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

;

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

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

; U3.ASM (include u0.asm) //// UNIX v1 -> u3.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: 08/03/2014 ] ;;; 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>

;

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

; 08/03/2014 wswap, rswap, swap

; 25/02/2014 swap

; 23/02/2014 putlu, swap

; 14/02/2014 swap ('SRUN' check), putlu (single level runq)

; 05/02/2014 swap (SSLEEP/SWAIT/SRUN, p.waitc)

; 23/10/2013 swap (consistency check), idle

; 10/10/2013 idle

; 24/09/2013 swap, wswap, rswap, tswap (consistency check)

; 20/09/2013 swap

; 30/08/2013 swap

; 09/08/2013 swap

; 08/08/2013 putlu, wswap, rswap

; 03/08/2013

; 01/08/2013

; 29/07/2013

; 24/07/2013

; 23/07/2013

; 09/07/2013

; 26/05/2013

; 24/05/2013

; 21/05/2013

; 17/05/2013

; 16/05/2013 swap

; 19/04/2013 swap, wrswap

; 14/04/2013 tswap, swap

; 10/04/2013

; 11/03/2013

tswap:

; 14/02/2014 single level runq

; 24/09/2013 consistency check -> ok

; 26/05/2013 (swap, putlu modifications)

; 14/04/2013

; time out swap, called when a user times out.

; the user is put on the low priority queue.

; This is done by making a link from the last user

; on the low priority queue to him via a call to 'putlu'.

; then he is swapped out.

;

; RETRO UNIX 8086 v1 modification ->

; 'swap to disk' is replaced with 'change running segment'

; according to 8086 cpu (x86 real mode) architecture.

; pdp-11 was using 64KB uniform memory while IBM PC

; compatibles was using 1MB segmented memory

; in 8086/8088 times.

;

; INPUTS ->

; u.uno - users process number

; runq+4 - lowest priority queue

; OUTPUTS ->

; r0 - users process number

; r2 - lowest priority queue address

;

; ((AX = R0, BX = R2)) output

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

;

mov al, byte ptr [u.uno]

; movb u.uno,r1 / move users process number to r1

;mov bx, offset runq + 4

; mov $runq+4,r2

; / move lowest priority queue address to r2

call putlu

; jsr r0,putlu / create link from last user on Q to

; / u.uno's user

swap:

; 08/03/2014

; 25/02/2014

; 23/02/2014

; 14/02/2014 single level runq

; 05/02/2014 SSLEEP/SWAIT/SRUN, p.waitc

; 23/10/2013 consistency check -> ok

; 24/09/2013 consistency check -> ok

; 20/09/2013 ('call idle' enabled again)

; 30/08/2013

; 09/08/2013

; 29/07/2013

; 24/07/2013 sstack (= file size + 256)

; 26/05/2013 wswap and rswap (are come back!)

; 24/05/2013 (u.usp -> sp modification)

; 21/05/2013

; 16/05/2013

; 19/04/2013 wrswap (instead of wswap and rswap)

; 14/04/2013

; 'swap' is routine that controls the swapping of processes

; in and out of core.

;

; RETRO UNIX 8086 v1 modification ->

; 'swap to disk' is replaced with 'change running segment'

; according to 8086 cpu (x86 real mode) architecture.

; pdp-11 was using 64KB uniform memory while IBM PC

; compatibles was using 1MB segmented memory

; in 8086/8088 times.

;

; INPUTS ->

; runq table - contains processes to run.

; p.link - contains next process in line to be run.

; u.uno - process number of process in core

; s.stack - swap stack used as an internal stack for swapping.

; OUTPUTS ->

; (original unix v1 -> present process to its disk block)

; (original unix v1 -> new process into core ->

; Retro Unix 8086 v1 -> segment registers changed

; for new process)

; u.quant = 3 (Time quantum for a process)

; ((INT 1Ch count down speed -> 18.2 times per second)

; RETRO UNIX 8086 v1 will use INT 1Ch (18.2 times per second)

; for now, it will swap the process if there is not

; a keyboard event (keystroke) (Int 15h, function 4Fh)

; or will count down from 3 to 0 even if there is a

; keyboard event locking due to repetitive key strokes.

; u.quant will be reset to 3 for RETRO UNIX 8086 v1.

;

; u.pri -points to highest priority run Q.

; r2 - points to the run queue.

; r1 - contains new process number

; r0 - points to place in routine or process that called

; swap all user parameters

;

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

;

swap\_0:

;mov $300,\*$ps / processor priority = 6

; 14/02/2014

mov si, offset runq ; 23/02/2014 BX -> DI -> SI

; mov $runq,r2 / r2 points to runq table

swap\_1: ; 1: / search runq table for highest priority process

mov ax, word ptr [SI]

and ax, ax

; tst (r2)+ / are there any processes to run

; / in this Q entry

jnz short swap\_2

; bne 1f / yes, process 1f

; cmp r2,$runq+6 / if zero compare address

; / to end of table

; bne 1b / if not at end, go back

;mov cl, byte ptr [u.uno]

;mov al, 'X'

;mov ah, 04Fh

;add cl, '0'

;mov ch, ah

;call write\_sign

;; 25/02/2014

;;mov al, byte ptr [ptty]

;;call wakeup

;;or al, al

;;jnz short swap\_1

;

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

;;; 30/08/2013

; 20/09/2013

call idle ; 23/10/2013 (consistency check !)

; jsr r0,idle; s.idlet+2 / wait for interrupt;

; / all queues are empty

; 14/02/2014

jmp short swap\_1

; br swap

swap\_2: ; 1:

; tst -(r2) / restore pointer to right Q entry

; mov r2,u.pri / set present user to this run queue

;mov ax, word ptr [SI]

; movb (r2)+,r1 / move 1st process in queue to r1

;

cmp al, ah ; 16/05/2013

; cmpb r1,(r2)+ / is there only 1 process

; / in this Q to be run

je short swap\_3

; beq 1f / yes

; tst -(r2) / no, pt r2 back to this Q entry

;

mov bl, al

xor bh, bh

mov ah, byte ptr [BX]+p.link-1

mov byte ptr [SI], ah

; movb p.link-1(r1),(r2) / move next process

; / in line into run queue

jmp short swap\_4

; br 2f

swap\_3: ; 1:

xor dx, dx

; 23/02/2014 BX -> SI

mov word ptr [SI], dx ;16/05/2013

; clr -(r2) / zero the entry; no processes on the Q

;

; 26/05/2013 (swap\_4 and swap\_5)

swap\_4: ; / write out core to appropriate disk area and read

; / in new process if required

; clr \*$ps / clear processor status

; 09/08/2013

mov ah, byte ptr [u.uno]

cmp ah, al

;cmp byte ptr [u.uno], al

; cmpb r1,u.uno / is this process the same as

; / the process in core?

je short swap\_6

; beq 2f / yes, don't have to swap

; mov r0,-(sp) / no, write out core; save r0

; / (address in routine that called swap)

mov word ptr [u.usp], sp

; mov sp,u.usp / save stack pointer

; 09/08/2013

; 24/07/2013

;mov sp, sstack ; offset sstack

; mov $sstack,sp / move swap stack pointer

; / to the stack pointer

;push ax

; mov r1,-(sp) / put r1 (new process #) on the stack

; 09/08/2013

or ah, ah

;cmp byte ptr [u.uno], dl ; 0

; tstb u.uno / is the process # = 0

jz short swap\_5

;jna short swap\_5

; beq 1f / yes, kill process by overwriting

call wswap

;jsr r0,wswap / write out core to disk

swap\_5: ;1:

; pop ax

; mov (sp)+,r1 / restore r1 to new process number

; 08/03/2014

; (protect 'rswap' return address from stack overwriting)

cli

mov sp, sstack - 190 ; (SizeOfFile + 2)

;

call rswap

; jsr r0,rswap / read new process into core

; jsr r0,unpack / unpack the users stack from next

; / to his program to its normal

mov sp, word ptr [u.usp]

; mov u.usp,sp / location; restore stack pointer to

; / new process stack

; mov (sp)+,r0 / put address of where the process

; / that just got swapped in, left off.,

; / i.e., transfer control to new process

sti

swap\_6: ;2:

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

; 30/08/2013

; RETRO UNIX 8086 v1 modification !

mov byte ptr [u.quant], time\_count

;mov byte ptr [uquant], 3

; movb $30.,uquant / initialize process time quantum

retn

; rts r0 / return

wswap: ; < swap out, swap to disk >

; 08/03/2014 major modification

; 24/09/2013 consistency check -> ok

; 08/08/2013

; 24/07/2013

; 26/05/2013

; 'wswap' writes out the process that is in core onto its

; appropriate disk area.

;

; Retro UNIX 8086 v1 modification ->

; 'swap to disk' is replaced with 'change running segment'

; according to 8086 cpu (x86 real mode) architecture.

; pdp-11 was using 64KB uniform memory while IBM PC

; compatibles was using 1MB segmented memory

; in 8086/8088 times.

;

; INPUTS ->

; u.break - points to end of program

; u.usp - stack pointer at the moment of swap

; core - beginning of process program

; ecore - end of core

; user - start of user parameter area

; u.uno - user process number

; p.dska - holds block number of process

; OUTPUTS ->

; swp I/O queue

; p.break - negative word count of process

; r1 - process disk address

; r2 - negative word count

;

; RETRO UNIX 8086 v1 input/output:

;

; INPUTS ->

; u.uno - process number (to be swapped out)

; OUTPUTS ->

; none

;

; ((Modified registers: CX, SI, DI))

mov di, sdsegmnt

mov es, di

xor cl, cl

mov ch, byte ptr [u.uno]

dec ch ; 0 based process number

;; 08/03/2014 (swap data space is 256 bytes for every process)

;;shr cx, 1 ; swap data space is 128 bytes for every process

mov di, cx

mov cx, 32

mov si, offset u ; user structure

rep movsw

;

mov si, word ptr [u.usp] ; sp (system stack pointer)

mov cx, sstack

sub cx, si ; NOTE: system stack size = 256-64 = 192 bytes

rep movsb

;

mov cx, ds

mov es, cx

retn

;

; 08/08/2013, 14 -> 16, 7 -> 8

;mov si, sstack - 16 ; 24/07/2013

; offset sstack - 16 ;; = word ptr [u.sp\_] - 2

;mov cx, 8

;rep movsw

;mov cl, 32

;mov si, offset u ; user structure

;rep movsw

;mov cx, ds

;mov es, cx

;retn

; Original UNIX v1 'wswap' routine:

; wswap:

; mov \*$30,u.emt / determines handling of emts

; mov \*$10,u.ilgins / determines handling of

; / illegal instructions

; mov u.break,r2 / put process program break address in r2

; inc r2 / add 1 to it

; bic $1,r2 / make it even

; mov r2,u.break / set break to an even location

; mov u.usp,r3 / put users stack pointer

; / at moment of swap in r3

; cmp r2,$core / is u.break less than $core

; blos 2f / yes

; cmp r2,r3 / no, is (u.break) greater than stack ptr.

; bhis 2f / yes

; 1:

; mov (r3)+,(r2)+ / no, pack stack next to users program

; cmp r3,$ecore / has stack reached end of core

; bne 1b / no, keep packing

; br 1f / yes

; 2:

; mov $ecore,r2 / put end of core in r2

; 1:

; sub $user,r2 / get number of bytes to write out

; / (user up to end of stack gets written out)

; neg r2 / make it negative

; asr r2 / change bytes to words (divide by 2)

; mov r2,swp+4 / word count

; movb u.uno,r1 / move user process number to r1

; asl r1 / x2 for index

; mov r2,p.break-2(r1) / put negative of word count

; / into the p.break table

; mov p.dska-2(r1),r1 / move disk address of swap area

; / for process to r1

; mov r1,swp+2 / put processes dska address in swp+2

; / (block number)

; bis $1000,swp / set it up to write (set bit 9)

; jsr r0,ppoke / write process out on swap area of disk

; 1:

; tstb swp+1 / is lt done writing?

; bne 1b / no, wait

; rts r0 / yes, return to swap

rswap: ; < swap in, swap from disk >

; 08/03/2014 major modification

; 24/09/2013 consistency check -> ok

; 08/08/2013

; 24/07/2013

; 26/05/2013

; 'rswap' reads a process whose number is in r1,

; from disk into core.

;

; RETRO UNIX 8086 v1 modification ->

; 'swap to disk' is replaced with 'change running segment'

; according to 8086 cpu (x86 real mode) architecture.

; pdp-11 was using 64KB uniform memory while IBM PC

; compatibles was using 1MB segmented memory

; in 8086/8088 times.

;

; INPUTS ->

; r1 - process number of process to be read in

; p.break - negative of word count of process

; p.dska - disk address of the process

; u.emt - determines handling of emt's

; u.ilgins - determines handling of illegal instructions

; OUTPUTS ->

; 8 = (u.ilgins)

; 24 = (u.emt)

; swp - bit 10 is set to indicate read

; (bit 15=0 when reading is done)

; swp+2 - disk block address

; swp+4 - negative word count

; ((swp+6 - address of user structure))

;

; RETRO UNIX 8086 v1 input/output:

;

; INPUTS ->

; AL - new process number (to be swapped in)

; OUTPUTS ->

; none

;

; ((Modified registers: AX, CX, SI, DI))

mov ah, al

dec ah

xor al, al

;;shr ax, 1 ; 08/03/2014 (256 bytes per process)

mov si, ax ; SI points copy of sstack in sdsegment

; u.sp\_ points sstack-12 (for 6 registers)

mov ax, sdsegmnt ; 17/05/2013

mov ds, ax ; sdsegment

; 08/03/2014

mov di, offset u

mov cx, 32

rep movsw

mov di, word ptr ES:[u.usp] ; system stack pointer location

mov cx, sstack

sub cx, di ; Max. 256-64 bytes stack space

rep movsb

mov ax, cs

mov ds, ax

retn

;

; 08/08/2013 14 -> 16, 7 ->8

; 24/07/2013

;mov di, sstack - 16 ; offset sstack-14

;mov cx, 8

;rep movsw

;mov di, offset u

;mov cl, 32

;rep movsw

;mov ax, cs

;mov ds, ax

;retn

; Original UNIX v1 'rswap' and 'unpack' routines:

;rswap:

; asl r1 / process number x2 for index

; mov p.break-2(r1), swp+4 / word count

; mov p.dska-2(r1),swp+2 / disk address

; bis $2000,swp / read

; jsr r0,ppoke / read it in

; 1:

; tstb swp+1 / done

; bne 1b / no, wait for bit 15 to clear (inhibit bit)

; mov u.emt,\*$30 / yes move these

; mov u.ilgins,\*$10 / back

; rts r0 / return

;unpack: ; / move stack back to its normal place

; mov u.break,r2 / r2 points to end of user program

; cmp r2,$core / at beginning of user program yet?

; blos 2f / yes, return

; cmp r2,u.usp / is break\_above the stack pointer

; / before swapping

; bhis 2f / yes, return

; mov $ecore,r3 / r3 points to end of core

; add r3,r2

; sub u.usp,r2 / end of users stack is in r2

; 1:

; mov -(r2),-(r3) / move stack back to its normal place

; cmp r2,u.break / in core

; bne 1b

; 2:

; rts r0

putlu:

; 23/02/2014

; 14/02/2014 single level run queue

; 08/08/2013

; 26/05/2013 (si -> di)

; 15/04/2013

;

; 'putlu' is called with a process number in r1 and a pointer

; to lowest priority Q (runq+4) in r2. A link is created from

; the last process on the queue to process in r1 by putting

; the process number in r1 into the last process's link.

;

; INPUTS ->

; r1 - user process number

; r2 - points to lowest priority queue

; p.dska - disk address of the process

; u.emt - determines handling of emt's

; u.ilgins - determines handling of illegal instructions

; OUTPUTS ->

; r3 - process number of last process on the queue upon

; entering putlu

; p.link-1 + r3 - process number in r1

; r2 - points to lowest priority queue

;

; ((Modified registers: DX, BX, DI))

;

; / r1 = user process no.; r2 points to lowest priority queue

; BX = r2

; AX = r1 (AL=r1b)

; 14/02/2014

mov bx, offset runq

; 23/02/2014

mov dx, word ptr [BX]

inc bx

and dx, dx

; tstb (r2)+ / is queue empty?

jz short putlu\_1

; beq 1f / yes, branch

mov dl, dh

xor dh, dh

mov di, dx

; movb (r2),r3 / no, save the "last user" process number

; / in r3

mov byte ptr [DI]+p.link-1, al

; movb r1,p.link-1(r3) / put pointer to user on

; / "last users" link

jmp short putlu\_2

; br 2f /

putlu\_1: ; 1:

mov byte ptr [BX]-1, al ; 08/08/2013

; movb r1,-1(r2) / user is only user;

; / put process no. at beginning and at end

putlu\_2: ; 2:

mov byte ptr [BX], al

; movb r1,(r2) / user process in r1 is now the last entry

; / on the queue

; 23/02/2014

mov dl, al

mov di, dx

mov byte ptr [DI]+p.link-1, dh ; 0

;

;14/02/2014

;dec bx

; dec r2 / restore r2

retn

; rts r0

;copyz:

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

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

; mov (r0)+,r1

; mov (r0)+,r2

;1:

; clr (r1)+ / clear all locations between r1 and r2

; cmp r1,r2

; blo 1b

; mov (sp)+,r2 / restore r2

; mov (sp)+,r1 / restore r1

; rts r0

idle:

; 23/10/2013

; 10/10/2013

; 29/07/2013

; 09/07/2013

; 10/04/2013

; (idle & wait loop)

; Retro Unix 8086 v1 modification on original Unixv1 idle procedure!

; input -> CX = wait count

;sti

; 29/07/2013

hlt

nop ; 10/10/2013

nop

nop

; 23/10/2013

nop

nop

nop

nop

retn

;sti

;;;push word ptr [clockp]

;or cx, cx

;jnz short @f

;inc cx

;@@:

;;;mov word ptr [clockp], cx

@@:

;hlt ; wait for interrupt (timer interrupt or keyboard interrupt etc.)

;;;dec word ptr [clockp]

;dec cx ; 09/07/2013 ;;;

;jnz short @b

;;; pop word ptr [clockp]

;retn

;mov \*$ps,-(sp) / save ps on stack

;clr \*$ps / clear ps

;mov clockp,-(sp) / save clockp on stack

;mov (r0)+,clockp / arg to idle in clockp

;1 / wait for interrupt

;mov (sp)+,clockp / restore clockp, ps

;mov (sp)+,\*$ps

;rts r0

clear:

; 03/08/2013

; 01/08/2013

; 23/07/2013

; 09/04/2013

;

; 'clear' zero's out of a block (whose block number is in r1)

; on the current device (cdev)

;

; INPUTS ->

; r1 - block number of block to be zeroed

; cdev - current device number

; OUTPUTS ->

; a zeroed I/O buffer onto the current device

; r1 - points to last entry in the I/O buffer

;

; ((AX = R1)) input/output

; (Retro UNIX Prototype : 18/11/2012 - 14/11/2012, UNIXCOPY.ASM)

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

call wslot

; jsr r0,wslot / get an I/O buffer set bits 9 and 15 in first

; / word of I/O queue r5 points to first data word in buffer

mov di, bx ; r5

mov dx, ax ; 01/08/2013

mov cx, 256

; mov $256.,r3

xor ax, ax

rep stosw ; 03/08/2013

mov ax, dx ; 01/08/2013

; 1:

; clr (r5)+ / zero data word in buffer

; dec r3

; bgt 1b / branch until all data words in buffer are zero

call dskwr

; jsr r0,dskwr / write zeroed buffer area out onto physical

; / block specified in r1

; AX (r1) = block number

retn

; rts r0