greater than the present size of

; / the file?

jna short dskw\_1

; blos 1f / no, branch

mov word ptr [i.size\_], dx

; mov r2,i.size / yes, increase the f11e size to

; / file offset + no. of data bytes

call setimod

; jsr r0,setimod / set imod=1 (i.e., core inode has been

; / modified), stuff time of modification into

; / core image of i-node

dskw\_1: ; 1:

call mget

; AX = Block number

; jsr r0,mget / get the block no. in which to write

; / the next data byte

mov bx, word ptr [u.fofp]

mov dx, word ptr [BX]

and dx, 1FFh

; bit \*u.fofp,$777 / test the lower 9 bits of the file offset

jnz short dskw\_2

; bne 2f / if its non-zero, branch; if zero, file offset = 0,

; / 512, 1024,...(i.e., start of new block)

cmp word ptr [u.count], 512

; cmp u.count,$512. / if zero, is there enough data to fill

; / an entire block? (i.e., no. of

jnb short dskw\_3

; bhis 3f / bytes to be written greater than 512.?

; / Yes, branch. Don't have to read block

dskw\_2: ; 2: / in as no past info. is to be saved (the entire block will be

; / overwritten).

call dskrd

; jsr r0,dskrd / no, must retain old info..

; / Hence, read block 'r1' into an I/O buffer

dskw\_3: ; 3:

; AX (r1) = block/sector number

call wslot

; jsr r0,wslot / set write and inhibit bits in I/O queue,

; / proc. status=0, r5 points to 1st word of data

; BX (r5) = system (I/O) buffer address

call sioreg

; jsr r0,sioreg / r3 = no. of bytes of data,

; / r1 = address of data, r2 points to location

; / in buffer in which to start writing data

; SI = file (user data) offset

; DI = sector (I/O) buffer offset

; CX = byte count

;

; 03/08/2013

; 01/08/2013

cmp byte ptr [mkdir\_w], 0

jna short dskw\_4 ; zf=0 -> the caller is 'mkdir'

rep movsb

jmp short dskw\_5

dskw\_4:

mov ax, word ptr [u.segmnt] ; Retro Unix 8086 v1 feature only !

mov ds, ax ; Retro Unix 8086 v1 feature: ES = user segment !

; 2:

rep movsb

; movb (r1 )+,(r2)+

; / transfer a byte of data to the I/O buffer

; dec r3 / decrement no. of bytes to be written

; bne 2b / have all bytes been transferred? No, branch

mov ax, cs ; Retro Unix 8086 v1 feature: CS = system segment !

mov ds, ax ; Retro Unix 8086 v1 feature: DS = system segment !

dskw\_5:

call dskwr

; jsr r0,dskwr / yes, write the block and the i-node

cmp word ptr [u.count], 0

; tst u.count / any more data to write?

ja short dskw\_1

; bne 1b / yes, branch

; 03/08/2013

mov byte ptr [mkdir\_w], 0

; 20/09/2013 (;;)

pop ax

retn

;;jmp short dskw\_ret

; jmp ret / no, return to the caller via 'ret'

cpass: ; / get next character from user area of core and put it in r1

cmp word ptr [u.count], 0 ; 14/08/2013

; tst u.count / have all the characters been transferred

; / (i.e., u.count, # of chars. left

jna short @f

; beq 1f / to be transferred = 0?) yes, branch

dec word ptr [u.count]

; dec u.count / no, decrement u.count

;

mov bx, word ptr [u.segmnt] ; Retro Unix 8086 v1 feature only !

mov es, bx ; Retro Unix 8086 v1 feature: ES = user segment !

;

mov bx, word ptr [u.base]

mov al, byte ptr ES:[BX] ; Runix v1: get data from user segment!

; movb \*u.base,r1 / take the character pointed to

; / by u.base and put it in r1

mov bx, ds ; Retro Unix 8086 v1 feature: DS = system segment !

mov es, bx ; Retro Unix 8086 v1 feature: ES = system segment !

;

inc word ptr [u.nread]

; inc u.nread / increment no. of bytes transferred

inc word ptr [u.base]

; inc u.base / increment the buffer address to point to the

@@: ; 20/09/2013 (;;)

retn

; rts r0 / next byte

;;@@: ; 1:

;; pop ax

; mov (sp)+,r0

; / put return address of calling routine into r0

;;dskw\_ret:

;; pop ax

; mov (sp)+,r1 / i-number in r1

;; retn

; rts r0 / non-local return

sioreg:

; 22/07/2013

; 14/03/2013 bx -> si, ax input -> bx input

; 12/03/2013

; INPUTS ->

; BX = system buffer (data) address (r5)

; OUTPUTS ->

; SI = user data offset (r1)

; DI = system (I/O) buffer offset (r2)

; CX = byte count (r3)

; ((Modified registers: AX)) ; 22/07/2013

mov si, word ptr [u.fofp]

mov di, word ptr [SI]

; mov \*u.fofp,r2 / file offset (in bytes) is moved to r2

mov cx, di

; mov r2,r3 / and also to r3

or cx, 0FE00h

; bis $177000,r3 / set bits 9,...,15 of file offset in r3

and di, 1FFh

; bic $!777,r2 / calculate file offset mod 512.

add di, bx ; BX = system buffer (data) address

; add r5,r2 / r2 now points to 1st byte in system buffer

; / where data is to be placed

mov ax, word ptr [u.base] ; 22/07/2013

; mov u.base,r1 / address of data is in r1

neg cx

; neg r3 / 512 - file offset (mod512.) in r3

; / (i.e., the no. of free bytes in the file block)

cmp cx, word ptr [u.count]

; cmp r3,u.count / compare this with the no. of data bytes

; / to be written to the file

jna short @f

; blos 2f / if less than branch. Use the no. of free bytes

; / in the file block as the number to be written

mov cx, word ptr [u.count]

; mov u.count,r3 / if greater than, use the no. of data

; / bytes as the number to be written

@@: ; 2:

add word ptr [u.nread], cx

; add r3,u.nread / r3 + number of bytes xmitted

; / during write is put into u.nread

sub word ptr [u.count], cx

; sub r3,u.count / u.count = no. of bytes that still

; / must be written or read

add word ptr [u.base], cx

; add r3,u.base / u.base points to the 1st of the remaining

; / data bytes

add word ptr [SI], cx

; add r3,\*u.fofp / new file offset = number of bytes done

; / + old file offset

mov si, ax ; 22/07/2013

retn

; rts r0