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

;

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

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

; U1.ASM (include u1.asm) //// UNIX v1 -> u1.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: 12/07/2014 ] ;;; 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>

;

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

; 11/06/2014, 26/06/2014, 04/07/2014

; 07/03/2014, 10/04/2014, 15/04/2014, 22/04/2014, 30/04/2014

; 18/01/2014, 26/01/2014, 05/02/2014, 14/02/2014, 23/02/2014

; 12/01/2014, 13/01/2014, 14/01/2014, 16/01/2014, 17/01/2014

; 18/11/2013, 04/12/2013, 06/12/2013, 07/12/2013, 10/12/2013

; 20/10/2013, 23/10/2013, 24/10/2013, 30/10/2013, 04/11/2013

; 03/09/2013, 16/09/2013, 17/09/2013, 22/09/2013, 29/09/2013

; 14/08/2013, 18/08/2013, 19/08/2013, 21/08/2013, 30/08/2013

; 26/07/2013, 02/08/2013, 07/08/2013, 08/08/2013, 11/08/2013

; 15/07/2013, 16/07/2013, 22/07/2013, 23/07/2013, 24/07/2013

; 27/05/2013, 30/05/2013, 02/06/2013, 03/06/2013, 14/07/2013

; 20/05/2013, 22/05/2013, 23/05/2013, 24/05/2013, 26/05/2013

; 26/04/2013, 04/05/2013, 09/05/2013, 15/05/2013, 16/05/2013

; 11/03/2013, 10/04/2013, 16/04/2013, 17/04/2013, 19/04/2013

;

unkni: ; / used for all system calls

sysent: ; < enter to system call >

; 18/01/2014

; 26/07/2013

; 24/07/2013

; 14/07/2013

; 24/05/2013

; 16/04/2013

; 10/04/2013

;

; 'unkni' or 'sysent' is sytem entry from various traps.

; The trap type is determined and an indirect jump is made to

; the appropriate system call handler. If there is a trap inside

; the system a jump to panic is made. All user registers are saved

; and u.sp points to the end of the users stack. The sys (trap)

; instructor is decoded to get the the system code part (see

; trap instruction in the PDP-11 handbook) and from this

; the indirect jump address is calculated. If a bad system call is

; made, i.e., the limits of the jump table are exceeded, 'badsys'

; is called. If the call is legitimate control passes to the

; appropriate system routine.

;

; Calling sequence:

; Through a trap caused by any sys call outside the system.

; Arguments:

; Arguments of particular system call.

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

;

; Retro UNIX 8086 v1 modification:

; System call number is in AX register.

;

; Other parameters are in DX, BX, CX, SI, DI, BP registers

; depending of function details.

;

; 16/04/2013 segment changing

push cs

pop ds

;

inc byte ptr [sysflg]

; incb sysflg / indicate a system routine is in progress

sti ; 18/01/2014

jnz panic ; 24/05/2013

;jz short @f

; beq 1f

;jmp short panic

; jmp panic ; / called if trap inside system

@@: ;1:

; 24/05/2013

mov word ptr [u.r0], ax

mov word ptr [u.usp], sp

; 16/04/2013 stack segment changing

;mov ax, ss

;mov word ptr [u.segmnt], ax

mov ax, cs

; 24/05/2013

;;;mov es, ax ; 14/07/2013

cli

; 24/07/2013

mov sp, sstack ; offset sstack ; swap stack

; (System/Kernel stack in Retro UNIX 8086 v1 !)

mov ss, ax

sti

; 24/05/2013

push word ptr [u.usp] ; user's stack pointer (old sp)

; which points to top of user's stack

; (Retro UNIX 8086 v1 modification!)

;

push dx

push cx

push bx

push si

push di

push bp

;

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

;;mov ax, word ptr [s.syst+2]

;;mov word ptr [clockp], ax

; mov $s.syst+2,clockp

; mov r0,-(sp) / save user registers

; mov sp,u.r0 / pointer to bottom of users stack

; / in u.r0

; mov r1,-(sp)

; mov r2,-(sp)

; mov r3,-(sp)

; mov r4,-(sp)

; mov r5,-(sp)

; mov ac,-(sp) / "accumulator" register for extended

; / arithmetic unit

; mov mq,-(sp) / "multiplier quotient" register for the

; / extended arithmetic unit

; mov sc,-(sp) / "step count" register for the extended

; / arithmetic unit

; mov sp,u.sp / u.sp points to top of users stack

; mov 18.(sp),r0 / store pc in r0

; mov -(r0),r0 / sys inst in r0 10400xxx

; sub $sys,r0 / get xxx code

mov ax, word ptr [u.r0]

shl ax, 1

; asl r0 / multiply by 2 to jump indirect in bytes

cmp ax, offset @f - offset syscalls

; cmp r0,$2f-1f / limit of table (35) exceeded

;jnb short badsys

; bhis badsys / yes, bad system call

; 16/04/2013

cmc

pushf

push ax

; 24/05/2013

mov bp, word ptr [u.usp]

; 26/07/2013

;mov ax, 0FFFEh

mov al, 0FEh ; 11111110b

adc al, 0 ; al = al + cf

;and word ptr ES:[BP]+4, ax ; flags

;;mov ax, word ptr [u.segmnt]

;;mov es, ax

and byte ptr ES:[BP]+4, al ; flags (reset carry flag)

; bic $341,20.(sp) / set users processor priority to 0

; / and clear carry bit

mov ax, ds ; 14/07/2013

mov es, ax ; 17/07/2013

;pop ax

;mov bp, ax

;shr ax, 1

pop bp ; ax

;mov ax, word ptr [u.r0]

popf

jc badsys

mov ax, word ptr [u.r0]

; system call registers: AX, DX, CX, BX, SI, DI

jmp word ptr [BP]+syscalls

; jmp \*1f(r0) / jump indirect thru table of addresses

; / to proper system routine.

syscalls: ; 1:

dw offset sysrele ; / 0

dw offset sysexit ; / 1

dw offset sysfork ; / 2

dw offset sysread ; / 3

dw offset syswrite ; / 4

dw offset sysopen ; / 5

dw offset sysclose ; / 6

dw offset syswait ; / 7

dw offset syscreat ; / 8

dw offset syslink ; / 9

dw offset sysunlink ; / 10

dw offset sysexec ; / 11

dw offset syschdir ; / 12

dw offset systime ; / 13

dw offset sysmkdir ; / 14

dw offset syschmod ; / 15

dw offset syschown ; / 16

dw offset sysbreak ; / 17

dw offset sysstat ; / 18

dw offset sysseek ; / 19

dw offset systell ; / 20

dw offset sysmount ; / 21

dw offset sysumount ; / 22

dw offset syssetuid ; / 23

dw offset sysgetuid ; / 24

dw offset sysstime ; / 25

dw offset sysquit ; / 26

dw offset sysintr ; / 27

dw offset sysfstat ; / 28

dw offset sysemt ; / 29

dw offset sysmdate ; / 30

dw offset sysstty ; / 31

dw offset sysgtty ; / 32

dw offset sysilgins ; / 33

dw offset syssleep ; 34 ; Retro UNIX 8086 v1 feature only !

; 11/06/2014

@@: ;2:

error:

; 07/08/2013

; 26/05/2013

; 24/05/2013

; 22/05/2013

; 04/05/2013

; 18/04/2013

; 16/04/2013

; 10/04/2013

; 'error' merely sets the error bit off the processor status (c-bit)

; then falls right into the 'sysret', 'sysrele' return sequence.

;

; INPUTS -> none

; OUTPUTS ->

; processor status - carry (c) bit is set (means error)

;

; 26/05/2013 (Stack pointer must be reset here!

; Because, jumps to error procedure

; disrupts push-pop nesting balance)

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

mov bp, sp

; mov u.sp,r1

mov bx, word ptr [BP]+12 ; user's stack pointer

;

mov ax, word ptr [u.segmnt]

mov es, ax

;;push ds

;;mov ds, ax

;

;;; word ptr ES:[BX] -> IP

;;; word ptr ES:[BX]+2 -> CS

;;; word ptr ES:[BX]+4 -> FLAGS

;;or byte ptr [BX]+4, 1

or byte ptr ES:[BX]+4, 1 ; set carry bit of flags register

; in user's stack

; bis $1,20.(r1) / set c bit in processor status word below

; / users stack

;;pop ds

mov ax, cs

mov es, ax

; 07/08/2013

mov word ptr [namei\_r], 0 ; namei\_r, mkdir\_w reset

sysret: ; < return from system call>

; 23/02/2014

; 07/08/2013

; 24/05/2013

; 04/05/2013

; 26/04/2013

; 10/04/2013

;

; 'sysret' first checks to see if process is about to be

; terminated (u.bsys). If it is, 'sysexit' is called.

; If not, following happens:

; 1) The user's stack pointer is restored.

; 2) r1=0 and 'iget' is called to see if last mentioned

; i-node has been modified. If it has, it is written out

; via 'ppoke'.

; 3) If the super block has been modified, it is written out

; via 'ppoke'.

; 4) If the dismountable file system's super block has been

; modified, it is written out to the specified device

; via 'ppoke'.

; 5) A check is made if user's time quantum (uquant) ran out

; during his execution. If so, 'tswap' is called to give

; another user a chance to run.

; 6) 'sysret' now goes into 'sysrele'.

; (See 'sysrele' for conclusion.)

;

; Calling sequence:

; jump table or 'br sysret'

; Arguments:

; -

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

;

; ((AX=r1 for 'iget' input))

;

xor ax, ax ; 04/05/2013

inc al ; 04/05/2013

cmp byte ptr [u.bsys], al ; 1

; tstb u.bsys / is a process about to be terminated because

jnb sysexit ; 04/05/2013

; bne sysexit / of an error? yes, go to sysexit

;mov sp, word ptr [u.sp\_] ; 24/05/2013 (that is not needed here)

; mov u.sp,sp / no point stack to users stack

dec al ; mov ax, 0

; clr r1 / zero r1 to check last mentioned i-node

call iget

; jsr r0,iget / if last mentioned i-node has been modified

; / it is written out

xor ax, ax ; 0

cmp byte ptr [smod], al ; 0

; tstb smod / has the super block been modified

jna short @f

; beq 1f / no, 1f

mov byte ptr [smod], al ; 0

; clrb smod / yes, clear smod

mov bx, offset sb0 ;; 07/08//2013

or word ptr [BX], 200h ;;

;or word ptr [sb0], 200h ; write bit, bit 9

; bis $1000,sb0 / set write bit in I/O queue for super block

; / output

; AX = 0

call poke ; 07/08/2013

; call ppoke

; AX = 0

; jsr r0,ppoke / write out modified super block to disk

@@: ;1:

cmp byte ptr [mmod], al ; 0

; tstb mmod / has the super block for the dismountable file

; / system

jna short @f ; 23/02/2014 (@f location has been changed to u.quant check)

; beq 1f / been modified? no, 1f

mov byte ptr [mmod], al ; 0

; clrb mmod / yes, clear mmod

;mov ax, word ptr [mntd]

;;mov al, byte ptr [mdev] ; 26/04/2013

mov bx, offset sb1 ;; 07/08//2013

;;mov byte ptr [BX], al

;mov byte ptr [sb1], al

; movb mntd,sb1 / set the I/O queue

or word ptr [BX], 200h

;or word ptr [sb1], 200h ; write bit, bit 9

; bis $1000,sb1 / set write bit in I/O queue for detached sb

call poke ; 07/08/2013

;call ppoke

; jsr r0,ppoke / write it out to its device

;xor al, al ; 26/04/2013

;@@: ;1:

; cmp byte ptr [uquant], al ; 0

; ; tstb uquant / is the time quantum 0?

; ja short @f

; ;ja short swapret

; bne 1f / no, don't swap it out

sysrele: ; < release >

; 07/03/2014

; 23/02/2014

; 14/02/2014 uquant -> u.quant

; 18/01/2014

; 07/12/2013

; 20/10/2013

; 22/09/2013

; 16/05/2013

; 08/05/2013

; 16/04/2013

; 11/04/2013

; 10/04/2013

;

; 'sysrele' first calls 'tswap' if the time quantum for a user is

; zero (see 'sysret'). It then restores the user's registers and

; turns off the system flag. It then checked to see if there is

; an interrupt from the user by calling 'isintr'. If there is,

; the output gets flashed (see isintr) and interrupt action is

; taken by a branch to 'intract'. If there is no interrupt from

; the user, a rti is made.

;

; Calling sequence:

; Fall through a 'bne' in 'sysret' & ?

; Arguments:

; -

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

;

; 23/02/2014 (@@)

; 22/09/2013

@@: ;1:

cmp byte ptr [u.quant], 0 ; 16/05/2013

; tstb uquant / is the time quantum 0?

ja short @f

;ja short swapret

; bne 1f / no, don't swap it out

sysrelease: ; 07/12/2013 (jump from 'clock ')

;

call tswap

; jsr r0,tswap / yes, swap it out

;

; Retro Unix 8086 v1 feature: return from 'swap' to 'swapret' address.

@@:

;swapret: ;1:

; 26/05/2013

; 'sp' must be already equal to 'word ptr [u.sp\_]' here !

;mov sp, word ptr [u.sp\_] ; Retro Unix 8086 v1 modification!

; 10/04/2013

; (If an I/O error occurs during disk I/O,

; related procedures will jump to 'error'

; procedure directly without returning to

; the caller procedure. So, stack pointer

; must be restored here.)

pop bp

pop di

pop si

pop bx

pop cx

pop dx

; mov (sp)+,sc / restore user registers

; mov (sp)+,mq

; mov (sp)+,ac

; mov (sp)+,r5

; mov (sp)+,r4

; mov (sp)+,r3

; mov (sp)+,r2

; 22/09/2013

call isintr

; 20/10/2013

jz short @f

call intract

; jsr r0,isintr / is there an interrupt from the user

; br intract / yes, output gets flushed, take interrupt

; / action

@@:

; mov (sp)+,r1

pop ax ; user's stack pointer

; (was pushed on system stack by 'sysenter'.)

; mov (sp)+,r0

; 24/05/2013

; 18/01/2014

;cli ; disable (hardware) interrupts

mov sp, ax ; user's stack pointer

mov ax, word ptr [u.segmnt]

mov ss, ax ; user's stack segment

; 18/01/2014

;;sti ; enable interrupts ;; 07/03/2014

; 'sti' is not needed here

; (because 'iret' will restore interrupt flag)

mov es, ax

;;;mov ax, word ptr [s.chrgt]+2

;;;mov word ptr [clockp], ax

; 20/10/2013

mov ax, word ptr [u.r0] ; ((return value in AX))

dec byte ptr [sysflg]

; decb sysflg / turn system flag off

push es

pop ds

iret

; rti / no, return from interrupt

badsys:

; 27/05/2013

; 11/04/2013

inc byte ptr [u.bsys]

; incb u.bsys / turn on the user's bad-system flag

mov word ptr [u.namep], offset badsys\_3 ; 3f

; mov $3f,u.namep / point u.namep to "core\0\0"

call namei

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

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

; ax = 0 -> file not found

;jz short badsys\_1

jc short badsys\_1 ; 27/05/2013

; br 1f / error

neg ax ; AX = r1

; neg r1 / negate the i-number to open the core image file

; / for writing

call iopen

; jsr r0,iopen / open the core image file

call itrunc

; jsr r0,itrunc / free all associated blocks

jmp short badsys\_2

; br 2f

badsys\_1: ;1:

mov ax, 15 ; mode 17

; mov $17,r1 / put i-node mode (17) in r1

call maknod

; jsr r0,maknod / make an i-node

mov ax, word ptr [u.dirbuf] ; i-number

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

badsys\_2: ;2:

; 19/04/2013

mov si, offset user

mov di, ecore

mov cx, word ptr [u.segmnt]

mov es, cx

mov cx, 32

rep movsw

mov dx, ds

mov es, dx

mov word ptr [u.base], core

; mov $core,u.base / move address core to u.base

mov word ptr [u.count], ecore - core + 64

; mov $ecore-core,u.count / put the byte count in u.count

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

; mov $u.off,u.fofp / more user offset to u.fofp

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

; clr u.off / clear user offset

call writei

; jsr r0,writei / write out the core image to the user

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

; mov $user,u.base / pt. u.base to user

;mov word ptr [u.count], 64

; mov $64.,u.count / u.count = 64

;call writei

; jsr r0,writei / write out all the user parameters

neg ax ; r1

; neg r1 / make i-number positive

call iclose

; jsr r0,iclose / close the core image file

jmp short sysexit

; br sysexit /

badsys\_3: ;3:

db 'core',0,0

; <core\0\0>

@@: ; 22/09/2013

retn

intract: ; / interrupt action

; 07/12/2013

; 06/12/2013

; 20/10/2013

; 22/09/2013

; 03/09/2013

; 16/05/2013 task/process/tty switch

; 15/05/2013 (ptty, set video page)

; 09/05/2013

; Retro UNIX 8086 v1 modification !

; (Process/task switching and quit routine by using

; Retro UNIX 8086 v1 keyboard interrupt output.))

;

; input -> 'u.quit' (also value of 'u.intr' > 0)

; output -> If value of 'u.quit' = FFFFh ('ctrl+brk' sign)

; 'intract' will jump to 'sysexit'.

; Intract will return to the caller

; if value of 'u.quit' <> FFFFh.

; 07/12/2013

inc word ptr [u.quit]

jz short @f ; FFFFh -> 0

dec word ptr [u.quit]

jmp short @b

@@:

; 20/10/2013

pop ax ; call intract -> retn

pop ax ; user's stack pointer ('sysrele')

;

xor ax, ax

inc al ; mov ax, 1

; 06/12/2013

;mov word ptr [u.quit], ax ; reset to

; 'ctrl+brk' enabled

;jmp sysexit

;;;

; UNIX v1 original 'intract' routine...

; / interrupt action

;cmp \*(sp),$rti / are you in a clock interrupt?

; bne 1f / no, 1f

; cmp (sp)+,(sp)+ / pop clock pointer

; 1: / now in user area

; mov r1,-(sp) / save r1

; mov u.ttyp,r1

; / pointer to tty buffer in control-to r1

; cmpb 6(r1),$177

; / is the interrupt char equal to "del"

; beq 1f / yes, 1f

; clrb 6(r1)

; / no, clear the byte

; / (must be a quit character)

; mov (sp)+,r1 / restore r1

; clr u.quit / clear quit flag

; bis $20,2(sp)

; / set trace for quit (sets t bit of

; / ps-trace trap)

; rti ; / return from interrupt

; 1: / interrupt char = del

; clrb 6(r1) / clear the interrupt byte

; / in the buffer

; mov (sp)+,r1 / restore r1

; cmp u.intr,$core / should control be

; / transferred to loc core?

; blo 1f

; jmp \*u.intr / user to do rti yes,

; / transfer to loc core

; 1:

; sys 1 / exit

sysexit: ; <terminate process>

; 14/02/2014

; 05/02/2014

; 17/09/2013

; 30/08/2013

; 19/04/2013

;

; 'sysexit' terminates a process. First each file that

; the process has opened is closed by 'flose'. The process

; status is then set to unused. The 'p.pid' table is then

; searched to find children of the dying process. If any of

; children are zombies (died by not waited for), they are

; set free. The 'p.pid' table is then searched to find the

; dying process's parent. When the parent is found, it is

; checked to see if it is free or it is a zombie. If it is

; one of these, the dying process just dies. If it is waiting

; for a child process to die, it notified that it doesn't

; have to wait anymore by setting it's status from 2 to 1

; (waiting to active). It is awakened and put on runq by

; 'putlu'. The dying process enters a zombie state in which

; it will never be run again but stays around until a 'wait'

; is completed by it's parent process. If the parent is not

; found, process just dies. This means 'swap' is called with

; 'u.uno=0'. What this does is the 'wswap' is not called

; to write out the process and 'rswap' reads the new process

; over the one that dies..i.e., the dying process is

; overwritten and destroyed.

;

; Calling sequence:

; sysexit or conditional branch.

; Arguments:

; -

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

;

; Retro UNIX 8086 v1 modification:

; System call number (=1) is in AX register.

;

; Other parameters are in DX, BX, CX, SI, DI, BP registers

; depending of function details.

;

; ('swap' procedure is mostly different than original UNIX v1.)

;

; / terminate process

; AX = 1

dec ax ; 0

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

; clr u.intr / clear interrupt control word

; clr r1 / clear r1

; AX = 0

sysexit\_1: ; 1:

; AX = File descriptor

; / r1 has file descriptor (index to u.fp list)

; / Search the whole list

call fclose

; jsr r0,fclose / close all files the process opened

;; ignore error return

; br .+2 / ignore error return

;inc ax

inc al

; inc r1 / increment file descriptor

;cmp ax, 10

cmp al, 10

; cmp r1,$10. / end of u.fp list?

jb short sysexit\_1

; blt 1b / no, go back

xor bh, bh ; 0

mov bl, byte ptr [u.uno]

; movb u.uno,r1 / yes, move dying process's number to r1

mov byte ptr [BX]+p.stat-1, ah ; 0, SFREE, 05/02/2014

; clrb p.stat-1(r1) / free the process

;shl bx, 1

shl bl, 1

; asl r1 / use r1 for index into the below tables

mov cx, word ptr [BX]+p.pid-2

; mov p.pid-2(r1),r3 / move dying process's name to r3

mov dx, word ptr [BX]+p.ppid-2

; mov p.ppid-2(r1),r4 / move its parents name to r4

; xor bx, bx ; 0

xor bl, bl ; 0

; clr r2

xor si, si ; 0

; clr r5 / initialize reg

sysexit\_2: ; 1:

; / find children of this dying process,

; / if they are zombies, free them

;add bx, 2

add bl, 2

; add $2,r2 / search parent process table

; / for dying process's name

cmp word ptr [BX]+p.ppid-2, cx

; cmp p.ppid-2(r2),r3 / found it?

jne short sysexit\_4

; bne 3f / no

;shr bx, 1

shr bl, 1

; asr r2 / yes, it is a parent

cmp byte ptr [BX]+p.stat-1, 3 ; SZOMB, 05/02/2014

; cmpb p.stat-1(r2),$3 / is the child of this

; / dying process a zombie

jne short sysexit\_3

; bne 2f / no

mov byte ptr [BX]+p.stat-1, ah ; 0, SFREE, 05/02/2014

; clrb p.stat-1(r2) / yes, free the child process

sysexit\_3: ; 2:

;shr bx, 1

shl bl, 1

; asl r2

sysexit\_4: ; 3:

; / search the process name table

; / for the dying process's parent

cmp word ptr [BX]+p.pid-2, dx ; 17/09/2013

; cmp p.pid-2(r2),r4 / found it?

jne short sysexit\_5

; bne 3f / no

mov si, bx

; mov r2,r5 / yes, put index to p.pid table (parents

; / process # x2) in r5

sysexit\_5: ; 3:

;cmp bx, nproc + nproc

cmp bl, nproc + nproc

; cmp r2,$nproc+nproc / has whole table been searched?

jb short sysexit\_2

; blt 1b / no, go back

; mov r5,r1 / yes, r1 now has parents process # x2

and si, si ; r5=r1

jz short sysexit\_6

; beq 2f / no parent has been found.

; / The process just dies

shr si, 1

; asr r1 / set up index to p.stat

mov al, byte ptr [SI]+p.stat-1

; movb p.stat-1(r1),r2 / move status of parent to r2

and al, al

jz short sysexit\_6

; beq 2f / if its been freed, 2f

cmp al, 3

; cmp r2,$3 / is parent a zombie?

je short sysexit\_6

; beq 2f / yes, 2f

; BH = 0

mov bl, byte ptr [u.uno]

; movb u.uno,r3 / move dying process's number to r3

mov byte ptr [BX]+p.stat-1, 3

; movb $3,p.stat-1(r3) / make the process a zombie

; 05/02/2014

cmp al, 1 ; SRUN

je short sysexit\_6

;cmp al, 2

; cmp r2,$2 / is the parent waiting for

; / this child to die

;jne short sysexit\_6

; bne 2f / yes, notify parent not to wait any more

; 05/02/2014

; p.stat = 2 --> waiting

; p.stat = 4 --> sleeping

mov byte ptr [SI]+p.stat-1, 1 ; SRUN ; 05/02/2014

;dec byte ptr [SI]+p.stat-1

; decb p.stat-1(r1) / awaken it by putting it (parent)

mov ax, si ; r1 (process number in AL)

; 14/02/2014

;mov bx, offset runq + 4

; mov $runq+4,r2 / on the runq

call putlu

; jsr r0, putlu

sysexit\_6: ; 2:

; / the process dies

mov byte ptr [u.uno], 0

; clrb u.uno / put zero as the process number,

; / so "swap" will

call swap

; jsr r0,swap / overwrite process with another process

; 30/08/2013

;mov sp, word ptr [u.sp\_] ; Retro Unix 8086 v1 modification!

;jmp @b

;;jmp swapret ; Retro UNIX 8086 v1 modification !

hlt\_sys:

;sti ; 18/01/2014

@@:

hlt

;jmp short hlt\_sys

jmp short @b

; 0 / and thereby kill it; halt?

syswait: ; < wait for a processs to die >

; 05/02/2014

; 10/12/2013

; 04/11/2013

; 30/10/2013

; 23/10/2013

; 24/05/2013

; 'syswait' waits for a process die.

; It works in following way:

; 1) From the parent process number, the parent's

; process name is found. The p.ppid table of parent

; names is then searched for this process name.

; If a match occurs, r2 contains child's process

; number. The child status is checked to see if it is

; a zombie, i.e; dead but not waited for (p.stat=3)

; If it is, the child process is freed and it's name

; is put in (u.r0). A return is then made via 'sysret'.

; If the child is not a zombie, nothinh happens and

; the search goes on through the p.ppid table until

; all processes are checked or a zombie is found.

; 2) If no zombies are found, a check is made to see if

; there are any children at all. If there are none,

; an error return is made. If there are, the parent's

; status is set to 2 (waiting for child to die),

; the parent is swapped out, and a branch to 'syswait'

; is made to wait on the next process.

;

; Calling sequence:

; ?

; Arguments:

; -

; Inputs: -

; Outputs: if zombie found, it's name put in u.r0.

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

;

; / wait for a process to die

syswait\_0:

xor bh, bh

mov bl, byte ptr [u.uno]

; movb u.uno,r1 / put parents process number in r1

shl bl, 1

;shl bx, 1

; asl r1 / x2 to get index into p.pid table

mov ax, word ptr [BX]+p.pid-2

; mov p.pid-2(r1),r1 / get the name of this process

xor si, si

; clr r2

xor cx, cx ; 30/10/2013

;xor cl, cl

; clr r3 / initialize reg 3

syswait\_1: ; 1:

add si, 2

; add $2,r2 / use r2 for index into p.ppid table

; / search table of parent processes

; / for this process name

cmp ax, word ptr [SI]+p.ppid-2

; cmp p.ppid-2(r2),r1 / r2 will contain the childs

; / process number

jne short syswait\_3

;bne 3f / branch if no match of parent process name

;inc cx

inc cl

;inc r3 / yes, a match, r3 indicates number of children

shr si, 1

; asr r2 / r2/2 to get index to p.stat table

; The possible states ('p.stat' values) of a process are:

; 0 = free or unused

; 1 = active

; 2 = waiting for a child process to die

; 3 = terminated, but not yet waited for (zombie).

cmp byte ptr [SI]+p.stat-1, 3 ; SZOMB, 05/02/2014

; cmpb p.stat-1(r2),$3 / is the child process a zombie?

jne short syswait\_2

; bne 2f / no, skip it

mov byte ptr [SI]+p.stat-1, bh ; 0

; clrb p.stat-1(r2) / yes, free it

shl si, 1

; asl r2 / r2x2 to get index into p.pid table

mov ax, word ptr [SI]+p.pid-2

mov word ptr [u.r0], ax

; mov p.pid-2(r2),\*u.r0

; / put childs process name in (u.r0)

jmp sysret

; br sysret1 / return cause child is dead

syswait\_2: ; 2:

shl si, 1

; asl r2 / r2x2 to get index into p.ppid table

syswait\_3: ; 3:

cmp si, nproc+nproc

; cmp r2,$nproc+nproc / have all processes been checked?

jb syswait\_1

; blt 1b / no, continue search

;and cx, cx

and cl, cl

; tst r3 / one gets here if there are no children

; / or children that are still active

; 30/10/2013

jnz short @f

;jz error

; beq error1 / there are no children, error

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

jmp error

@@:

mov bl, byte ptr [u.uno]

; movb u.uno,r1 / there are children so put

; / parent process number in r1

inc byte ptr [BX]+p.stat-1 ; 2, SWAIT, 05/02/2014

; incb p.stat-1(r1) / it is waiting for

; / other children to die

; 04/11/2013

call swap

; jsr r0,swap / swap it out, because it's waiting

jmp syswait\_0

; br syswait / wait on next process

sysfork: ; < create a new process >

; 14/02/2014

; 05/02/2014

; 07/12/2013

; 06/12/2013

; 18/11/2013

; 17/09/2013

; 16/09/2013

; 30/08/2013

; 08/08/2013

; 22/07/2013

; 26/05/2013

; 24/05/2013

; 'sysfork' creates a new process. This process is referred

; to as the child process. This new process core image is

; a copy of that of the caller of 'sysfork'. The only

; distinction is the return location and the fact that (u.r0)

; in the old process (parent) contains the process id (p.pid)

; of the new process (child). This id is used by 'syswait'.

; 'sysfork' works in the following manner:

; 1) The process status table (p.stat) is searched to find

; a process number that is unused. If none are found

; an error occurs.

; 2) when one is found, it becomes the child process number

; and it's status (p.stat) is set to active.

; 3) If the parent had a control tty, the interrupt

; character in that tty buffer is cleared.

; 4) The child process is put on the lowest priority run

; queue via 'putlu'.

; 5) A new process name is gotten from 'mpid' (actually

; it is a unique number) and is put in the child's unique

; identifier; process id (p.pid).

; 6) The process name of the parent is then obtained and

; placed in the unique identifier of the parent process

; name is then put in 'u.r0'.

; 7) The child process is then written out on disk by

; 'wswap',i.e., the parent process is copied onto disk

; and the child is born. (The child process is written

; out on disk/drum with 'u.uno' being the child process

; number.)

; 8) The parent process number is then restored to 'u.uno'.

; 9) The child process name is put in 'u.r0'.

; 10) The pc on the stack sp + 18 is incremented by 2 to

; create the return address for the parent process.

; 11) The 'u.fp' list as then searched to see what files

; the parent has opened. For each file the parent has

; opened, the corresponding 'fsp' entry must be updated

; to indicate that the child process also has opened

; the file. A branch to 'sysret' is then made.

;

; Calling sequence:

; from shell ?

; Arguments:

; -

; Inputs: -

; Outputs: \*u.r0 - child process name

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

;

; Retro UNIX 8086 v1 modification:

; AX = r0 = PID (>0) (at the return of 'sysfork')

; = process id of child a parent process returns

; = process id of parent when a child process returns

;

; In original UNIX v1, sysfork is called and returns as

; in following manner: (with an example: c library, fork)

;

; 1:

; sys fork

; br 1f / child process returns here

; bes 2f / parent process returns here

; / pid of new process in r0

; rts pc

; 2: / parent process condionally branches here

; mov $-1,r0 / pid = -1 means error return

; rts pc

;

; 1: / child process brances here

; clr r0 / pid = 0 in child process

; rts pc

;

; In UNIX v7x86 (386) by Robert Nordier (1999)

; // pid = fork();

; //

; // pid == 0 in child process;

; // pid == -1 means error return

; // in child,

; // parents id is in par\_uid if needed

;

; \_fork:

; mov $.fork,eax

; int $0x30

; jmp 1f

; jnc 2f

; jmp cerror

; 1:

; mov eax,\_par\_uid

; xor eax,eax

; 2:

; ret

;

; In Retro UNIX 8086 v1,

; 'sysfork' returns in following manner:

;

; mov ax, sys\_fork

; mov bx, offset @f ; routine for child

; int 20h

; jc error

;

; ; Routine for parent process here (just after 'jc')

; mov word ptr [pid\_of\_child], ax

; jmp next\_routine\_for\_parent

;

; @@: ; routine for child process here

; ....

; NOTE: 'sysfork' returns to specified offset

; for child process by using BX input.

; (at first, parent process will return then

; child process will return -after swapped in-

; 'syswait' is needed in parent process

; if return from child process will be waited for.)

;

; / create a new process

; BX = return address for child process

; (Retro UNIX 8086 v1 modification !)

xor si, si

; clr r1

sysfork\_1: ; 1: / search p.stat table for unused process number

inc si

; inc r1

cmp byte ptr [SI]+p.stat-1, 0 ; SFREE, 05/02/2014

; tstb p.stat-1(r1) / is process active, unused, dead

jna short sysfork\_2

; beq 1f / it's unused so branch

cmp si, nproc

; cmp r1,$nproc / all processes checked

jb short sysfork\_1 ; 08/08/2013

; blt 1b / no, branch back

; Retro UNIX 8086 v1. modification:

; Parent process returns from 'sysfork' to address

; which is just after 'sysfork' system call in parent

; process. Child process returns to address which is put

; in BX register by parent process for 'sysfork'

; system call.

; so, it is not needed to increment return address

; of system call on the top of the user's stack.

; If the routine would be same with original UNIX v1

; 'sysfork' routine, 'add word ptr [SP]+12, 2'

; instruction would be put here.

;; add word ptr [SP]+12, 2

;; jmp error

;add $2,18.(sp) / add 2 to pc when trap occured, points

; / to old process return

; br error1 / no room for a new process

jmp error ; 08/08/2013

sysfork\_2: ; 1:

; Retro UNIX 8086 v1. modification !

; 08/08/2013

mov ax, offset sysret

push ax ; \*

mov word ptr [u.usp], sp

;;push es

; 08/08/2013

; Return address for the parent process is already set

; by sysenter routine.

;mov ax, word ptr [u.segmnt]

;mov es, ax

;mov bp, sp

;mov di, word ptr [BP]+12 ; user's stack pointer

;;pop es

;push word ptr ES:[DI]

;;;mov ax, word ptr ES:[DI] ; return address (IP)

;;;push ax ; \*\*\*\* return address for the parent process

;;mov ax, cs

;;mov es, ax

;;

push word ptr [u.segmnt] ; \*\*

; Retro UNIX 8086 v1 feature only !

;

; 06/12/2013

;push word ptr [u.uno] ; \*\*\*

; movb u.uno,-(sp) / save parent process number

xor ah, ah

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

push ax ; \*\*\*

mov di, ax

; 07/12/2013

mov al, byte ptr [DI]+p.ttyc-1 ; console tty (parent)

mov byte ptr [SI]+p.ttyc-1, al ; set child's console tty

; 05/02/2014 (p.ttys has been removed)

;mov byte ptr [SI]+p.ttys-1, al ; set parent's console tty

mov byte ptr [SI]+p.waitc-1, al ; set parent's console tty

; 22/07/2013

mov ax, si

mov byte ptr [u.uno], al

;

;mov word ptr [u.uno], si

;movb r1,u.uno / set child process number to r1

inc byte ptr [SI]+p.stat-1 ; 1, SRUN, 05/02/2014

; incb p.stat-1(r1) / set p.stat entry for child

; / process to active status

; mov u.ttyp,r2 / put pointer to parent process'

; / control tty buffer in r2

;;and di, di

;;jz short sysfork\_3

; beq 2f / branch, if no such tty assigned

;; ????

; clrb 6(r2) / clear interrupt character in tty buffer

sysfork\_3: ; 2:

push bx ; \* return address for the child process

; \* Retro UNIX 8086 v1 feature only !

;;mov ax, si ;; 22/07/2013

; 14/02/2014

;mov bx, offset runq + 2 ; middle priority !

; (Retro UNIX 8086 v1 modification!)

; mov $runq+4,r2

call putlu

; jsr r0,putlu / put child process on lowest priority

; / run queue

shl si, 1

; asl r1 / multiply r1 by 2 to get index

; / into p.pid table

inc word ptr [mpid]

; inc mpid / increment m.pid; get a new process name

mov ax, word ptr [mpid]

mov word ptr [SI]+p.pid-2, ax

;mov mpid,p.pid-2(r1) / put new process name

; / in child process' name slot

pop dx ; \* return address for the child process

; \* Retro UNIX 8086 v1 feature only !

; 08/08//2013

pop bx ; \*\*\*

push bx ; \*\*\*

;mov bp, sp

;mov bx, word ptr [BP] ; \*\*\*

; movb (sp),r2 / put parent process number in r2

xor bh, bh ; 08/08/2013

shl bx, 1

;asl r2 / multiply by 2 to get index into below tables

mov ax, word ptr [BX]+p.pid-2

; mov p.pid-2(r2),r2 / get process name of parent

; / process

mov word ptr [SI]+p.ppid-2, ax

; mov r2,p.ppid-2(r1) / put parent process name

; / in parent process slot for child

mov word ptr [u.r0], ax

; mov r2,\*u.r0 / put parent process name on stack

; / at location where r0 was saved

; 22/07/2013

call segm\_sw ; User segment switch

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

;

mov ax, word ptr [u.segmnt] ; 08/08/2013

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

mov es, bx

xor si, si

xor di, di

mov cx, 16384

mov ds, ax ; 08/08/2013

rep movsw ; copy process (in current segment) to

; new process segment

; 08/08/2013

mov ax, cs

mov ds, ax

mov ax, bx ; new user segment

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

mov bx, word ptr [BP]+12 ; user's stack pointer

mov word ptr ES:[BX], dx ; \*, CS:IP -> IP

; \* return address for the child process

mov word ptr ES:[BX]+2, ax ; CS:IP -> CS

; \* return address for the child process

;mov ax, cs

;mov es, ax

;\*

;;mov ax, offset sysret

;;push ax ; \*

; mov $sysret1,-(sp) /

;mov word ptr [u.usp], sp

; mov sp,u.usp / contents of sp at the time when

; / user is swapped out

; mov $sstack,sp / point sp to swapping stack space

; ES = u.segmnt

; 06/12/2013

;push word ptr [u.intr] ; \*\*\*\*

; 30/08/2013

push word ptr [u.ttyp] ; \*\*\*\*\*

xor ax, ax

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

;

call wswap ; Retro UNIX 8086 v1 modification !

;jsr r0,wswap / put child process out on drum

;jsr r0,unpack / unpack user stack

;mov u.usp,sp / restore user stack pointer

; ES = DS

;;mov sp, word ptr [u.usp]

; 30/08/2013

pop word ptr [u.ttyp] ; \*\*\*\*\*

; 06/12/2013

;pop word ptr [u.intr] ; \*\*\*\*

;;pop ax ; \*

; tst (sp)+ / bump stack pointer

;pop word ptr [u.uno] ; \*\*\*

pop ax ; \*\*\* 22/07/2013

mov byte ptr [u.uno], al

;movb (sp)+,u.uno / put parent process number in u.uno

;

pop word ptr [u.segmnt] ; \*\*

; Retro UNIX 8086 v1 feature only !

;

mov ax, word ptr [mpid]

mov word ptr [u.r0], ax

; mov mpid,\*u.r0 / put child process name on stack

; / where r0 was saved

; 08/08/2013

; Return address for the parent process is already set

; by sysenter routine.

;pop dx ; \*\*\*\* return address for the parent process

;mov ax, word ptr [u.segmnt]

;mov es, ax

;mov word ptr ES:[BX]+2, ax ; user's CS for iret <- ax

;mov word ptr ES:[BX], dx ; user's IP for iret <- dx

; add $2,18.(sp) / add 2 to pc on stack; gives parent

; / process return

;pop ax ; \* 08/08/2013

;

xor si, si

;clr r1

sysfork\_4: ; 1: / search u.fp list to find the files

; / opened by the parent process

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

; movb u.fp(r1),r2 / get an open file for this process

or bl, bl

jz short sysfork\_5

; beq 2f / file has not been opened by parent,

; / so branch

xor bh, bh ; 18/11/2013

shl bx, 1

; asl r2 / multiply by 8

shl bx, 1

; asl r2 / to get index into fsp table

shl bx, 1

; asl r2

inc byte ptr [BX]+fsp-2

; incb fsp-2(r2) / increment number of processes

; / using file, because child will now be

; / using this file

sysfork\_5: ; 2:

inc si

; inc r1 / get next open file

cmp si, 10

; cmp r1,$10. / 10. files is the maximum number which

; / can be opened

jb short sysfork\_4

; blt 1b / check next entry

; 08/08/2013

retn ; \* -> sysret

;jmp sysret

; br sysret1

segm\_sw:

; 24/07/2013

; 23/07/2013

; 22/07/2013

; Retro UNIX 8086 v1 feature only !

; (User segment switch)

; INPUT -> none

; OUTPUT -> bx = new user segment

; (word ptr [u.segmnt] = ax)

; ((Modified registers: cx))

;

mov cl, byte ptr [u.uno] ; 23/07/2013

mov bx, csgmnt ; segment of process 1

@@:

dec cl

jz short @f

add bx, 2048 ; (32768/16)

jmp short @b

@@:

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

retn

sysread: ; < read from file >

; 23/05/2013

; 'sysread' is given a buffer to read into and the number of

; characters to be read. If finds the file from the file

; descriptor located in \*u.r0 (r0). This file descriptor

; is returned from a successful open call (sysopen).

; The i-number of file is obtained via 'rw1' and the data

; is read into core via 'readi'.

;

; Calling sequence:

; sysread; buffer; nchars

; Arguments:

; buffer - location of contiguous bytes where

; input will be placed.

; nchars - number of bytes or characters to be read.

; Inputs: \*u.r0 - file descriptor (& arguments)

; Outputs: \*u.r0 - number of bytes read.

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

;

; Retro UNIX 8086 v1 modification:

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

; Retro UNIX 8086 v1 argument transfer method 3 is used

; to get sysread system call arguments from the user;

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

; \* 2nd argument, buffer address/offset in CX register

; \* 3rd argument, number of bytes is in DX register

;

; AX register (will be restored via 'u.r0') will return

; to the user with number of bytes read.

;

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

; arguments in these registers;

; (BX= file descriptor)

; (CX= buffer address in user's program segment)

; (DX= number of bytes)

; then

; \* file descriptor (in BX) is moved into AX

; \* buffer address (in CX) is moved into 'u.base'.

; \* byte count (in DX) is moved into 'u.count'.

;

call rw1

; jsr r0,rw1 / get i-number of file to be read into r1

test ah, 80h

; tst r1 / negative i-number?

jnz error

; ble error1 / yes, error 1 to read

; / it should be positive

call readi

; jsr r0,readi / read data into core

jmp short @f

; br 1f

syswrite: ; < write to file >

; 23/05/2013

; 'syswrite' is given a buffer to write onto an output file

; and the number of characters to write. If finds the file

; from the file descriptor located in \*u.r0 (r0). This file

; descriptor is returned from a successful open or create call

; (sysopen or syscreat). The i-number of file is obtained via

; 'rw1' and buffer is written on the output file via 'write'.

;

; Calling sequence:

; syswrite; buffer; nchars

; Arguments:

; buffer - location of contiguous bytes to be writtten.

; nchars - number of characters to be written.

; Inputs: \*u.r0 - file descriptor (& arguments)

; Outputs: \*u.r0 - number of bytes written.

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

; Retro UNIX 8086 v1 modification:

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

; Retro UNIX 8086 v1 argument transfer method 3 is used

; to get syswrite system call arguments from the user;

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

; \* 2nd argument, buffer address/offset in CX register

; \* 3rd argument, number of bytes is in DX register

;

; AX register (will be restored via 'u.r0') will return

; to the user with number of bytes written.

;

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

; arguments in these registers;

; (BX= file descriptor)

; (CX= buffer address in user's program segment)

; (DX= number of bytes)

; then

; \* file descriptor (in BX) is moved into AX

; \* buffer address (in CX) is moved into 'u.base'.

; \* byte count (in DX) is moved into 'u.count'.

call rw1

; jsr r0,rw1 / get i-number in r1 of file to write

test ah, 80h

; tst r1 / positive i-number ?

jz error

; bge error1 / yes, error 1

; / negative i-number means write

neg ax

; neg r1 / make it positive

call writei

; jsr r0,writei / write data

@@: ; 1:

mov ax, word ptr [u.nread]

mov word ptr [u.r0], ax

; mov u.nread,\*u.r0 / put no. of bytes transferred

; / into (u.r0)

jmp sysret

; br sysret1

rw1: ; 23/05/2013

; 'rw1' returns i-number of the file for 'sysread' & 'syswrite'.

; Retro UNIX 8086 v1 modification:

; 'arg' routine is different than 'arg' in original Unix v1.

;mov ax, 3 ; number of arguments

;call arg

; 24/05/2013

; System call registers: bx, cx, dx (through 'sysenter')

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

; (in the user's program segment)

mov word ptr [u.count], dx

;

; jsr r0,arg; u.base / get buffer pointer

; jsr r0,arg; u.count / get no. of characters

;;mov ax, bx ; file descriptor

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

; / (index to u.fp table) in r1

;; call getf

; BX = File descriptor

call getf1 ; calling point in 'getf' from 'rw1'

; jsr r0,getf / get i-number of the file in r1

; AX = I-number of the file ; negative i-number means write

retn

; rts r0

sysopen: ;<open file>

; 27/05/2013

; 24/05/2013

; 22/05/2013

; 'sysopen' opens a file in following manner:

; 1) The second argument in a sysopen says whether to

; open the file ro read (0) or write (>0).

; 2) I-node of the particular file is obtained via 'namei'.

; 3) The file is opened by 'iopen'.

; 4) Next housekeeping is performed on the fsp table

; and the user's open file list - u.fp.

; a) u.fp and fsp are scanned for the next available slot.

; b) An entry for the file is created in the fsp table.

; c) The number of this entry is put on u.fp list.

; d) The file descriptor index to u.fp list is pointed

; to by u.r0.

;

; Calling sequence:

; sysopen; name; mode

; Arguments:

; name - file name or path name

; mode - 0 to open for reading

; 1 to open for writing

; Inputs: (arguments)

; Outputs: \*u.r0 - index to u.fp list (the file descriptor)

; is put into r0's location on the stack.

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

;

; Retro UNIX 8086 v1 modification:

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

; Retro UNIX 8086 v1 argument transfer method 2 is used

; to get sysopen system call arguments from the user;

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

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

;

; AX register (will be restored via 'u.r0') will return

; to the user with the file descriptor/number

; (index to u.fp list).

;

; 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).

; \* mode - sysopen argument 2 is in CX register

; which is on top of stack.

;

; jsr r0,arg2 / get sys args into u.namep and on stack

; 24/05/2013

; system call registers: bx, cx (through 'sysenter')

mov word ptr [u.namep], bx

push cx

call namei

; jsr r0,namei / i-number of file in r1

;and ax, ax

;jz error ; File not found

jc error ; 27/05/2013

; br error2 / file not found

pop dx ; mode

push dx

;or dx, dx

or dl, dl

; tst (sp) / is mode = 0 (2nd arg of call;

; / 0 means, open for read)

jz short @f

; beq 1f / yes, leave i-number positive

neg ax

; neg r1 / open for writing so make i-number negative

@@: ;1:

call iopen

;jsr r0,iopen / open file whose i-number is in r1

pop dx

;and dx, dx

and dl, dl

; tst (sp)+ / pop the stack and test the mode

jz short @f

; beq op1 / is open for read op1

op0:

neg ax

; neg r1

;/ make i-number positive if open for writing [???]

;; NOTE: iopen always make i-number positive.

;; Here i-number becomes negative again

;; perhaps iclose then makes it positive ??? E. Tan [22/05/2013]

@@: ;op1:

xor si, si

; clr r2 / clear registers

xor bx, bx

; clr r3

@@: ;1: / scan the list of entries in fsp table

cmp byte ptr [SI]+u.fp, bl ; 0

; tstb u.fp(r2) / test the entry in the u.fp list

jna short @f

; beq 1f / if byte in list is 0 branch

inc si

; inc r2 / bump r2 so next byte can be checked

cmp si, 10

; cmp r2,$10. / reached end of list?

jb short @b

; blt 1b / no, go back

jmp error

; br error2 / yes, error (no files open)

@@: ; 1:

cmp word ptr [BX]+fsp, 0

; tst fsp(r3) / scan fsp entries

jna short @f

; beq 1f / if 0 branch

add bx, 8

; add $8.,r3 / add 8 to r3

; / to bump it to next entry mfsp table

cmp bx, nfiles\*8

; cmp r3,$[nfiles\*8.] / done scanning

jb short @b

; blt 1b / no, back

jmp error

; br error2 / yes, error

@@: ; 1: / r2 has index to u.fp list; r3, has index to fsp table

mov word ptr [BX]+fsp, ax

; mov r1,fsp(r3) / put i-number of open file

; / into next available entry in fsp table,

mov di, word ptr [cdev] ; word ? byte ?

mov word ptr [BX]+fsp+2, di

; mov cdev,fsp+2(r3) / put # of device in next word

xor di, di

mov word ptr [BX]+fsp+4, di

; clr fsp+4(r3)

mov word ptr [BX]+fsp+6, di

; clr fsp+6(r3) / clear the next two words

shr bx, 1

; asr r3

shr bx, 1

; asr r3 / divide by 8

shr bx, 1

; asr r3 ; / to get number of the fsp entry-1

;inc bx

inc bl

; inc r3 / add 1 to get fsp entry number

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

; movb r3,u.fp(r2) / move entry number into

; / next available slot in u.fp list

mov word ptr [u.r0], si

; mov r2,\*u.r0 / move index to u.fp list

; / into r0 loc on stack

jmp sysret

; br sysret2

syscreat: ; < create file >

; 27/05/2013

; 'syscreat' called with two arguments; name and mode.

; u.namep points to name of the file and mode is put

; on the stack. 'namei' is called to get i-number of the file.

; If the file aready exists, it's mode and owner remain

; unchanged, but it is truncated to zero length. If the file

; did not exist, an i-node is created with the new mode via

; 'maknod' whether or not the file already existed, it is

; open for writing. The fsp table is then searched for a free

; entry. When a free entry is found, proper data is placed

; in it and the number of this entry is put in the u.fp list.

; The index to the u.fp (also know as the file descriptor)

; is put in the user's r0.

;

; Calling sequence:

; syscreate; name; mode

; Arguments:

; name - name of the file to be created

; mode - mode of the file to be created

; Inputs: (arguments)

; Outputs: \*u.r0 - index to u.fp list

; (the file descriptor of new file)

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

; Retro UNIX 8086 v1 modification:

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

; Retro UNIX 8086 v1 argument transfer method 2 is used

; to get syscreate system call arguments from the user;

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

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

;

; AX register (will be restored via 'u.r0') will return

; to the user with the file descriptor/number

; (index to u.fp list).

;

; 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).

; \* mode - sysopen argument 2 is in CX register

; which is on top of stack.

; jsr r0,arg2 / put file name in u.namep put mode

; / on stack

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

push cx ; mode

call namei

; jsr r0,namei / get the i-number

;and ax, ax

;jz short @f

jc short @f

; br 2f / if file doesn't exist 2f

neg ax

; neg r1 / if file already exists make i-number

; / negative (open for writing)

call iopen

; jsr r0,iopen /

call itrunc

; jsr r0,itrunc / truncate to 0 length

pop cx ; pop mode (did not exist in original Unix v1 !?)

jmp short op0

; br op0

@@: ; 2: / file doesn't exist

pop ax

; mov (sp)+,r1 / put the mode in r1

xor ah, ah

; bic $!377,r1 / clear upper byte

call maknod

; jsr r0,maknod / make an i-node for this file

mov ax, word ptr [u.dirbuf]

; mov u.dirbuf,r1 / put i-number

; / for this new file in r1

jmp short op0

; br op0 / open the file

sysmkdir: ; < make directory >

; 02/08/2013

; 27/05/2013

; 'sysmkdir' creates an empty directory whose name is

; pointed to by arg 1. The mode of the directory is arg 2.

; The special entries '.' and '..' are not present.

; Errors are indicated if the directory already exists or

; user is not the super user.

;

; Calling sequence:

; sysmkdir; name; mode

; Arguments:

; name - points to the name of the directory

; mode - mode of the directory

; Inputs: (arguments)

; Outputs: -

; (sets 'directory' flag to 1;

; 'set user id on execution' and 'executable' flags to 0)

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

;

; Retro UNIX 8086 v1 modification:

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

; Retro UNIX 8086 v1 argument transfer method 2 is used

; to get sysmkdir system call arguments from the user;

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

; \* 2nd argument, mode is in 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.

; / make a directory

;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).

; \* mode - sysopen argument 2 is in CX register

; which is on top of stack.

; jsr r0,arg2 / put file name in u.namep put mode

; / on stack

mov word ptr [u.namep], bx

push cx

call namei

; jsr r0,namei / get the i-number

; br .+4 / if file not found branch around error

;xor ax, ax

;jnz error

jnc error

; br error2 / directory already exists (error)

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

;tstb u.uid / is user the super user

jna error

;bne error2 / no, not allowed

pop ax

;mov (sp)+,r1 / put the mode in r1

and ax, 0FFCFh ; 1111111111001111b

;bic $!317,r1 / all but su and ex

;or ax , 4000h ; 1011111111111111b

or ah, 40h ; Set bit 14 to 1

;bis $40000,r1 / directory flag

call maknod

;jsr r0,maknod / make the i-node for the directory

jmp sysret

;br sysret2 /

sysclose: ;<close file>

; 26/05/2013

; 22/05/2013

; 'sysclose', given a file descriptor in 'u.r0', closes the

; associated file. The file descriptor (index to 'u.fp' list)

; is put in r1 and 'fclose' is called.

;

; Calling sequence:

; sysclose

; Arguments:

; -

; Inputs: \*u.r0 - file descriptor

; Outputs: -

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

;

; Retro UNIX 8086 v1 modification:

; The user/application program puts file descriptor

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

; (argument transfer method 1)

; / close the file

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

;;call arg

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

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

;mov ax, word ptr [BX]

; mov \*u.r0,r1 / move index to u.fp list into r1

mov ax, bx ; 26/05/2013

call fclose

; jsr r0,fclose / close the file

jc error

; br error2 / unknown file descriptor

jmp sysret

; br sysret2

sysemt:

; 10/04/2014 Bugfix [u.uid --> u.uid\_]

; 18/01/2014

; 10/12/2013

; Retro UNIX 8086 v1 modification:

; 'Enable Multi Tasking' system call instead

; of 'Emulator Trap' in original UNIX v1 for PDP-11.

;

; Retro UNIX 8086 v1 feature only!

; Using purpose: Kernel will start without time-out

; (internal clock/timer) functionality.

; Then etc/init will enable clock/timer for

; multi tasking. (Then it will not be disabled again

; except hardware reset/restart.)

cmp byte ptr [u.uid\_], 0 ; BugFix u.uid --> u.uid\_

ja error

push es

xor ax, ax

mov es, ax ; 0

mov di, 28\*4 ; INT 1Ch vector - offset

; 18/01/2014

cli

and bx, bx

jz short emt\_2

; Enable INT 1Ch time-out functionality.

mov ax, offset clock

emt\_1:

stosw ; offset

mov ax, cs

stosw ; segment

; 18/01/2014

sti

pop es

jmp sysret

emt\_2:

; Disable INT 1Ch time-out functionality.

mov ax, offset emt\_iret

jmp short emt\_1

emt\_iret:

iret

; Original UNIX v1 'sysemt' routine

;sysemt:

;

;jsr r0,arg; 30 / put the argument of the sysemt call

; / in loc 30

;cmp 30,$core / was the argument a lower address

; / than core

;blo 1f / yes, rtssym

;cmp 30,$ecore / no, was it higher than "core"

; / and less than "ecore"

;blo 2f / yes, sysret2

;1:

;mov $rtssym,30

;2:

;br sysret2

sysilgins:

; 03/06/2013,

; Retro UNIX 8086 v1 modification:

; not a valid system call ! (not in use)

;

jmp error

;jmp sysret

; Original UNIX v1 'sysemt' routine

;sysilgins: / calculate proper illegal instruction trap address

;jsr r0,arg; 10 / take address from sysilgins call

;/ put it in loc 8.,

;cmp 10,$core / making it the illegal instruction

; / trap address

;blo 1f / is the address a user core address?

; / yes, go to 2f

;cmp 10,$ecore

;blo 2f

;1:

;mov $fpsym,10 / no, make 'fpsum' the illegal

; / instruction trap address for the system

;2:

;br sysret2 / return to the caller via 'sysret'

sysmdate: ; < change the modification time of a file >

; 02/08/2013

; 03/06/2013

; 'sysmdate' is given a file name. It gets inode of this

; file into core. The user is checked if he is the owner

; or super user. If he is neither an error occurs.

; 'setimod' is then called to set the i-node modification

; byte and the modification time, but the modification time

; is overwritten by whatever get put on the stack during

; a 'systime' system call. This calls are restricted to

; the super user.

;

; Calling sequence:

; sysmdate; name

; Arguments:

; name - points to the name of file

; Inputs: (arguments)

; Outputs: -

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

;

; Retro UNIX 8086 v1 modification:

; The user/application program puts address

; of the file name in BX register

; as 'sysmdate' system call argument.

;

; / change the modification time of a file

; jsr r0,arg; u.namep / point u.namep to the file name

mov word ptr [u.namep], bx

call namei

; jsr r0,namei / get its i-number

jc error

; br error2 / no, such file

call iget

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

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

cmp al, byte ptr [i.uid]

; cmpb u.uid,i.uid / is user same as owner

je short @f

; beq 1f / yes

and al, al

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

jnz error

; bne error2 / no, error

@@: ;1:

call setimod

; jsr r0,setimod / fill in modification data,

; / time etc.

; Retro UNIX 8086 v1 modification !

mov si, offset p\_time

mov di, offset i.mtim

movsw

movsw

; mov 4(sp),i.mtim / move present time to

; mov 2(sp),i.mtim+2 / modification time

jmp sysret

; br sysret2

@@:

retn

sysstty: ; < set tty status and mode >

; 12/07/2014

; 04/07/2014

; 26/06/2014

; 15/04/2014

; 18/01/2014

; 17/01/2014

; 16/01/2014

; 14/01/2014

; 13/01/2014

; 12/01/2014

; 07/12/2013

; 04/12/2013

; 30/10/2013

; 24/10/2013

; 03/09/2013

; 19/08/2013

; 15/08/2013 (set console tty)

; 11/08/2013

; 16/07/2013

; 15/07/2013

; 02/06/2013

;

; 'sysstty' sets the status and mode of the typewriter

; whose file descriptor is in (u.r0).

;

; Calling sequence:

; sysstty; arg

; Arguments:

; arg - address of 3 consequitive words that contain

; the source of status data

; Inputs: ((\*u.r0 - file descriptor & argument))

; Outputs: ((status in address which is pointed to by arg))

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

;

; Retro UNIX 8086 v1 modification:

; 'sysstty' system call will set the tty

; (clear keyboard buffer and set cursor position)

; in following manner:

; NOTE: All of tty setting functions are here (16/01/2014)

;

; Inputs:

; BX = 0 --> means

; If CH = 0

; set console tty for (current) process

; CL = tty number (0 to 9)

; (If ch = 0, character will not be written)

; If CH > 0

; set cursor position or comm. parameters only

; If CL = FFh

; set cursor position for console tty

; or CL = tty number (0 to 9)

; CH = character will be written

; at requested cursor position (in DX)

; (For tty numbers 0 to 7, if CH = FFh, character

; will not be written)

; DX = cursor position for tty number 0 to 7.

; (only tty number 0 to 7)

; DL = communication parameters (for serial ports)

; (only for COM1 and COM2 serial ports)

; DH < 0FFh -> DL is valid, initialize serial port

; or set cursor position

; DH = 0FFh -> DL is not valid

; do not set serial port parameters

; or do not set cursor position

;

; BX > 0 --> points to name of tty

; CH > 0 -->

; CL = character will be written in current

; cursor position (for tty number from 0 to 7)

; or character will be sent to serial port

; (for tty number 8 or 9)

; CH = color of the character if tty number < 8.

; CH = 0 --> Do not write a character,

; set mode (tty 8 to 9) or

; set current cursor positions (tty 0 to 7) only.

; DX = cursor position for tty number 0 to 7.

; DH = FFh --> Do not set cursor pos (or comm. params.)

; (DL is not valid)

; DL = communication parameters

; for tty number 8 or 9 (COM1 or COM2).

; Outputs:

; cf = 0 -> OK

; AL = tty number (0 to 9)

; AH = line status if tty number is 8 or 9

; AH = process number (of the caller)

; cf = 1 means error (requested tty is not ready)

; AH = FFh if the tty is locked

; (owned by another process)

; = process number (of the caller)

; (if < FFh and tty number < 8)

; AL = tty number (0FFh if it does not exist)

; AH = line status if tty number is 8 or 9

; NOTE: Video page will be cleared if cf = 0.

;

; 14/01/2014

mov word ptr [u.r0], 0FFFFh

and bx, bx

jnz sysstty\_6

; set console tty

; 17/01/2014

cmp cl, 9

jna short sysstty\_0

or ch, ch

jz error

cmp cl, 0FFh

jb error

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

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

sysstty\_0:

cmp cl, 8

jb short sysstty\_2

;

cmp dh, 0FFh

je short sysstty\_2

; set communication parameters for serial ports

mov si, offset com1p

; 12/07/2014

cmp cl, 9

jb short sysstty\_1

inc si

sysstty\_1:

mov byte ptr [SI], dl ; comm. parameters

sysstty\_2:

push dx

push cx

xor dl, dl ; sysstty call sign

mov al, cl

mov byte ptr [u.r0], al

; AH = 0

;cbw

; ah = 0

call ottyp

pop cx

pop dx

;

jc error

xor bh, bh

; 17/01/2014

and ch, ch ; set cursor position

; or comm. parameters ONLY

jnz short sysstty\_3

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

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

sysstty\_3:

; 16/01/2014

mov al, ch ; character ; 0 to FFh

cmp cl, 7

jna short sysstty\_9

sysstty\_12:

;; BX = 0, CL = 8 or CL = 9

; (Set specified serial port as console tty port)

; CH = character to be written

; 15/04/2014

; CH = 0 --> initialization only

; AL = character

; 26/06/2014

mov byte ptr [u.ttyn], cl

; 12/07/2014

mov ah, cl ; tty number (8 or 9)

and al, al

jz short sysstty\_4 ; al = ch = 0

; 04/07/2014

call sndc

; 12/07/2014

jmp short sysstty\_5

sysstty\_4:

; 12/07/2014

xchg ah, al ; al = 0 -> al = ah, ah = 0

sub al, 8

mov dx, ax ; 0 or 1

mov ah, 3 ; Get serial port status

int 14h

sysstty\_5:

mov byte ptr [u.r0]+1, ah ; line status

pushf

xor dl, dl ; sysstty call sign

mov al, byte ptr [u.ttyn] ; 26/06/2014

cbw ; ax = tty number (ah=0)

call cttyp

popf

jc error

jmp sysret

sysstty\_6:

push dx

push cx

mov word ptr [u.namep], bx

call namei

pop cx

pop dx

jc error

cmp ax, 19 ; inode number of /dev/COM2

ja error

cmp al, 10 ; /dev/tty0 .. /dev/tty7

; /dev/COM1, /dev/COM2

jb short sysstty\_7

sub al, 10

jmp short sysstty\_8

sysstty\_7:

cmp al, 1 ; /dev/tty

jne error

xor bh, bh

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

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

sysstty\_8:

mov byte ptr [u.r0], al

push dx

push ax

push cx

call ottyp

pop cx

pop ax

pop dx

jc error

; 12/07/2014

xchg al, cl

cmp cl, 7

ja sysstty\_12

;

; 16/01/2014

xor bh, bh

;

sysstty\_9: ; tty 0 to tty 7

; al = character

cmp dh, 0FFh ; Do not set cursor position

je short sysstty\_10

push cx

push ax

mov bl, cl ; (tty number = video page number)

;xor bh, bh

call set\_cpos

pop ax

pop cx

sysstty\_10:

; 17/01/2014

inc ch

jz short sysstty\_11 ; ch = FFh

dec ch

jz short sysstty\_11 ; ch = 0

; ch > 0 and ch < FFh

; write a character at current cursor position

mov ah, 07h ; ah = 7 (color/attribute), al = char

; 12/07/2014

push cx

call write\_c\_current

pop cx

sysstty\_11:

; 14/01/2014

xor dl, dl ; sysstty call sign

; 18/01/2014

mov al, cl

cbw

call cttyp

jmp sysret

; Original UNIX v1 'sysstty' routine:

; gtty:

;sysstty: / set mode of typewriter; 3 consequtive word arguments

;jsr r0,gtty / r1 will have offset to tty block,

; / r2 has source

;mov r2,-(sp)

;mov r1,-(sp) / put r1 and r2 on the stack

;1: / flush the clist wait till typewriter is quiescent

;mov (sp),r1 / restore r1 to tty block offset

;movb tty+3(r1),0f / put cc offset into getc argument

;mov $240,\*$ps / set processor priority to 5

;jsr r0,getc; 0:../ put character from clist in r1

; br .+4 / list empty, skip branch

;br 1b / get another character until list is empty

;mov 0b,r1 / move cc offset to r1

;inc r1 / bump it for output clist

;tstb cc(r1) / is it 0

;beq 1f / yes, no characters to output

;mov r1,0f / no, put offset in sleep arg

;jsr r0,sleep; 0:.. / put tty output process to sleep

;br 1b / try to calm it down again

;1:

;mov (sp)+,r1

;mov (sp)+,r2 / restore registers

;mov (r2)+,r3 / put reader control status in r3

;beq 1f / if 0, 1f

;mov r3,rcsr(r1) / move r.c. status to reader

; / control status register

;1:

;mov (r2)+,r3 / move pointer control status to r3

;beq 1f / if 0 1f

;mov r3,tcsr(r1) / move p.c. status to printer

; / control status reg

;1:

;mov (r2)+,tty+4(r1) / move to flag byte of tty block

;jmp sysret2 / return to user

sysgtty: ; < get tty status >

; 12/07/2014

; 22/04/2014

; 26/01/2014

; 17/01/2014

; 16/01/2014

; 07/12/2013

; 04/12/2013

; 03/09/2013

; 15/08/2013

; 16/07/2013

; 02/06/2013

; 30/05/2013

; 'sysgtty' gets the status of tty in question.

; It stores in the three words addressed by it's argument

; the status of the typewriter whose file descriptor

; in (u.r0).

;

; Calling sequence:

; sysgtty; arg

; Arguments:

; arg - address of 3 words destination of the status

; Inputs: ((\*u.r0 - file descriptor))

; Outputs: ((status in address which is pointed to by arg))

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

;

; Retro UNIX 8086 v1 modification:

; 'sysgtty' system call will return status of tty

; (keyboard, serial port and video page status)

; in following manner:

;

; Inputs:

; BX = 0 --> means

; CH = 0 --> 'return status of the console tty'

; for (current) process

; CL = 0 --> return keyboard status (tty 0 to 7)

; CL = 1 --> return video page status (tty 0 to 7)

; CH > 0 --> tty number + 1

;

; BX > 0 --> points to name of tty

; CL = 0 --> return keyboard status

; CL = 1 --> return video page status

; CH = undefined

;

; Outputs:

; cf = 0 ->

;

; AL = tty number from 0 to 9

; (0 to 7 is also the video page of the tty)

; AH = 0 if the tty is free/unused

; AH = the process number of the caller

; AH = FFh if the tty is locked by another process

;

; (if calling is for serial port status)

; BX = serial port status if tty number is 8 or 9

; (BH = modem status, BL = Line status)

; CX = 0FFFFh (if data is ready)

; CX = 0 (if data is not ready or undefined)

;

; (if calling is for keyboard status)

; BX = current character in tty/keyboard buffer

; (BH = scan code, BL = ascii code)

; (BX=0 if there is not a waiting character)

; CX is undefined

;

; (if calling is for video page status)

; BX = cursor position on the video page

; if tty number < 8

; (BH = row, BL = column)

; CX = current character (in cursor position)

; on the video page of the tty

; if tty number < 8

; (CH = color, CL = character)

;

; cf = 1 means error (requested tty is not ready)

;

; AH = FFh if the caller is not owner of

; specified tty or console tty

; AL = tty number (0FFh if it does not exist)

; BX, CX are undefined if cf = 1

;

; (If tty number is 8 or 9)

; AL = tty number

; AH = the process number of the caller

; BX = serial port status

; (BH = modem status, BL = Line status)

; CX = 0

;

sysgtty\_0:

gtty: ; get (requested) tty number

; 12/07/2014

; 22/04/2014

; 15/04/2014

; 26/01/2014

; 17/01/2014

; 16/01/2014

; 07/12/2013

; 04/12/2013

; 03/09/2013

; 19/08/2013

; 16/07/2013

; 02/06/2013

; 30/05/2013

; Retro UNIX 8086 v1 modification !

;

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

;

; 16/01/2014

mov word ptr [u.r0], 0FFFFh

cmp cl, 1

ja error

;

and bx, bx

jz short sysgtty\_1

;

mov word ptr [u.namep], bx

call namei

jc error

;

xor bh, bh

cmp ax, 1

jna short sysgtty\_2

sub ax, 10

cmp ax, 9

ja error

mov ch, al

jmp short sysgtty\_4

sysgtty\_1:

; 16/01/2014

cmp ch, 10

ja error

dec ch ; 0 -> FFh (negative)

jns short sysgtty\_3 ; not negative

;

sysgtty\_2:

; get tty number of console tty

mov ah, byte ptr [u.uno]

mov bl, ah

;xor bh, bh

mov ch, byte ptr [BX]+p.ttyc-1

sysgtty\_3:

mov al, ch

sysgtty\_4:

mov byte ptr [u.r0], al

;cmp ch, 9

;ja error

cmp ch, 8 ; cmp al, 8

jb short sysgtty\_6

;

; 12/07/2014

mov dx, 0

je short sysgtty\_5

inc dl

sysgtty\_5:

; 12/07/2014

mov ah, 3 ; get serial port status

int 14h

xchg ah, al

mov word ptr [BP]+6, ax ; serial port status

mov ah, byte ptr [u.uno]

mov byte ptr [u.r0]+1, ah

mov word ptr [BP]+8, 0 ; data status (0 = not ready)

test al, 80h

jnz error

test al, 1

jz sysret

dec word ptr [BP]+8 ; data status (FFFFh = ready)

jmp sysret

sysgtty\_6:

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

mov byte ptr [u.ttyn], al ; tty number

;xor bh, bh

mov bl, al ; tty number (0 to 7)

shl bl, 1 ; aligned to word

; 22/04/2014

add bx, offset ttyl

mov ah, byte ptr [BX]

cmp ah, byte ptr [u.uno]

je short sysgtty\_7

and ah, ah

;jz short sysgtty\_7

jnz short sysgtty\_8

;mov ah, 0FFh

sysgtty\_7:

mov byte ptr [u.r0]+1, ah

sysgtty\_8:

or cl, cl

jnz short sysgtty\_9

mov al, 1 ; test a key is available

call getc

mov word ptr [BP]+6, ax ; bx, character

jmp sysret

sysgtty\_9:

mov bl, byte ptr [u.ttyn]

; bl = video page number

call get\_cpos

; dx = cursor position

mov word ptr [BP]+6, dx ; bx

;mov bl, byte ptr [u.ttyn]

; bl = video page number

call read\_ac\_current

; ax = character and attribute/color

mov word ptr [BP]+8, ax ; cx

jmp sysret

; Original UNIX v1 'sysgtty' routine:

; sysgtty:

;jsr r0,gtty / r1 will have offset to tty block,

; / r2 has destination

;mov rcsr(r1),(r2)+ / put reader control status

; / in 1st word of dest

;mov tcsr(r1),(r2)+ / put printer control status

; / in 2nd word of dest

;mov tty+4(r1),(r2)+ / put mode in 3rd word

;jmp sysret2 / return to user

; Original UNIX v1 'gtty' routine:

; gtty:

;jsr r0,arg; u.off / put first arg in u.off

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

;jsr r0,getf / get the i-number of the file

;tst r1 / is it open for reading

;bgt 1f / yes

;neg r1 / no, i-number is negative,

; / so make it positive

;1:

;sub $14.,r1 / get i-number of tty0

;cmp r1,$ntty-1 / is there such a typewriter

;bhis error9 / no, error

;asl r1 / 0%2

;asl r1 / 0%4 / yes

;asl r1 / 0%8 / multiply by 8 so r1 points to

; ; / tty block

;mov u.off,r2 / put argument in r2

;rts r0 / return