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

;

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

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

; U2.ASM (include u2.asm) //// UNIX v1 -> u2.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: 24/03/2014 ] ;;; 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>

;

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

; 24/03/2014 sysbreak

; 12/01/2014 fclose

; 06/12/2013 sysexec

; 19/11/2013 sysbreak

; 18/11/2013 getf (getf1)

; 24/10/2013 sysexec

; 03/09/2013 sysexec (u.intr, u.quit reset -> enabled)

; 05/08/2013 fclose, seektell

; 02/08/2013 maknod, (u.uid -> u.uid\_)

; 01/08/2013 mkdir

; 31/07/2013 u.namei\_r -> namei\_r, maknod

; 30/07/2013 fclose

; 28/07/2013 namei (u.namei\_r)

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

; 25/07/2013 sysexec (arguments)

; 24/07/2013 sysexec

; 22/07/2013 sysexec, namei

; 18/07/2013 sysexec, namei

; 17/07/2013 maknod (inode->i)

; 09/07/2013 namei (rootdir)

; 07/07/2013 sysseek, systell, sysintr, sysquit, syssetuid, sysgetuid

; 07/07/2013 syschmod, syschown

; 20/06/2013 syschmod, syschown, systime, sysstime, sysbreak

; 19/06/2013 syslink, sysunlink, sysstat, sysfstat, syschdir

; 04/06/2013 sysexec

; 03/06/2013 sysexec

; 27/05/2013 namei (stc)

; 23/05/2013 getf1

; 02/05/2013 maknod

; 29/04/2013 mkdir

; 25/04/2013 anyi

; 24/04/2013 namei

; 19/04/2013 fclose

; 11/03/2013

syslink:

; 19/06/2013

; 'syslink' is given two arguments, name 1 and name 2.

; name 1 is a file that already exists. name 2 is the name

; given to the entry that will go in the current directory.

; name2 will then be a link to the name 1 file. The i-number

; in the name 2 entry of current directory is the same

; i-number for the name 1 file.

;

; Calling sequence:

; syslink; name 1; name 2

; Arguments:

; name 1 - file name to which link will be created.

; name 2 - name of entry in current directory that

; links to name 1.

; Inputs: -

; Outputs: -

; ...............................................................

;

; Retro UNIX 8086 v1 modification:

; 'syslink' system call has two arguments; so,

; Retro UNIX 8086 v1 argument transfer method 2 is used

; to get syslink system call arguments from the user;

; \* 1st argument, name 1 is pointed to by BX register

; \* 2nd argument, name 2 is pointed to by CX register

; NOTE: Retro UNIX 8086 v1 'arg2' routine gets these

; arguments which were in these registers;

; but, it returns by putting the 1st argument

; in 'u.namep' and the 2nd argument

; on top of stack.

;

; / name1, name2

;call arg2

; jsr r0,arg2 / u.namep has 1st arg u.off has 2nd

mov word ptr [u.namep], bx

push cx

call namei

; jsr r0,namei / find the i-number associated with

; / the 1st path name

;and ax, ax

;jz error ; File not found

jc error

; br error9 / cannot be found

call iget

; jsr r0,iget / get the i-node into core

pop word ptr [u.namep] ; cx

; mov (sp)+,u.namep / u.namep points to 2nd name

push ax

; mov r1,-(sp) / put i-number of name1 on the stack

; / (a link to this file is to be created)

push word ptr [cdev]

; mov cdev,-(sp) / put i-nodes device on the stack

call isdir

; jsr r0,isdir / is it a directory

call namei

; jsr r0,namei / no, get i-number of name2

jnc error

; br .+4 / not found

; / so r1 = i-number of current directory

; / ii = i-number of current directory

; br error9 / file already exists., error

pop cx

cmp cx, word ptr [cdev]

; cmp (sp)+,cdev / u.dirp now points to

; / end of current directory

jne error

; bne error9

pop ax

push ax

mov word ptr [u.dirbuf], ax

; mov (sp),u.dirbuf / i-number of name1 into u.dirbuf

call mkdir

; jsr r0,mkdir / make directory entry for name2

; / in current directory

pop ax

; mov (sp)+,r1 / r1 has i-number of name1

call iget

; jsr r0,iget / get i-node into core

inc byte ptr [i.nlks]

; incb i.nlks / add 1 to its number of links

call setimod

; jsr r0,setimod / set the i-node modified flag

jmp sysret

isdir:

; 02/08/2013

; 04/05/2013

; 'isdir' check to see if the i-node whose i-number is in r1

; is a directory. If it is, an error occurs, because 'isdir'

; called by syslink and sysunlink to make sure directories

; are not linked. If the user is the super user (u.uid=0),

; 'isdir' does not bother checking. The current i-node

; is not disturbed.

;

; INPUTS ->

; r1 - contains the i-number whose i-node is being checked.

; u.uid - user id

; OUTPUTS ->

; r1 - contains current i-number upon exit

; (current i-node back in core)

;

; ((AX = R1))

;

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

;

; / if the i-node whose i-number is in r1 is a directory

; / there is an error unless super user made the call

cmp byte ptr [u.uid\_], 0

; tstb u.uid / super user

jna short @f

; beq 1f / yes, don't care

push word ptr [ii]

; mov ii,-(sp) / put current i-number on stack

call iget

; jsr r0,iget / get i-node into core (i-number in r1)

test word ptr [i.flgs], 4000h ; Bit 14 : Directory flag

; bit $40000,i.flgs / is it a directory

jnz error

; bne error9 / yes, error

pop ax

; mov (sp)+,r1 / no, put current i-number in r1 (ii)

call iget

; jsr r0,iget / get it back in

@@: ; 1:

retn

; rts r0

sysunlink:

; 19/06/2013

; 'sysunlink' removes the entry for the file pointed to by

; name from its directory. If this entry was the last link

; to the file, the contents of the file are freed and the

; file is destroyed. If, however, the file was open in any

; process, the actual destruction is delayed until it is

; closed, even though the directory entry has disappeared.

;

; The error bit (e-bit) is set to indicate that the file

; does not exist or that its directory can not be written.

; Write permission is not required on the file itself.

; It is also illegal to unlink a directory (except for

; the superuser).

;

; Calling sequence:

; sysunlink; name

; Arguments:

; name - name of directory entry to be removed

; Inputs: -

; Outputs: -

; ...............................................................

; Retro UNIX 8086 v1 modification:

; The user/application program puts address of the name

; in BX register as 'sysunlink' system call argument.

; / name - remove link name

;;mov ax, 1 ; one/single argument, put argument in BX

;;call arg

;mov bp, word ptr [u.sp\_] ; points to user's BP register

;add bp, 6 ; bx now points to BX on stack

;mov bx, word ptr [BP]

mov word ptr [u.namep], bx

;jsr r0,arg; u.namep / u.namep points to name

call namei

; jsr r0,namei / find the i-number associated

; / with the path name

jc error

; br error9 / not found

push ax

; mov r1,-(sp) / put its i-number on the stack

call isdir

; jsr r0,isdir / is it a directory

xor ax, ax

mov word ptr [u.dirbuf], ax ; 0

; clr u.dirbuf / no, clear the location that will

; / get written into the i-number portion

; / of the entry

sub word ptr [u.off], 10

; sub $10.,u.off / move u.off back 1 directory entry

call wdir

; jsr r0,wdir / free the directory entry

pop ax

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

call iget

; jsr r0,iget / get i-node

call setimod

; jsr r0,setimod / set modified flag

dec byte ptr [i.nlks]

; decb i.nlks / decrement the number of links

jnz sysret

; bgt sysret9 / if this was not the last link

; / to file return

; AX = r1 = i-number

call anyi

; jsr r0,anyi / if it was, see if anyone has it open.

; / Then free contents of file and destroy it.

jmp sysret

; br sysret9

mkdir:

; 01/08/2013

; 29/04/2013

; 'mkdir' makes a directory entry from the name pointed to

; by u.namep into the current directory.

;

; INPUTS ->

; u.namep - points to a file name

; that is about to be a directory entry.

; ii - current directory's i-number.

; OUTPUTS ->

; u.dirbuf+2 - u.dirbuf+10 - contains file name.

; u.off - points to entry to be filled

; in the current directory

; u.base - points to start of u.dirbuf.

; r1 - contains i-number of current directory

;

; ((AX = R1)) output

;

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

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

;

mov cx, 4

xor ax, ax

mov di, offset u.dirbuf+2

mov si, di

rep stosw

; jsr r0,copyz; u.dirbuf+2; u.dirbuf+10. / clear this

mov di, si

mov si, word ptr [u.namep]

; mov u.namep,r2 / r2 points to name of directory entry

; mov $u.dirbuf+2,r3 / r3 points to u.dirbuf+2

mkdir\_1: ; 1:

; / put characters in the directory name in u.dirbuf+2 - u.dirbuf+10

; 01/08/2013

push cs ; push ds

mov ax, word ptr [u.segmnt]

mov ds, ax

@@:

lodsb

; movb (r2)+,r1 / move character in name to r1

and al, al

jz short mkdir\_2

; beq 1f / if null, done

cmp al, '/'

; cmp r1,$'/ / is it a "/"?

je short @f

;je error

; beq error9 / yes, error

cmp di, offset u.dirbuf+10

; cmp r3,$u.dirbuf+10. / have we reached the last slot for

; / a char?

je short @b

;je short mkdir\_1

; beq 1b / yes, go back

stosb

; movb r1,(r3)+ / no, put the char in the u.dirbuf

; 01/08/2013

jmp short @b

; jmp short mkdir\_1

; br 1b / get next char

@@:

; 01/08/2013

pop ds

jmp error

mkdir\_2: ; 1:

; 01/08/2013

pop ds

;

mov ax, word ptr [u.dirp]

mov word ptr [u.off], ax

; mov u.dirp,u.off / pointer to empty current directory

; / slot to u.off

wdir: ; 29/04/2013

mov word ptr [u.base], offset u.dirbuf

; mov $u.dirbuf,u.base / u.base points to created file name

mov word ptr [u.count], 10

; mov $10.,u.count / u.count = 10

mov ax, word ptr [ii]

; mov ii,r1 / r1 has i-number of current directory

mov dl, 1 ; owner flag mask ; RETRO UNIX 8086 v1 modification !

call access

; jsr r0,access; 1 / get i-node and set its file up

; / for writing

; AX = i-number of current directory

; 01/08/2013

inc byte ptr [mkdir\_w] ; the caller is 'mkdir' sign

call writei

; jsr r0,writei / write into directory

retn

; rts r0

sysexec:

; 06/12/2013

; 24/10/2013, 22/09/2013, 03/09/2013

; 02/08/2013, 25/07/2013, 24/07/2013

; 22/07/2013, 18/07/2013, 03/06/2013

; 'sysexec' initiates execution of a file whose path name if

; pointed to by 'name' in the sysexec call.

; 'ssysexec' performs the following operations:

; 1. obtains i-number of file to be executed via 'namei'.

; 2. obtains i-node of file to be exceuted via 'iget'.

; 3. sets trap vectors to system routines.

; 4. loads arguments to be passed to executing file into

; highest locations of user's core

; 5. puts pointers to arguments in locations immediately

; following arguments.

; 6. saves number of arguments in next location.

; 7. intializes user's stack area so that all registers

; will be zeroed and the PS is cleared and the PC set

; to core when 'sysret' restores registers

; and does an rti.

; 8. inializes u.r0 and u.sp

; 9. zeros user's core down to u.r0

; 10. reads executable file from storage device into core

; starting at location 'core'.

; 11. sets u.break to point to end of user's code with

; data area appended.

; 12. calls 'sysret' which returns control at location

; 'core' via 'rti' instruction.

;

; Calling sequence:

; sysexec; namep; argp

; Arguments:

; namep - points to pathname of file to be executed

; argp - address of table of argument pointers

; argp1... argpn - table of argument pointers

; argp1:<...0> ... argpn:<...0> - argument strings

; Inputs: (arguments)

; Outputs: -

; ...............................................................

;

; Retro UNIX 8086 v1 modification:

; user/application segment and system/kernel segment

; are different and sysenter/sysret/sysrele routines

; are different (user's registers are saved to

; and then restored from system's stack.)

;

; NOTE: Retro UNIX 8086 v1 'arg2' routine gets these

; arguments which were in these registers;

; but, it returns by putting the 1st argument

; in 'u.namep' and the 2nd argument

; on top of stack. (1st argument is offset of the

; file/path name in the user's program segment.)

;call arg2

; \* name - 'u.namep' points to address of file/path name

; in the user's program segment ('u.segmnt')

; with offset in BX register (as sysopen argument 1).

; \* argp - sysexec argument 2 is in CX register

; which is on top of stack.

;

; jsr r0,arg2 / arg0 in u.namep,arg1 on top of stack

mov word ptr [u.namep], bx ; argument 1

push cx ; argument 2

call namei

; jsr r0,namei / namei returns i-number of file

; / named in sysexec call in r1

jc error

; br error9

call iget

; jsr r0,iget / get i-node for file to be executed

test word ptr [i.flgs], 10h

; bit $20,i.flgs / is file executable

jz error

; beq error9

call iopen

; jsr r0,iopen / gets i-node for file with i-number

; / given in r1 (opens file)

; AX = i-number of the file

test word ptr [i.flgs], 20h

; bit $40,i.flgs / test user id on execution bit

jz short sysexec\_1

; beq 1f

cmp byte ptr [u.uid\_], 0 ; 02/08/2013

; tstb u.uid / test user id

jna short sysexec\_1

; beq 1f / super user

mov cl, byte ptr [i.uid]

mov byte ptr [u.uid\_], cl ; 02/08/2013

; movb i.uid,u.uid / put user id of owner of file

; / as process user id

sysexec\_1: ; 1:

; 22/07/2013

call segm\_sw ; User segment switch

; BX = New user segment ; 24/07/2013

;

pop cx

; mov (sp)+,r5 / r5 now contains address of list of

; / pointers to arguments to be passed

; mov $1,u.quit / u.quit determines handling of quits;

; / u.quit = 1 take quit

; mov $1,u.intr / u.intr determines handling of

; / interrupts; u.intr = 1 take interrupt

; mov $rtssym,30 / emt trap vector set to take

; / system routine

; mov $fpsym,\*10 / reserved instruction trap vector

; / set to take system routine

; 24/07/2013

mov sp, sstack ; offset sstack

; mov $sstack,sp / stack space used during swapping

;push cx

; mov r5,-(sp) / save arguments pointer on stack

mov di, ecore

; mov $ecore,r5 / r5 has end of core

;mov bp, core

xor bp, bp ; core = 0

; mov $core,r4 / r4 has start of users core

mov word ptr [u.base], bp

; mov r4,u.base / u.base has start of users core

; 24/07/2013

mov es, bx ; new user segment

; If the caller is a user, es = word ptr [u.segmnt]

; If the caller is system (sysexec for '/etc/init')

; es = csgmnt and word ptr [u.segmnt] = cs

mov dx, word ptr [u.segmnt]

mov ds, dx

mov bx, cx

; mov (sp),r2 / move arguments list pointer into r2

sysexec\_2: ; 1:

; AX = i-number of the file (at return of 'iopen' call)

mov dx, word ptr [BX]

and dx, dx

jz short @f

; tst (r2)+ / argument char = "nul"

; bne 1b

inc bx

inc bx

jmp short sysexec\_2

@@:

; tst -(r2) / decrement r2 by 2; r2 has addr of end of

; / argument pointer list

sysexec\_3: ; 1:

; / move arguments to bottom of users core

dec bx

dec bx

;mov si, word ptr [BX]

;; mov -(r2),r3 / (r3) last non zero argument ptr

cmp bx, cx

; cmp r2,(sp) / is r2 = beginning of argument

; / ptr list

jb short sysexec\_6

; blo 1f / branch to 1f when all arguments

; / are moved

mov si, word ptr [BX]

; mov -(r2),r3 / (r3) last non zero argument ptr

sysexec\_4: ; 2:

mov dl, byte ptr [SI]

and dl, dl

; tstb (r3)+

jz short sysexec\_5

inc si

jmp short sysexec\_4

; bne 2b / scan argument for \0 (nul)

sysexec\_5: ; 2:

dec di

mov byte ptr ES:[DI], dl ; 24/07/2013

; movb -(r3),-(r5) / move argument char

; / by char starting at "ecore"

cmp si, word ptr [BX]

; cmp r3,(r2) / moved all characters in

; / this argument

; bhi 2b / branch 2b if not

jna short @f

dec si

mov dl, byte ptr [SI]

jmp short sysexec\_5

@@:

mov word ptr ES:[BP], di ; 24/07/2013

inc bp

inc bp

; mov r5,(r4)+ / move r5 into top of users core;

; / r5 has pointer to nth arg

jmp sysexec\_3

; br 1b / string

sysexec\_6: ; 1:

dec di

dec di ; 24/10/2013

;mov byte ptr ES:[DI], 0 ; 24/07/2013

; clrb -(r5)

shr di, 1

shl di, 1

; bic $1,r5 / make r5 even, r5 points to

; / last word of argument strings

;mov si, core

xor si, si ; core = 0

; mov $core,r2

mov word ptr ES:[DI], si ; 24/07/2013

sysexec\_7: ; 1: / move argument pointers into core following

; / argument strings

cmp si, bp

; cmp r2,r4

jnb short sysexec\_8

; bhis 1f / branch to 1f when all pointers

; / are moved

mov dx, word ptr ES:[SI] ; 25/07/2013

inc si

dec di

inc si

dec di

mov word ptr ES:[DI], dx ; 24/07/2013

; mov (r2)+,-(r5)

jmp short sysexec\_7

; br 1b

sysexec\_8: ; 1:

;sub bp, core ; core = 0

; sub $core,r4 / gives number of arguments \*2

shr bp, 1

; asr r4 / divide r4 by 2 to calculate

; / the number of args stored

dec di

dec di

mov word ptr ES:[DI], bp ; 24/07/2013

; mov r4,-(r5) / save number of arguments ahead

; / of the argument pointers

xor cx, cx

pushf

pop dx

dec di

dec di

; 24/07/2013 (ES:[DI])

mov word ptr ES:[DI], dx ; FLAGS (for 'IRET')

; clr -(r5) / popped into ps when rti in

; / sysrele is executed

mov bx, es ; 24/07/2013

dec di

dec di

mov word ptr ES:[DI], bx ; CS (for 'IRET')

;mov cx, core ; core = 0

dec di

dec di

mov word ptr ES:[DI], cx ; IP (for 'IRET')

; mov $core,-(r5) / popped into pc when rti

; / in sysrele is executed

;mov r5,0f / load second copyz argument

;tst -(r5) / decrement r5

mov bx, cs

mov ds, bx

mov word ptr [u.r0], cx ; ax = 0

mov word ptr [u.usp], di

push di ; user's stack pointer

push cx ; dx = 0

push cx ; cx = 0

push cx ; bx = 0

push cx ; si = 0

push cx ; di = 0

push cx ; bp = 0

mov word ptr [u.sp\_], sp

mov cx, di

; 24/07/2013

xor di, di ; 0

push ax ; i-number

xor ax, ax ; 0

shr cx, 1 ; cx/2 -> word count

; ES = word ptr [u.segmnt] or csgmnt

rep stosw ; clear user's core/memory segment

mov ax, es ; 24/07/2013

mov word ptr [u.segmnt], ax ; 24/07/2013

mov es, bx ; es = ds = cs

pop ax ; i-number

; mov r5,u.r0 /

; sub $16.,r5 / skip 8 words

; mov r5,u.sp / assign user stack pointer value,

; / effectively zeroes all regs

; / when sysrele is executed

; jsr r0,copyz; core; 0:0 / zero user's core

mov word ptr [u.break\_], cx ; 0

; clr u.break

; mov r5,sp / point sp to user's stack

mov word ptr [u.count], 12

; mov $14,u.count

mov word ptr [u.fofp], offset u.off

; mov $u.off,u.fofp

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

; clr u.off / set offset in file to be read to zero

; AX = i-number of the executable file

call readi

; jsr r0,readi / read in first six words of

; / user's file, starting at $core

mov cx, word ptr [u.usp]

; mov sp,r5 / put users stack address in r5

sub cx, core+40 ; 40 bytes will be reserved

; for user stack

; sub $core+40.,r5 / subtract $core +40,

; / from r5 (leaves number of words

; / less 26 available for

; / program in user core

mov word ptr [u.count], cx

; mov r5,u.count /

mov bx, word ptr [u.segmnt]

mov es, bx

;mov bx, core ; 0

xor bx, bx ; 0

cmp word ptr ES:[BX], 0AEBh ; EBh, 0Ah -> jump to +12

; cmp core,$405 / br .+14 is first instruction

; / if file is standard a.out format

jne short sysexec\_9

; bne 1f / branch, if not standard format

add bl, 2

;add cx, word ptr ES:[BX]+2

add cx, word ptr ES:[BX]

; mov core+2,r5 / put 2nd word of users program in r5;

; / number of bytes in program text

mov dx, ds

mov es, dx

sub cx, 12

; sub $14,r5 / subtract 12

cmp cx, word ptr [u.count]

; cmp r5,u.count /

jg short sysexec\_9

; bgt 1f / branch if r5 greater than u.count

mov word ptr [u.count], cx

; mov r5,u.count

push bx

call readi

; jsr r0,readi / read in rest of user's program text

mov bx, word ptr [u.segmnt]

mov es, bx

pop bx

;mov cx, word ptr ES:[BX]+8

add bl, 6 ; 2+6 = 8

mov cx, word ptr ES:[BX]

;

mov bx, ds

mov es, bx

;

mov word ptr [u.nread], cx

; add core+10,u.nread / add size of user data area

; / to u.nread

jmp short sysexec\_10

; br 2f

sysexec\_9: ; 1:

call readi

; jsr r0,readi / read in rest of file

sysexec\_10: ; 2:

mov cx, word ptr [u.nread]

add cx, core+12 ; 18/07/2013

;mov word ptr [u.break\_], cx

; mov u.nread,u.break / set users program break to end of

; / user code

;add word ptr [u.break\_], core+12 ; 12

; add $core+14,u.break / plus data area

mov word ptr [u.break\_], cx ; 18/07/2013

call iclose

; jsr r0,iclose / does nothing

;; mov sp , word ptr [u.sp\_]

; 06/12/2013

xor ax, ax

inc al

mov word ptr [u.intr], ax ; 1 (interrupt/time-out is enabled)

mov word ptr [u.quit], ax ; 1 ('crtl+brk' signal is enabled)

;

jmp sysret

; br sysret3 / return to core image at $core

sysfstat:

; 19/06/2013

; 'sysfstat' is identical to 'sysstat' except that it operates

; on open files instead of files given by name. It puts the

; buffer address on the stack, gets the i-number and

; checks to see if the file is open for reading or writing.

; If the file is open for writing (i-number is negative)

; the i-number is set positive and a branch into 'sysstat'

; is made.

;

; Calling sequence:

; sysfstat; buf

; Arguments:

; buf - buffer address

;

; Inputs: \*u.r0 - file descriptor

; Outputs: buffer is loaded with file information

; ...............................................................

;

; Retro UNIX 8086 v1 modification:

; 'sysfstat' system call has two arguments; so,

; Retro UNIX 8086 v1 argument transfer method 2 is used

; to get sysfstat system call arguments from the user;

; \* 1st argument, file descriptor is in BX register

; \* 2nd argument, buf is pointed to by CX register

; / set status of open file

;call arg2

; jsr r0,arg; u.off / put buffer address in u.off

push cx

; mov u.off,-(sp) / put buffer address on the stack

; mov ax, word ptr [u.r0]

; mov \*u.r0,r1 / put file descriptor in r1

;jsr r0,getf / get the files i-number

; BX = file descriptor (file number)

call getf1

and ax, ax ; i-number of the file

; tst r1 / is it 0?

jz error

; beq error3 / yes, error

cmp ah, 80h

jb short @f

; bgt 1f / if i-number is negative (open for writing)

neg ax

; neg r1 / make it positive, then branch

jmp short @f

; br 1f / to 1f

sysstat:

; 19/06/2013

; 'sysstat' gets the status of a file. Its arguments are the

; name of the file and buffer address. The buffer is 34 bytes

; long and information about the file placed in it.

; sysstat calls 'namei' to get the i-number of the file.

; Then 'iget' is called to get i-node in core. The buffer

; is then loaded and the results are given in the UNIX

; Programmers Manual sysstat (II).

;

; Calling sequence:

; sysstat; name; buf

; Arguments:

; name - points to the name of the file

; buf - address of a 34 bytes buffer

; Inputs: -

; Outputs: buffer is loaded with file information

; ...............................................................

;

; Retro UNIX 8086 v1 modification:

; 'sysstat' system call has two arguments; so,

; Retro UNIX 8086 v1 argument transfer method 2 is used

; to get sysstat system call arguments from the user;

; \* 1st argument, name is pointed to by BX register

; \* 2nd argument, buf is pointed to by CX register

;

; NOTE: Retro UNIX 8086 v1 'arg2' routine gets these

; arguments which were in these registers;

; but, it returns by putting the 1st argument

; in 'u.namep' and the 2nd argument

; on top of stack. (1st argument is offset of the

; file/path name in the user's program segment.)

; / ; name of file; buffer - get files status

;call arg2

; jsr r0,arg2 / get the 2 arguments

mov word ptr [u.namep], bx

push cx

call namei

; jsr r0,namei / get the i-number for the file

jc error

; br error3 / no such file, error

@@: ; 1:

call iget

; jsr r0,iget / get the i-node into core

mov si, word ptr [u.segmnt]

pop di

; mov (sp)+,r3 / move u.off to r3 (points to buffer)

mov es, si

stosw

; mov r1,(r3)+ / put i-number in 1st word of buffer

;mov si, offset inode

mov si, offset i

; mov $inode,r2 / r2 points to i-node

@@: ; 1:

movsw

; mov (r2)+,(r3)+ / move rest of i-node to buffer

cmp si, offset i + 32

; cmp r2,$inode+32 / done?

jne short @b

; bne 1b / no, go back

mov ax, ds

mov es, ax

jmp sysret

; br sysret3 / return through sysret

fclose:

; 12/01/2014

; 05/08/2013, 30/07/2013, 19/04/2013

; Given the file descriptor (index to the u.fp list)

; 'fclose' first gets the i-number of the file via 'getf'.

; If i-node is active (i-number > 0) the entry in

; u.fp list is cleared. If all the processes that opened

; that file close it, then fsp etry is freed and the file

; is closed. If not a return is taken.

; If the file has been deleted while open, 'anyi' is called

; to see anyone else has it open, i.e., see if it is appears

; in another entry in the fsp table. Upon return from 'anyi'

; a check is made to see if the file is special.

;

; INPUTS ->

; r1 - contains the file descriptor (value=0,1,2...)

; u.fp - list of entries in the fsp table

; fsp - table of entries (4 words/entry) of open files.

; OUTPUTS ->

; r1 - contains the same file descriptor

; r2 - contains i-number

;

; ((AX = R1))

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

;

; Retro UNIX 8086 v1 modification : CF = 1

; if i-number of the file is 0. (error)

;

mov dx, ax ; \*\*

push ax ; \*\*\*

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

; / the index to u.fp list)

call getf

; jsr r0,getf / r1 contains i-number,

; / cdev has device =, u.fofp

; / points to 3rd word of fsp entry

cmp ax, 1 ; r1

; tst r1 / is inumber 0?

jb short fclose\_2

; beq 1f / yes, i-node not active so return

; tst (r0)+ / no, jump over error return

mov bx, dx ; \*\*

mov dx, ax ; \*

; mov r1,r2 / move i-number to r2 ;\*

; mov (sp),r1 / restore value of r1 from the stack

; / which is index to u.fp ; \*\*

mov byte ptr [BX]+u.fp, 0 ; 30/07/2013

; clrb u.fp(r1) / clear that entry in the u.fp list

mov bx, word ptr [u.fofp]

; mov u.fofp,r1 / r1 points to 3rd word in fsp entry

@@:

dec byte ptr [BX]+2

; decb 2(r1) / decrement the number of processes

; / that have opened the file

jns short fclose\_2 ; jump if not negative (jump if bit 7 is 0)

; bge 1f / if all processes haven't closed the file, return

push dx ;\*

; mov r2,-(sp) / put r2 on the stack (i-number)

xor ax, ax ; 0

mov word ptr [BX]-4, ax ; 0

; clr -4(r1) / clear 1st word of fsp entry

; 12/1/2014 (removing Retro UNIX 8086 v1 modification, 30/7/2013)

; (returning to original unix v1 code)

mov al, byte ptr [BX]+3

; tstb 3(r1) / has this file been deleted

and al, al

jz short fclose\_1

; beq 2f / no, branch

mov ax, dx ; \*

; mov r2,r1 / yes, put i-number back into r1

; AX = inode number

call anyi

; jsr r0,anyi / free all blocks related to i-number

; / check if file appears in fsp again

fclose\_1: ; 2:

pop ax ; \*

; mov (sp)+,r1 / put i-number back into r1

call iclose ; close if it is special file

; jsr r0,iclose / check to see if its a special file

fclose\_2: ; 1:

pop ax ; \*\*\*

; mov (sp)+,r1 / put index to u.fp back into r1

retn

; rts r0

getf: ; 18/11/2013 (mov ax, bx)

; 19/04/2013

; / get the device number and the i-number of an open file

mov bx, ax

getf1: ;; Calling point from 'rw1' (23/05/2013)

cmp bx, 10

; cmp r1,$10. / user limited to 10 open files

jnb error

; bhis error3 / u.fp is table of users open files,

; / index in fsp table

mov bl, byte ptr [BX]+u.fp

; movb u.fp(r1),r1 / r1 contains number of entry

; / in fsp table

or bl, bl

jnz short @f ; 18/11/2013

;jz short @f

; beq 1f / if its zero return

; 18/11/2013

mov ax, bx ; 0

retn

@@:

shl bx, 1

; asl r1

shl bx, 1

; asl r1 / multiply by 8 to get index into

; / fsp table entry

shl bx, 1

; asl r1

add bx, offset fsp - 4

; add $fsp-4,r1 / r1 is pointing at the 3rd word

; / in the fsp entry

mov word ptr [u.fofp], bx

; mov r1,u.fofp / save address of 3rd word

; / in fsp entry in u.fofp

dec bx

dec bx

mov ax, word ptr [BX]

;mov byte ptr [cdev], al ; ;;Retro UNIX 8086 v1 !

mov word ptr [cdev], ax ; ;;in fact (!)

; ;;dev number is in 1 byte

; mov -(r1),cdev / remove the device number cdev

dec bx

dec bx

mov ax, word ptr [BX]

; mov -(r1),r1 / and the i-number r1

;@@: ; 1:

retn

; rts r0

namei:

; 31/07/2013

; 28/07/2013

; 26/07/2013 (namei\_r)

; 22/07/2013

; 18/07/2013

; 09/07/2013 mov ax, word ptr [rootdir]

; 27/05/2013 (cf=1 return for indicating 'file not found')

; 24/04/2013

; 'namei' takes a file path name and returns i-number of

; the file in the current directory or the root directory

; (if the first character of the pathname is '/').

;

; INPUTS ->

; u.namep - points to a file path name

; u.cdir - i-number of users directory

; u.cdev - device number on which user directory resides

; OUTPUTS ->

; r1 - i-number of file

; cdev

; u.dirbuf - points to directory entry where a match

; occurs in the search for file path name.

; If no match u.dirb points to the end of

; the directory and r1 = i-number of the current

; directory.

; ((AX = R1))

;

; (Retro UNIX Prototype : 07/10/2012 - 05/01/2013, UNIXCOPY.ASM)

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

;

;;push es ; Retro UNIX 8086 v1 Feature only !

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

mov es, ax ; Retro UNIX 8086 v1 Feature only !

mov ax, word ptr [u.cdir]

; mov u.cdir,r1 / put the i-number of current directory

; / in r1

mov dx, word ptr [u.cdrv]

mov word ptr [cdev], dx ; NOTE: Retro UNIX 8086 v1

; device/drive number is in 1 byte,

; not in 1 word!

; mov u.cdev,cdev / device number for users directory

; / into cdev

xor dx, dx ; 18/07/2013

mov si, word ptr [u.namep]

cmp byte ptr ES:[SI], '/'

; cmpb \*u.namep,$'/ / is first char in file name a /

jne short namei\_1

; bne 1f

inc si ; go to next char

mov word ptr [u.namep], si

; inc u.namep / go to next char

mov ax, word ptr [rootdir] ; 09/07/2013 (mov ax, rootdir)

; mov rootdir,r1 / put i-number of rootdirectory in r1

;xor dx, dx

mov word ptr [cdev], dx

; clr cdev / clear device number

namei\_1: ; 1:

;; 18/07/2013

mov dl, byte ptr ES:[SI]

mov cx, cs

mov es, cx

and dl, dl

jz short nig

;;

;cmp byte ptr ES:[SI], dl ; 0

; tstb \*u.namep / is the character in file name a nul

;;jna nig

; beq nig / yes, end of file name reached;

; / branch to "nig"

namei\_2: ; 1:

;mov dx, 2

mov dl, 2 ; user flag (read, non-owner)

call access

; jsr r0,access; 2 / get i-node with i-number r1

; 'access' will not return here if user has not "r" permission !

test word ptr [i.flgs], 4000h

; bit $40000,i.flgs / directory i-node?

jz error

; beq error3 / no, got an error

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

mov word ptr [u.dirp], ax

; mov i.size,u.dirp / put size of directory in u.dirp

xor ax, ax

mov word ptr [u.off], ax ; 0

; clr u.off / u.off is file offset used by user

mov word ptr [u.fofp], offset u.off

; mov $u.off,u.fofp / u.fofp is a pointer to

; / the offset portion of fsp entry

namei\_3: ; 2:

mov word ptr [u.base], offset u.dirbuf

; mov $u.dirbuf,u.base / u.dirbuf holds a file name

; / copied from a directory

mov word ptr [u.count], 10

; mov $10.,u.count / u.count is byte count

; / for reads and writes

mov ax, word ptr [ii]

; 31/07/2013

inc byte ptr [namei\_r] ; the caller is 'namei' sign

; 28/07/2013 nameir -> u.nameir

; 26/07/2013

;;inc byte ptr [u.namei\_r] ; the caller is 'namei' sign

call readi

; ES = DS after 'readi' !

; jsr r0,readi / read 10. bytes of file

; with i-number (r1); i.e. read a directory entry

mov cx, word ptr [u.nread]

or cx, cx

; tst u.nread

jz short nib

; ble nib / gives error return

;

mov bx, word ptr [u.dirbuf]

and bx, bx

; tst u.dirbuf /

jnz short namei\_4

; bne 3f / branch when active directory entry

; / (i-node word in entry non zero)

mov ax, word ptr [u.off]

sub ax, 10

mov word ptr [u.dirp], ax

; mov u.off,u.dirp

; sub $10.,u.dirp

jmp short namei\_3

; br 2b

; 18/07/2013

nib:

xor ax, ax

stc

nig:

retn

namei\_4: ; 3:

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

;

mov si, word ptr [u.namep]

; mov u.namep,r2 / u.namep points into a file name string

mov di, offset u.dirbuf + 2

; mov $u.dirbuf+2,r3 / points to file name of directory entry

mov dx, offset u.dirbuf + 10

; AX = user segment

mov ds, ax ; Retro UNIX 8086 v1 Feature only !

namei\_5: ; 3:

lodsb ; mov al, byte ptr [SI] ; inc si (al = r4)

; movb (r2)+,r4 / move a character from u.namep string into r4

or al, al

jz short namei\_6

; beq 3f / if char is nul, then the last char in string

; / has been moved

cmp al, '/'

; cmp r4,$'/ / is char a </>

je short namei\_6

; beq 3f

cmp di, dx ; offset u\_dirbuf + 10

; cmp r3,$u.dirbuf+10. / have I checked

; / all 8 bytes of file name

je short namei\_5

; beq 3b

scasb

; cmpb (r3)+,r4 / compare char in u.namep string to file name

; / char read from directory

je short namei\_5

; beq 3b / branch if chars match

mov ax, cs ; Retro UNIX 8086 v1 Feature only !

mov ds, ax ; Retro UNIX 8086 v1 Feature only !

jmp short namei\_3 ; 2b

; br 2b / file names do not match go to next directory entry

namei\_6: ; 3:

; 22/07/2013

mov cx, cs ; Retro UNIX 8086 v1 Feature only !

mov ds, cx ; Retro UNIX 8086 v1 Feature only !

;

cmp di, dx

; cmp r3,$u.dirbuf+10. / if equal all 8 bytes were matched

je short namei\_7

; beq 3f

mov ah, byte ptr [DI]

;inc di

and ah, ah

; tstb (r3)+ /

jnz short namei\_3

; bne 2b

namei\_7: ; 3

mov word ptr [u.namep], si

; mov r2,u.namep / u.namep points to char

; / following a / or nul

;mov bx, word ptr [u.dirbuf]

; mov u.dirbuf,r1 / move i-node number in directory

; / entry to r1

and al, al

; tst r4 / if r4 = 0 the end of file name reached,

; / if r4 = </> then go to next directory

mov ax, bx

jnz namei\_2

; bne 1b

; AX = i-number of the file

;;nig:

;;pop es ; Retro UNIX 8086 v1 Feature only !

retn

; tst (r0)+ / gives non-error return

;;nib:

;; xor ax, ax ; Retro UNIX 8086 v1 modification !

; ax = 0 -> file not found

;;pop es ; Retro UNIX 8086 v1 Feature only !

;; stc ; 27/05/2013

;; retn

; rts r0

syschdir:

; 19/06/2013

; 'syschdir' makes the directory specified in its argument

; the current working directory.

;

; Calling sequence:

; syschdir; name

; Arguments:

; name - address of the path name of a directory

; terminated by nul byte.

; Inputs: -

; Outputs: -

; ...............................................................

;

; Retro UNIX 8086 v1 modification:

; The user/application program puts address of

; the path name in BX register as 'syschdir'

; system call argument.

; (argument transfer method 1)

; / makes the directory specified in the argument

; / the current directory

;;mov ax, 1 ; one/single argument, put argument in BX

;;call arg

;mov bp, word ptr [u.sp\_] ; points to user's BP register

;add bp, 6 ; bx now points to BX on stack

;mov bx, word ptr [BP]

mov word ptr [u.namep], bx

;jsr r0,arg; u.namep / u.namep points to path name

call namei

; jsr r0,namei / find its i-number

jc error

; br error3

call access

; jsr r0,access; 2 / get i-node into core

test word ptr [i.flgs], 4000h

; bit $40000,i.flgs / is it a directory?

jz error

; beq error3 / no error

mov word ptr [u.cdir], ax

; mov r1,u.cdir / move i-number to users

; / current directory

mov ax, word ptr [cdev]

mov word ptr [u.cdrv], ax

; mov cdev,u.cdev / move its device to users

; / current device

jmp sysret

; br sysret3

syschmod: ; < change mode of file >

; 07/07/2013

; 20/06/2013

; 'syschmod' changes mode of the file whose name is given as

; null terminated string pointed to by 'name' has it's mode

; changed to 'mode'.

;

; Calling sequence:

; syschmod; name; mode

; Arguments:

; name - address of the file name

; terminated by null byte.

; mode - (new) mode/flags < attributes >

;

; Inputs: -

; Outputs: -

; ...............................................................

;

; Retro UNIX 8086 v1 modification:

; 'syschmod' system call has two arguments; so,

; Retro UNIX 8086 v1 argument transfer method 2 is used

; to get syschmod system call arguments from the user;

; \* 1st argument, name is pointed to by BX register

; \* 2nd argument, mode is in CX register

;

; Mode bits (Flags):

; bit 0 - write permission for non-owner (1)

; bit 1 - read permission for non-owner (2)

; bit 2 - write permission for owner (4)

; bit 3 - read permission for owner (8)

; bit 4 - executable flag (16)

; bit 5 - set user ID on execution flag (32)

; bit 6,7,8,9,10,11 are not used (undefined)

; bit 12 - large file flag (4096)

; bit 13 - file has modified flag (always on) (8192)

; bit 14 - directory flag (16384)

; bit 15 - 'i-node is allocated' flag (32768)

; / name; mode

call isown

;jsr r0,isown / get the i-node and check user status

test word ptr [i.flgs], 4000h

; bit $40000,i.flgs / directory?

jz short @f

; beq 2f / no

; AL = (new) mode

and al, 0CFh ; 11001111b (clears bit 4 & 5)

; bic $60,r2 / su & ex / yes, clear set user id and

; / executable modes

@@: ; 2:

mov byte ptr [i.flgs], al

; movb r2,i.flgs / move remaining mode to i.flgs

jmp short @f

; br 1f

isown:

; 07/07/2013

; 27/05/2013, 04/05/2013

; 'isown' is given a file name (the 1st argument).

; It find the i-number of that file via 'namei'

; then gets the i-node into core via 'iget'.

; It then tests to see if the user is super user.

; If not, it cheks to see if the user is owner of

; the file. If he is not an error occurs.

; If user is the owner 'setimod' is called to indicate

; the inode has been modificed and the 2nd argument of

; the call is put in r2.

;

; INPUTS ->

; arguments of syschmod and syschown calls

; OUTPUTS ->

; u.uid - id of user

; imod - set to a 1

; r2 - contains second argument of the system call ;

; ((AX=R2) output as 2nd argument))

;

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

;

;;call arg2

;; ; jsr r0,arg2 / u.namep points to file name

;; ! 2nd argument on top of stack !

;; 07/07/2013

mov word ptr [u.namep], bx ;; 1st argument

push cx ;; 2nd argument

;;

call namei

; jsr r0,namei / get its i-number

; Retro UNIX 8086 v1 modification !

; ax = 0 -> file not found

;and ax, ax

;jz error

jc error ; 27/05/2013

; br error3

call iget

; jsr r0,iget / get i-node into core

mov al, byte ptr [u.uid\_] ; 02/08/2013

or al, al

; tstb u.uid / super user?

jz short @f

; beq 1f / yes, branch

cmp al, byte ptr [i.uid]

; cmpb i.uid,u.uid / no, is this the owner of

; / the file

jne error

; beq 1f / yes

; jmp error3 / no, error

@@: ; 1:

call setimod

; jsr r0,setimod / indicates

; ; / i-node has been modified

pop ax ; 2nd argument

; mov (sp)+,r2 / mode is put in r2

; / (u.off put on stack with 2nd arg)

retn

; rts r0

syschown: ; < change owner of file >

; 02/08/2013

; 07/07/2013, 20/06/2013

; 'syschown' changes the owner of the file whose name is given

; as null terminated string pointed to by 'name' has it's owner

; changed to 'owner'

;

; Calling sequence:

; syschown; name; owner

; Arguments:

; name - address of the file name

; terminated by null byte.

; owner - (new) owner (number/ID)

;

; Inputs: -

; Outputs: -

; ...............................................................

; Retro UNIX 8086 v1 modification:

; 'syschown' system call has two arguments; so,

; Retro UNIX 8086 v1 argument transfer method 2 is used

; to get syschown system call arguments from the user;

; \* 1st argument, name is pointed to by BX register

; \* 2nd argument, owner number is in CX register

;

; / name; owner

call isown

; jsr r0,isown / get the i-node and check user status

cmp byte ptr [u.uid\_], 0 ; 02/08/2013

; tstb u.uid / super user

jz short @f

; beq 2f / yes, 2f

test byte ptr [i.flgs], 20h ; 32

; bit $40,i.flgs / no, set userid on execution?

jnz error

; bne 3f / yes error, could create Trojan Horses

@@: ; 2:

; AL = owner (number/ID)

mov byte ptr [u.uid\_], al ; 02/08/2013

; movb r2,i.uid / no, put the new owners id

; / in the i-node

jmp sysret

; 1:

; jmp sysret4

; 3:

; jmp error

;;arg: ; < get system call arguments >

; 22/05/2013 'method 4' has been modified (corrected)

; 04/05/2013

; 'arg' extracts an argument for a routine whose call is

; of form:

; sys 'routine' ; arg1

; or

; sys 'routine' ; arg1 ; arg2

; or

; sys 'routine' ; arg1;...;arg10 (sys exec)

;

; RETRO UNIX 8086 v1 Modification !

; Retro Unix 8086 v1 system call argument

; transfer methods:

; 1) Single argument in BX register

; ('arg' routine is called with AX=1)

; 2) Two arguments,

; 1st argument in BX register

; 2nd argument in CX register

; ('arg' routine is called with AX=2)

; 3) Three arguments

; 3rd argument in DX register

; ('arg' routine is called with AX=3)

; 4) Argument list address in BP register

; ('arg' routine is called with AX=0)

; 'arg' routine will return arguments in same registers

; except method 4 will return current argument

; which is pointed by BP register and 'arg' will

; increase value of (user's) BP register (on stack)

; in order to point next argument. AX register will

; return address of current argument.

; INPUTS ->

; u.sp+18 - contains a pointer to one of arg1..argn

; This pointers's value is actually the value of

; update pc at the the trap to sysent (unkni) is

; made to process the sys instruction

; r0 - contains the return address for the routine

; that called arg. The data in the word pointer

; to by the return address is used as address

; in which the extracted argument is stored

;

; OUTPUTS ->

; 'address' - contains the extracted argument

; u.sp+18 - is incremented by 2

; r1 - contains the extracted argument

; r0 - points to the next instruction to be

; executed in the calling routine.

;

; ((Modified registers: AX, DX, CX, BX))

; Retro UNIX 8086 v1 modification !

; [ sysunlink, sysfstat, syschdir, sysbreak, sysseek (seektell),

; sysintr, sysquit, rw1 (sysread, syswrite), sysemt, sysilgins

; sysmdate, gtty (sysgtty) etc. call arg.]

;

; Note: If all of system calls which call 'arg' routine will have

; only 1 argument, this 'arg' routine may be simplified

; and system calls with 2 arguments may be changed to use 'arg1'

; instead of 'arg' (04/05/2013).

;; mov bx, word ptr [u.sp\_] ; points to user's BP register

;; mov cx, ax

;; or cx, cx

;; jnz short @f

;arg\_bp: ; method 4

;; mov ax, word ptr [BX] ; value of BP register on stack

; (sAX = uBP)

;; mov dx, ax

; AX = 1st argument or current argument (method 4)

;; inc dx

;; inc dx

;; mov word ptr [BX], dx ; BP will point to next argument

; (uBP = uBP+2)

;; retn

; method 1, 2, 3

;;@@:

;; add bx, 6 ; bx now points to BX on stack

;,@@:

;; mov dx, word ptr [BX]

;; push dx ; 1st or 2nd or 3rd argument (depends on CX)

;; dec cx

;; jz short @f

;; inc bx

;; inc bx

;; jmp short @b

;;@@:

;; dec ax

;; jz short @f

;; pop cx ; 2nd or 3rd argument (depends on value in AX)

;; dec ax

;; jz short @f

;; mov dx, cx ; 3rd argument

;; pop cx ; 2nd argument

;;@@:

;; pop bx ; 1st argument

;; retn

; UNIX v1 original 'arg' routine here:

; mov u.sp,r1

; mov \*18.(r1),\*(r0)+ / put argument of system call

; / into argument of arg2

; add $2,18.(r1) / point pc on stack

; / to next system argument

; rts r0

;;arg2: ; < get system calls arguments - with file name pointer>

; 22/05/2013 arg1 modified (corrected)

; 04/05/2013

; 'arg2' takes first argument in system call

; (pointer to name of the file) and puts it in location

; u.namep; takes second argument and puts it in u.off

; and on top of the stack

;

; RETRO UNIX 8086 v1 Modification !

; Retro Unix 8086 v1 system call argument

; transfer methods:

; 1) Single argument in BX register

; ('arg' routine is called with AX=1)

; 2) Two arguments,

; 1st argument in BX register

; 2nd argument in CX register

; ('arg' routine is called with AX=2)

; 3) Three arguments

; 3rd argument in DX register

; ('arg' routine is called with AX=3)

; 4) Argument list address in BP register

; ('arg' routine is called with AX=0)

; 'arg2' routine uses method 2 when calling 'arg' routine

; then puts 1st argument (BX) in u.namep and pushes

; 2nd argument (CX) on stack.

; (Retro UNIX 8086 v1 does not put 2nd argument in u.off)

;

; INPUTS ->

; u.sp, r0

;

; OUTPUTS ->

; u.namep

; u.off

; u.off pushed on stack

; r1

;

; ((Modified registers: AX, DX, CX, BX))

;

; arg2 (1) -- 04/05/2013 (1)

; mov ax, 2 ; two arguments, method 2

; call arg

; ; BX = 1st argument

; ; CX = 2nd argument

; arg2 (modified for arg1 call) -- 04/05/2013 (2)

; Retro UNIX 8086 v1 modification !

; Direct argument handling instead of using 'arg' call.

; [ sysexec, sysmount, sysopen, syslink, sysstat,

; isown (syschmod, syschown),sysopen, syscreat, sysmkdir, sysmount

; call arg2 ]

;; call arg1 ; 04/05/2013

;; mov word ptr [u.namep], ax ; 1st argument

;; pop dx ; return address

;; push cx ; 2nd argument

;; push dx

; warning !

; ! Caller must pop 2nd argument on stack !

;; retn

;;arg1: ; Retro UNIX 8086 v1 feature only !

; 22/05/2013 modified (corrected)

;; mov bx, word ptr [u.sp\_] ; points to user's BP register

;; add bx, 6

;, mov ax, [BX] ; points to user's BX register

;(sAX = uBX)

;; inc bx

;; inc bx

;, mov cx, [BX] ; points to user's CX register

;(sCX = uCX)

; retn

;; arg2 (2) -- 04/05/2013 (1)

; mov word ptr [u.namep], bx ; file name pointer

; ;mov word ptr [u.off], cx ; 2nd argument

; pop dx ; return address

; push cx

; push dx

; ; warning !

; ; ! Caller must pop 2nd argument on stack !

; retn

; UNIX v1 original 'arg2' routine here:

; jsr r0,arg; u.namep / u.namep contains value of

; / first arg in sys call

; jsr r0,arg; u.off / u.off contains value of

; / second arg in sys call

; mov r0,r1 / r0 points to calling routine

; mov (sp),r0 / put operation code back in r0

; mov u.off,(sp) / put pointer to second argument

; / on stack

; jmp (r1) / return to calling routine

systime:

; 20/06/2013

; 'systime' gets the time of the year.

; The present time is put on the stack.

;

; Calling sequence:

; systime

; Arguments: -

;

; Inputs: -

; Outputs: sp+2, sp+4 - present time

; ...............................................................

; Retro UNIX 8086 v1 modification:

; 'systime' system call will return to the user

; with unix time (epoch) in DX:AX register pair

;

; !! Major modification on original Unix v1 'systime'

; system call for PC compatibility !!

; / get time of year

call epoch

mov word ptr [u.r0], ax

mov bp, word ptr [u.sp\_]

add bp, 10 ; points to the user's DX register

mov word ptr [BP], dx

; mov s.time,4(sp)

; mov s.time+2,2(sp) / put the present time

; / on the stack

; br sysret4

jmp sysret

sysstime:

; 02/08/2013

; 20/06/2013

; 'sysstime' sets the time. Only super user can use this call.

;

; Calling sequence:

; sysstime

; Arguments: -

;

; Inputs: sp+2, sp+4 - time system is to be set to.

; Outputs: -

; ...............................................................

; Retro UNIX 8086 v1 modification:

; the user calls 'sysstime' with unix (epoch) time

; (to be set) is in CX:BX register pair as two arguments.

;

; Retro UNIX 8086 v1 argument transfer method 2 is used

; to get sysstime system call arguments from the user;

; \* 1st argument, lowword of unix time is in BX register

; \* 2nd argument, highword of unix time is in CX register

;

; !! Major modification on original Unix v1 'sysstime'

; system call for PC compatibility !!

; / set time

cmp byte ptr [u.uid\_], 0 ; 02/08/2013

; tstb u.uid / is user the super user

ja error

; bne error4 / no, error

; CX:BX = unix (epoch) time (from user)

mov dx, cx

mov ax, bx

; DX:AX = unix (epoch) time (to subroutine)

;call convert\_from\_epoch

call set\_date\_time

; mov 4(sp),s.time

; mov 2(sp),s.time+2 / set the system time

jmp sysret

; br sysret4

sysbreak:

; 24/03/2014

; 19/11/2013

; 20/06/2013

; 'sysbreak' sets the programs break points.

; It checks the current break point (u.break) to see if it is

; between "core" and the stack (sp). If it is, it is made an

; even address (if it was odd) and the area between u.break

; and the stack is cleared. The new breakpoint is then put

;in u.break and control is passed to 'sysret'.

;

; Calling sequence:

; sysbreak; addr

; Arguments: -

;

; Inputs: u.break - current breakpoint

; Outputs: u.break - new breakpoint

; area between old u.break and the stack (sp) is cleared.

; ...............................................................

;

; Retro UNIX 8086 v1 modification:

; The user/application program puts breakpoint address

; in BX register as 'sysbreak' system call argument.

; (argument transfer method 1)

;

; NOTE: Beginning of core is 0 in Retro UNIX 8086 v1 !

; ((!'sysbreak' is not needed in Retro UNIX 8086 v1!))

; NOTE:

; 'sysbreak' clears extended part (beyond of previous

; 'u.break' address) of user's memory for original unix's

; 'bss' compatibility with Retro UNIX 8086 v1 (19/11/2013)

;cmp word ptr [u.break], core

; mov u.break,r1 / move users break point to r1

; cmp r1,$core / is it the same or lower than core?

;ja short sysbreak\_3

; blos 1f / yes, 1f

mov di, word ptr [u.break]

cmp di, word ptr [u.usp]

; cmp r1,sp / is it the same or higher

; / than the stack?

jnb short sysbreak\_3

; bhis 1f / yes, 1f

mov ax, word ptr [u.segmnt]

mov es, ax

xor ax, ax

test di, 1

; bit $1,r1 / is it an odd address

jz short sysbreak\_1

; beq 2f / no, its even

stosb

; clrb (r1)+ / yes, make it even

sysbreak\_0: ; 2: / clear area between the break point and the stack

cmp di, word ptr [u.usp] ; 24/03/2014

; cmp r1,sp / is it higher or same than the stack

jnb short sysbreak\_2

; bhis 1f / yes, quit

sysbreak\_1:

stosw

; clr (r1)+ / clear word

jmp short sysbreak\_0

; br 2b / go back

sysbreak\_2: ; 1:

mov ax, ds

mov es, ax

sysbreak\_3:

mov word ptr [u.break], bx

; jsr r0,arg; u.break / put the "address"

; / in u.break (set new break point)

jmp sysret

; br sysret4 / br sysret

maknod:

; 02/08/2013

; 31/07/2013

; 17/07/2013

; 02/05/2013

; 'maknod' creates an i-node and makes a directory entry

; for this i-node in the current directory.

;

; INPUTS ->

; r1 - contains mode

; ii - current directory's i-number

;

; OUTPUTS ->

; u.dirbuf - contains i-number of free i-node

; i.flgs - flags in new i-node

; i.uid - filled with u.uid

; i.nlks - 1 is put in the number of links

; i.ctim - creation time

; i.ctim+2 - modification time

; imod - set via call to setimod

;

; ((AX = R1)) input

;

; (Retro UNIX Prototype :

; 30/10/2012 - 01/03/2013, UNIXCOPY.ASM)

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

; / r1 contains the mode

or ah, 80h ; 10000000b

; bis $100000,r1 / allocate flag set

push ax

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

; 31/07/2013

mov ax, word ptr [ii] ; move current i-number to AX/r1

; mov ii,r1 / move current i-number to r1

mov dl, 1 ; owner flag mask

call access

; jsr r0,access; 1 / get its i-node into core

push ax

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

mov ax, 40

; mov $40.,r1 / r1 = 40

@@: ; 1: / scan for a free i-node (next 4 instructions)

inc ax

; inc r1 / r1 = r1 + 1

call imap

; jsr r0,imap / get byte address and bit position in

; / inode map in r2 & m

; DX (MQ) has a 1 in the calculated bit position

; BX (R2) has byte address of the byte with allocation bit

test byte ptr [BX], dl

; bitb mq,(r2) / is the i-node active

jnz short @b

; bne 1b / yes, try the next one

or byte ptr [BX], dl

; bisb mq,(r2) / no, make it active

; / (put a 1 in the bit map)

call iget

; jsr r0,iget / get i-node into core

test word ptr [i.flgs], 8000h

; tst i.flgs / is i-node already allocated

jnz short @b

; blt 1b / yes, look for another one

mov word ptr [u.dirbuf], ax

; mov r1,u.dirbuf / no, put i-number in u.dirbuf

pop ax

; mov (sp)+,r1 / get current i-number back

call iget

; jsr r0,iget / get i-node in core

call mkdir

; jsr r0,mkdir / make a directory entry

; / in current directory

mov ax, word ptr [u.dirbuf]

; mov u.dirbuf,r1 / r1 = new inode number

call iget

; jsr r0,iget / get it into core

;jsr r0,copyz; inode; inode+32. / 0 it out

mov cx, 16

xor ax, ax ; 0

;mov di, offset inode

mov di, offset i ; 17/07/2013

rep stosw

;

pop word ptr [i.flgs]

; mov (sp)+,i.flgs / fill flags

mov cl, byte ptr [u.uid\_] ; 02/08/2013

mov byte ptr [i.uid], cl

; movb u.uid,i.uid / user id

mov byte ptr [i.nlks], 1

; movb $1,i.nlks / 1 link

;call epoch ; Retro UNIX 8086 v1 modification !

;mov ax, word ptr [s.time]

;mov dx, word ptr [s.time]+2

;mov word ptr [i.ctim], ax

;mov word ptr [i.ctim]+2, dx

; mov s.time,i.ctim / time created

; mov s.time+2,i.ctim+2 / time modified

; Retro UNIX 8086 v1 modification !

; i.ctime=0, i.ctime+2=0 and

; 'setimod' will set ctime of file via 'epoch'

call setimod

; jsr r0,setimod / set modified flag

retn

; rts r0 / return

sysseek: ; / moves read write pointer in an fsp entry

; 05/08/2013

; 07/07/2013

; 'sysseek' changes the r/w pointer of (3rd word of in an

; fsp entry) of an open file whose file descriptor is in u.r0.

; The file descriptor refers to a file open for reading or

; writing. The read (or write) pointer is set as follows:

; \* if 'ptrname' is 0, the pointer is set to offset.

; \* if 'ptrname' is 1, the pointer is set to its

; current location plus offset.

; \* if 'ptrname' is 2, the pointer is set to the

; size of file plus offset.

; The error bit (e-bit) is set for an undefined descriptor.

;

; Calling sequence:

; sysseek; offset; ptrname

; Arguments:

; offset - number of bytes desired to move

; the r/w pointer

; ptrname - a switch indicated above

;

; Inputs: r0 - file descriptor

; Outputs: -

; ...............................................................

;

; Retro UNIX 8086 v1 modification:

; 'sysseek' system call has three arguments; so,

; Retro UNIX 8086 v1 argument transfer method 3 is used

; to get sysseek system call arguments from the user;

; \* 1st argument, file descriptor is in BX (BL) register

; \* 2nd argument, offset is in CX register

; \* 3rd argument, ptrname/switch is in DX (DL) register

;

call seektell

; jsr r0,seektell / get proper value in u.count

; AX = u.count

; BX = \*u.fofp

; add u.base,u.count / add u.base to it

add ax, word ptr [u.base] ; add offset (u.base) to base

mov word ptr [BX], ax

; mov u.count,\*u.fofp / put result into r/w pointer

jmp sysret

; br sysret4

systell: ; / get the r/w pointer

; 05/08/2013

; 07/07/2013

; Retro UNIX 8086 v1 modification:

; ! 'systell' does not work in original UNIX v1,

; it returns with error !

; Inputs: r0 - file descriptor

; Outputs: r0 - file r/w pointer

;xor cx, cx ; 0

mov dx, 1 ; 05/08/2013

;call seektell

call seektell0 ; 05/08/2013

;mov bx, word ptr [u.fofp]

mov ax, word ptr [BX]

mov word ptr [u.r0], ax

jmp sysret

; Original unix v1 'systell' system call:

; jsr r0,seektell

; br error4

seektell:

; 05/08/2013 (return AX as base for offset)

; 07/07/2013

; 'seektell' puts the arguments from sysseek and systell

; call in u.base and u.count. It then gets the i-number of

; the file from the file descriptor in u.r0 and by calling

; getf. The i-node is brought into core and then u.count

; is checked to see it is a 0, 1, or 2.

; If it is 0 - u.count stays the same

; 1 - u.count = offset (u.fofp)

; 2 - u.count = i.size (size of file)

;

; !! Retro UNIX 8086 v1 modification:

; Argument 1, file descriptor is in BX;

; Argument 2, offset is in CX;

; Argument 3, ptrname/switch is in DX register.

;

; mov ax, 3 ; Argument transfer method 3 (three arguments)

; call arg

;

; ((Return -> ax = base for offset (position= base+offset))

;

mov word ptr [u.base], cx ; offset

; jsr r0,arg; u.base / puts offset in u.base

seektell0:

mov word ptr [u.count], dx

; jsr r0,arg; u.count / put ptr name in u.count

; mov ax, bx

; mov \*u.r0,r1 / file descriptor in r1

; / (index in u.fp list)

; call getf

; jsr r0,getf / u.fofp points to 3rd word in fsp entry

; BX = file descriptor (file number)

call getf1

or ax, ax ; i-number of the file

; mov r1,-(sp) / r1 has i-number of file,

; / put it on the stack

jz error

; beq error4 / if i-number is 0, not active so error

;push ax

cmp ah, 80h

jb short @f

; bgt .+4 / if its positive jump

neg ax

; neg r1 / if not make it positive

@@:

call iget

; jsr r0,iget / get its i-node into core

mov bx, word ptr [u.fofp] ; 05/08/2013

cmp byte ptr [u.count], 1

; cmp u.count,$1 / is ptr name =1

ja short @f

; blt 2f / no its zero

je short seektell\_1

; beq 1f / yes its 1

xor ax, ax

;jmp short seektell\_2

retn

@@:

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

; mov i.size,u.count / put number of bytes

; / in file in u.count

;jmp short seektell\_2

; br 2f

retn

seektell\_1: ; 1: / ptrname =1

;mov bx, word ptr [u.fofp]

mov ax, word ptr [BX]

; mov \*u.fofp,u.count / put offset in u.count

;seektell\_2: ; 2: / ptrname =0

;mov word ptr [u.count], ax

;pop ax

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

retn

; rts r0

sysintr: ; / set interrupt handling

; 07/07/2013

; 'sysintr' sets the interrupt handling value. It puts

; argument of its call in u.intr then branches into 'sysquit'

; routine. u.tty is checked if to see if a control tty exists.

; If one does the interrupt character in the tty buffer is

; cleared and 'sysret'is called. If one does not exits

; 'sysret' is just called.

;

; Calling sequence:

; sysintr; arg

; Argument:

; arg - if 0, interrupts (ASCII DELETE) are ignored.

; - if 1, intterupts cause their normal result

; i.e force an exit.

; - if arg is a location within the program,

; control is passed to that location when

; an interrupt occurs.

; Inputs: -

; Outputs: -

; ...............................................................

; Retro UNIX 8086 v1 modification:

; 'sysintr' system call sets u.intr to value of BX

; then branches into sysquit.

;

mov word ptr [u.intr], bx

;jmp short @f

;jsr r0,arg; u.intr / put the argument in u.intr

; br 1f / go into quit routine

jmp sysret

sysquit:

; 07/07/2013

; 'sysquit' turns off the quit signal. it puts the argument of

; the call in u.quit. u.tty is checked if to see if a control

; tty exists. If one does the interrupt character in the tty

; buffer is cleared and 'sysret'is called. If one does not exits

; 'sysret' is just called.

;

; Calling sequence:

; sysquit; arg

; Argument:

; arg - if 0, this call diables quit signals from the

; typewriter (ASCII FS)

; - if 1, quits are re-enabled and cause execution to

; cease and a core image to be produced.

; i.e force an exit.

; - if arg is an addres in the program,

; a quit causes control to sent to that

; location.

; Inputs: -

; Outputs: -

; ...............................................................

; Retro UNIX 8086 v1 modification:

; 'sysquit' system call sets u.quit to value of BX

; then branches into 'sysret'.

;

mov word ptr [u.quit], bx

jmp sysret

; jsr r0,arg; u.quit / put argument in u.quit

;1:

; mov u.ttyp,r1 / move pointer to control tty buffer

; / to r1

; beq sysret4 / return to user

; clrb 6(r1) / clear the interrupt character

; / in the tty buffer

; br sysret4 / return to user

syssetuid: ; / set process id

; 02/08/2013

; 07/07/2013

; 'syssetuid' sets the user id (u.uid) of the current process

; to the process id in (u.r0). Both the effective user and

; u.uid and the real user u.ruid are set to this.

; Only the super user can make this call.

;

; Calling sequence:

; syssetuid

; Arguments: -

;

; Inputs: (u.r0) - contains the process id.

; Outputs: -

; ...............................................................

;

; Retro UNIX 8086 v1 modification:

; BL contains the (new) user ID of the current process

; movb \*u.r0,r1 / move process id (number) to r1

cmp bl, byte ptr [u.ruid]

; cmpb r1,u.ruid / is it equal to the real user

; / id number

je short @f

; beq 1f / yes

cmp byte ptr [u.uid\_], 0 ; 02/08/2013

; tstb u.uid / no, is current user the super user?

ja error

; bne error4 / no, error

mov byte ptr [u.ruid], bl

@@: ; 1:

mov byte ptr [u.uid\_], bl ; 02/08/2013

; movb r1,u.uid / put process id in u.uid

; movb r1,u.ruid / put process id in u.ruid

jmp sysret

; br sysret4 / system return

sysgetuid: ; < get user id >

; 07/07/2013

; 'sysgetuid' returns the real user ID of the current process.

; The real user ID identifies the person who is logged in,

; in contradistinction to the effective user ID, which

; determines his access permission at each moment. It is thus

; useful to programs which operate using the 'set user ID'

; mode, to find out who invoked them.

;

; Calling sequence:

; syssetuid

; Arguments: -

;

; Inputs: -

; Outputs: (u.r0) - contains the real user's id.

; ...............................................................

;

; Retro UNIX 8086 v1 modification:

; AL contains the real user ID at return.

;

;xor ah, ah

mov al, byte ptr [u.ruid]

mov word ptr [u.r0], ax

; movb u.ruid,\*u.r0 / move the real user id to (u.r0)

jmp sysret

; br sysret4 / systerm return, sysret

anyi:

; 25/04/2013

; 'anyi' is called if a file deleted while open.

; "anyi" checks to see if someone else has opened this file.

;

; INPUTS ->

; r1 - contains an i-number

; fsp - start of table containing open files

;

; OUTPUTS ->

; "deleted" flag set in fsp entry of another occurrence of

; this file and r2 points 1st word of this fsp entry.

; if file not found - bit in i-node map is cleared

; (i-node is freed)

; all blocks related to i-node are freed

; all flags in i-node are cleared

; ((AX = R1)) input

;

; (Retro UNIX Prototype : 02/12/2012, UNIXCOPY.ASM)

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

;

; / r1 contains an i-number

mov bx, offset fsp

; mov $fsp,r2 / move start of fsp table to r2

anyi\_1: ; 1:

cmp ax, word ptr [BX]

; cmp r1,(r2) / do i-numbers match?

je short anyi\_2

; beq 1f / yes, 1f

neg ax

; neg r1 / no complement r1

cmp ax, word ptr [BX]

; cmp r1,(r2) / do they match now?

je short anyi\_2

; beq 1f / yes, transfer

; / i-numbers do not match

add bx, 8

; add $8,r2 / no, bump to next entry in fsp table

cmp bx, offset fsp + (nfiles\*8)

; cmp r2,$fsp+[nfiles\*8]

; / are we at last entry in the table

jb short anyi\_1

; blt 1b / no, check next entries i-number

;cmp ax, 32768

cmp ah, 80h ; negative number check

; tst r1 / yes, no match

; bge .+4

jb short @f

neg ax

; neg r1 / make i-number positive

@@:

call imap

; jsr r0,imap / get address of allocation bit

; / in the i-map in r2

;; DL/DX (MQ) has a 1 in the calculated bit position

;; BX (R2) has address of the byte with allocation bit

; not dx

not dl ;; 0 at calculated bit position, other bits are 1

;and word ptr [BX], dx

and byte ptr [BX], dl

; bicb mq,(r2) / clear bit for i-node in the imap

call itrunc

; jsr r0,itrunc / free all blocks related to i-node

mov word ptr [i.flgs], 0

; clr i.flgs / clear all flags in the i-node

retn

;rts r0 / return

anyi\_2: ; 1: / i-numbers match

inc byte ptr [BX]+7

;incb 7(r2) / increment upper byte of the 4th word

; / in that fsp entry (deleted flag of fsp entry)

retn

; rts r0