Relatório MC404 - Trabalho 3 - Prof. Rodolfo

Gustavo Sverzut Barbieri, Ivens Prates Telles Alves

Grupo:
Gustavo Sverzut Barbieri RA: 008849
Ivens Prates Telles Alves RA: 008908

1 O Projeto

O terceiro trabalho de MC404 consta em implementar um escalonador de processos. Um escalonador deve simular um sistema multi tarefas, rodando mais de um processo ao mesmo tempo. Nosso escalonador rodará 5 processos ao mesmo tempo podendo parar e continuar o andamento dos mesmos. Os processos a serem rodados serão:

- Imprimir um texto no topo da tela (12 primeiras linhas)
- Imprimir um texto no centro-esquerdo da tela (linhas 13 a 24, 40 primeiras colunas)
- Copiar o texto da área centro-esquerda para a centro-direita, espelhando-o. (Copia o impresso pelo Processo 2, espelhado, para o centro-direito)
- Tocar uma música.
- Mostrar status dos processos anteriores e parar/continuar os mesmos.

2 Funções/Procedimentos:

Nosso projeto foi constituído de 5 processos e um escalonador. Processos os quais foram implementados somente para ajudar a visualizar o funcionamento do escalonador. Eles foram implementados conforme os trabalhos anteriores, portanto não serão discutidos aqui.

O Escalonador foi colocado como tratamento para a interrupção do relógio (8h). Ao ser chamado ele guarda os registradores do processo ativo na pilha, muda para o próximo processo ativo e restaura os registradores deste, que estavam na pilha.

Os processos são marcados ativos ou inativos pelo processo 5, que muda o estado do processo num vetor (Processes).

A inicialização dos processos é feita pela rotina *InitProcess*, que recebe como parâmetro em *al* o número do processo e em *bx* o apontador para o processo. Esta função pega um espaço da pilha reservado para o processo e coloca nela os flags, *cs*, *ip*, registradores. Assim é como se o programa estivesse "dormente" nesta área da pilha. O escalonador usa estas informações para restaurar o processo mais tarde.

3 Decisões e Dificuldades encontradas:

O escalonador foi inicialmente feito usando-se "bitmasks" e ficou grande e mal-feito, apesar de funcionar. Após este funcionar, refiz o escalonador usando vetores, assim pode-se rodar qualquer número de processos fazendo pequenos ajustes no tamanho que se reserva para a pilha. O código ficou bem mais legível com esta abordagem.

```
; s$$$$$$$s   s$$$$$$$$s  s$$$$$$$s
;$$$$$’  ‘’   $$$$$$$ ‘’  $$$$  $$$$
;$$$$  $$$$$     ‘$$$$$s  $$$$$$$$$
;‘$$$$s.s$$$ s$$$s..s$$$  $$$$  $$$$
; ‘$$$$$$$$’  ‘$$$$$$$$’  ‘$$$$$$$’



[BITS 16]               ;Set code generation to 16 bit mode
[ORG 0x0100]            ;Set code start address to 0100h


[SEGMENT .text]         ;Main code segment
;;; Reserve processes stack
    ;; init video mode
    mov    ax, 0003h
    int    10h

    ;; save the stack
    mov    [cs:Stack_Main], sp

    ;; init process 1
    mov    al, 0
    mov    bx, PROCESS_1
    call   InitProcess

    ;; init process 2
    mov    al, 1
    mov    bx, PROCESS_2
    call   InitProcess

    ;; init process 3
    mov    al, 2
    mov    bx, PROCESS_3
    call   InitProcess

    ;; init process 4
```

```
    mov    al, 3
    mov    bx, PROCESS_4
    call   InitProcess

    ;; init process 5
    mov    al, 4
    mov    bx, PROCESS_5
    call   InitProcess

    ;; Assign to ints
     call  AssignIntFunction

    ;; start with process 1 running
    mov    ax, [cs:Stacks]
    mov    sp, ax
    mov    [cs:Process], word 0
    jmp    PROCESS_1    ; say goodbye! let's go!
```

```
;;; SCHEDULER: (Interruption Handler) swap between active processes
; Parameters
;      None
; Return
;      None
SCHEDULER:
    int    60h
    pusha
    pushf
    push   es
    push   ds

    mov    bx, [cs:Process]
    ;; save stack:
```

```
    add    bx, bx        ; cx *= 2
    mov    [cs:Stacks+bx], sp
    ;; now, run the next process
    mov    bx, [cs:Process]
SCHEDULER_NextProcess:
    inc    bx                          ; next process
    cmp    bx, [cs:N_Processes]        ; we reached the last
                                       ; process?
    jne    SCHEDULER_NextProcess_Cont1
    xor    bx, bx                      ; back to the 1st p.
SCHEDULER_NextProcess_Cont1:
    mov    cx, bx                      ; save bx
    add    bx, bx                      ; bx *= 2
    cmp    [cs:Processes+bx], word 1   ; is this p. active?
    je     SCHEDULER_NextProcess_Cont   ; no, next...
    mov    bx, cx                      ; restore bx
    jmp    SCHEDULER_NextProcess
SCHEDULER_NextProcess_Cont:
    mov    [cs:Process], cx            ; set new process
    mov    sp, [cs:Stacks+bx]          ; retore p. stack

    pop    ds
    pop    es
    popf
    popa
    iret
;;; End: SCHEDULER
```

```
;;; PROCESS_1:    print some text on the top of the screen
; Parameters
;      None
; Return
;      None
```

```
PROCESS_1:
    mov    si, STR_PROC_1
    mov    ax, cs
    mov    ds, ax

    les    di, [cs:VideoSegment]
    mov    ah, 1Fh
    mov    al, ' '
    ;; clear the screen
    push   bx
    xor    bx, bx
PROCESS_1_Loop1:
    inc    bx
    stosw
    cmp    bx, 12*80    ; 12 lines with 2-bytes 80 columns
    jne    PROCESS_1_Loop1
    ;; screen cleaned.

    ;; Enter the main loop (infinite loop)
    pop    bx
    xor    di, di
    xor    cx, cx
PROCESS_1_Loop_Main:
PROCESS_1_Loop2:
    ;; load char from string and print on the screen
    lodsb
    cmp    al, 0
    je     PROCESS_1_Loop2_NOPRINT
    stosw
    inc    cx
    ;; Wait Loop
PROCESS_1_Loop2_NOPRINT:
    push   cx
    push   dx
    xor    cx, cx
PROCESS_1_Loop3:
    inc    cx
    xor    dx, dx
```

```
PROCESS_1_Loop4:
    inc    dx
    cmp    dx, 65535
    jne    PROCESS_1_Loop4
    cmp    cx, 100
    jne    PROCESS_1_Loop3
    ;; end of the wait loop
    pop    dx
    pop    cx

    cmp    cx, 12*80
    jl     PROCESS_1_Loop_Main_Cont
    xor    di, di
    xor    cx, cx
PROCESS_1_Loop_Main_Cont:
    ;; we reach the end of the string?
    cmp    al, 0
    jne    PROCESS_1_Loop2
    ;; reset to the begining of the string, change the color
    mov    si, STR_PROC_1
    cmp    ah, 1Fh
    je     PROCESS_1_Color_2
PROCESS_1_Color_1:
    mov    ah, 1Fh
    jmp    PROCESS_1_Color_Cont
PROCESS_1_Color_2:
    mov    ah, 17h
    jmp    PROCESS_1_Color_Cont
PROCESS_1_Color_Cont:
    jmp    PROCESS_1_Loop_Main;  the loop that never finnishes!
;;; End: PROCESS_1
```

```
;;; PROCESS_2:    print some text on the center-left of the screen
; Parameters
;      None
; Return
;      None
PROCESS_2:
    mov    si, STR_PROC_2
    mov    ax, cs
    mov    ds, ax

    les    di, [cs:VideoSegment]
    mov    di, 12*80*2    ; go to start position
    mov    ah, 2Fh
    mov    al, ' '
    ;; clear the screen
    push   bx
    push   cx
    xor    bx, bx
PROCESS_2_Loop1_1:
    inc    bx
    xor    cx, cx
PROCESS_2_Loop1_2:
    inc    cx
    stosw
    cmp    cx, 40            ; 40 columns
    jl     PROCESS_2_Loop1_2
    add    di, 40*2          ; add 40 rows (2 bytes per row)
    cmp    bx, 12            ; 12 rows
    jl     PROCESS_2_Loop1_1
    ;; screen cleaned.

    ;; Enter the main loop (infinite loop)
    pop    cx
    pop    bx
    les    di, [cs:VideoSegment]
    mov    di, 12*80*2       ; go to start position
    xor    cx, cx
    xor    bx, bx
```

```
PROCESS_2_Loop_Main:
PROCESS_2_Loop2:
    ;; load char from string and print on the screen
    lodsb
    cmp    al, 0
    je    PROCESS_2_Loop2_NOPRINT
    stosw
    inc    cx
    inc    bx
    ;; Wait Loop:
PROCESS_2_Loop2_NOPRINT:
    push   cx
    push   dx
    xor    cx, cx
PROCESS_2_Loop3:
    inc    cx
    xor    dx, dx
PROCESS_2_Loop4:
    inc    dx
    cmp    dx, 65535
    jne    PROCESS_2_Loop4
    cmp    cx, 20
    jne    PROCESS_2_Loop3
    ;; end of the wait loop
    pop    dx
    pop    cx

    cmp    bx, 40
    jl     PROCESS_2_Loop_Main_Cont1
    xor    bx, bx
    add    di, 40*2

PROCESS_2_Loop_Main_Cont1:
    cmp    cx, 12*40
    jl     PROCESS_2_Loop_Main_Cont2
    mov    di, 12*80*2    ; go to start position
    xor    cx, cx
PROCESS_2_Loop_Main_Cont2:
```

```
    ;; we reach the end of the string?
    cmp    al, 0
    jne    PROCESS_2_Loop2
    ;; reset to the begining of the string, change the color
    mov    si, STR_PROC_2
    ;; Change colors
    cmp    ah, 2Fh
    je     PROCESS_2_Color_2
PROCESS_2_Color_1:
    mov    ah, 2Fh
    jmp    PROCESS_2_Color_Cont
PROCESS_2_Color_2:
    mov    ah, 57h
    jmp    PROCESS_2_Color_Cont
PROCESS_2_Color_Cont:
    jmp    PROCESS_2_Loop_Main;  the loop that never finnishes!
;;; End: PROCESS_2




;;; PROCESS_3: copy the text at the center-left of the screen to
;;;            the center-right
; Parameters
;      None
; Return
;      None
PROCESS_3:
    lds    si, [cs:VideoSegment] ; read from ds:si
    les    di, [cs:VideoSegment] ; write to es:di

    mov    si, 80*2*12      ; start reading from 12,0
    mov    di, 80*2*12+79*2 ; start writing to 12,79 (and dec)

    xor    dx, dx
PROCESS_3_Loop1:
    inc    dx
```

```
    xor    cx, cx
PROCESS_3_Loop2:
    inc    cx
    ;; read and write from video memory
    lodsw
    mov    [es:di], ax
    sub    di, 2

    cmp    cx, 40
    jl     PROCESS_3_Loop2
    ;; end of column loop
    add    si, 40*2
    add    di, 80*2+40*2

    cmp    dx, 12
    jl     PROCESS_3_Loop1
    ;; end of line loop
    jmp    PROCESS_3
;;; End: PROCESS_3
```

```
;;; PROCESS_4: Play sounds
; Parameters
;      None
; Return
;      None
PROCESS_4:
    ;; Activates the PC Speaker
    in     al, 61h        ; get data from 61h
    or     al, 11b        ; set bits 0 and 1
    out    61h, al        ; send data to 61h
    ;; Activates the PC Speaker Controller
```

```
    mov    ax, 0b6h
    out    43h, ax
PROCESS_4_SetMusic:
    mov    si, 0          ; our counter
PROCESS_4_Play:
    mov    ax, [cs:PROC_4_MUSIC+si]
    add    si, 2

    cmp    ax, 0FFFFh
    je    PROCESS_4_SetMusic

    out    42h, al        ; -play the sound
    xchg   ah, al         ; -
    out    42h, al        ; -
    xchg   ah, al

    ;; wait
    xor    cx, cx
PROCESS_4_Loop1:
    inc    cx
    xor    dx, dx
PROCESS_4_Loop2:
    inc    dx
    cmp    dx, 65535
    jne    PROCESS_4_Loop2
    cmp    cx, 500
    jne    PROCESS_4_Loop1
    ;; end of wait loop

    jmp    PROCESS_4_Play
;;; End: PROCESS_4
```

```
;;; PROCESS_5: control other process
; Parameters
;      None
; Return
;      None
PROCESS_5:
    ;; show initial processes status
    xor    cx, cx
PROCESS_5_Loop1:
    mov    bx, cx
    add    bx, bx                ; bx *= 2
    mov    al, cl                ; parameter to PrintProcessStatus
    mov    dx, [cs:Processes+bx] ; get Process status
    mov    ah, dl                ; parameter to PrintProcessStatus
    call   PrintProcessStatus
    inc    cx
    cmp    cx, [cs:N_Processes]
    jl     PROCESS_5_Loop1

PROCESS_5_WaitKey:
    mov    ah, 1
    int    16h
    jz     PROCESS_5_WaitKey
    ;; read key
    mov    ah, 0
    int    16h
    ;; check keys
    cmp    al, 27                ; ESC
    jne    PROCESS_5_NoQuit
    call    Quit
PROCESS_5_NoQuit:
    cmp    al, '1'
    jl     PROCESS_5             ; it's less than '1'
    cmp    al, '4'
    jg     PROCESS_5             ; it's greater than '4'
    sub    al, '1'

    xor    ah, ah
```

```
    mov    bx, ax
    add    bx, bx
    mov    ax, [cs:Processes+bx]
    cmp    ax, 1
    je    PROCESS_5_Toggle_Off
PROCESS_5_Toggle_On:
    mov    ax, 1
    jmp    PROCESS_5_END_Toggle
PROCESS_5_Toggle_Off:
    mov    ax, 0
PROCESS_5_END_Toggle:
    mov    [cs:Processes+bx], ax ; set process status

    jmp    PROCESS_5
;;; End: PROCESS_5




;;; QUIT: exits the program, changing stack back to initial
;;;       one and returning...
; Parameters:
;    None
; Return:
;    None
Quit:
    ;; Restore ints
    call   RestoreIntFunction
    mov    sp, [Stack_Main]

    ;; clear the screen
    mov    ah, 07h                ; ah = bg | fg
    mov    al, ' '                ; al = char
    ; Let es be the Video Segment
    les    di, [cs:VideoSegment]  ; video memory address:
                                  ; es = b800, di = 0000
```

```
    xor     bx, bx
    ;; clear the screen
    ; Main loop ( while (bx < screen_size) )
Loop_FillScreen:
    mov     [es:bx], ax
    add     bx, 2
    cmp     bx, 4000
    jne     Loop_FillScreen

    ;; shut down the speaker
    in      al, 61h                 ; get data from 61h
    and     al, 11111100b           ; unset bits 0 and 1
    out     61h, al                 ; send data to 61h

    ret
;;; End: Quit




;;; PrintProcessStatus: print the process <al> status in
;;;                     the last line
; Parameters:
;   al:    process number
;   ah:    process status: 1=Running, 0=Paused
; Return:
;   none
PrintProcessStatus:
    push   es
    push   ds
    pusha

    les    di, [cs:VideoSegment]
    mov    cx, cs
    mov    ds, cx         ; read from cs:si

    push   ax
    ;; set position in screen
```

```
    mov    di, 80*2*24
    mov    dl, al
    mov    al, 20*2
    mul    dl
    xor    ah, ah
    add    di, ax
    pop    ax

    cmp    ah, 1
    je     PrintProcessStatus_Choose_Running
PrintProcessStatus_Choose_Paused:
    mov    si, STR_PROC_5_PAU
    mov    dh, 04h                        ; red
    jmp    PrintProcessStatus_END_Choose
PrintProcessStatus_Choose_Running:
    mov    si, STR_PROC_5_RUN
    mov    dh, 02h                        ; green
    jmp    PrintProcessStatus_END_Choose
PrintProcessStatus_END_Choose:

    mov    dl, al
    add    dl, '0'
    inc    dl
    mov    ax, dx
    mov    ah, 0Fh
    stosw
    mov    al, ':'
    stosw
    mov    ah, dh

PrintProcessStatus_Loop1:
    lodsb
    cmp    al, 0
    je     PrintProcessStatus_END_Loop1
    stosw
    jmp    PrintProcessStatus_Loop1
PrintProcessStatus_END_Loop1:
```

```
    popa
    pop    ds
    pop    es
    ret
;;; End: PrintProcessStatus



;;; InitProcess: Initalizate the process in the stack
; Parameters:
;      al: process number
;      bx: process label
; Return:
;      None
InitProcess:
    pusha
    xor    ah, ah
    mov    si, sp        ; save sp, we need to restore it later

    mov    dx, ax        ; save ax
    mov    sp, Stack_Area+127; sp will be the process sp now
    xchg   al, cl        ; -
    mov    al, 128       ; -
    mul    cl            ; ax = <process#> * 512
    add    sp, ax        ; go to the position reserved to process
    mov    ax, dx        ; restore original ax
    ;; make the stack
    pushf                ; save flags
    push   cs            ; save cs
    push   bx            ; PROCESS to stack
    pusha
    pushf
    push   es
    push   ds
    ;; save the stack
    add    ax, ax        ; ax *= 2
    xchg   ax, bx
```

```
    mov    [cs:Stacks+bx], sp
    mov    [cs:Processes+bx], word 1 ; Process is active
    ;; return to the old stack
    mov    sp, si
    popa
    ret
;;; End: InitProcess



;;; AssignIntFunction: Assigns SCHEDULER as Clock int
; Parameters:
;   - None
; Return
;   - None
AssignIntFunction:
    ; save parameters we will use
    pusha
    cli            ; no interrupts by now
    les    di, [IntSegment]    ; es = interruption segment
    ;; Clock
    ;; Save actual interruption value
    mov    dx, [es:8h*4]
    mov    cx, [es:8h*4+2]
    mov    [es:60h*4], dx
    mov    [es:60h*4+2], cx
    ;; Assign new values
    mov    ax, cs
    mov    bx, SCHEDULER
    mov    [es:8h*4], bx
    mov    [es:8h*4+2], ax

    sti            ; now interrupts are available again
    ; restore parameters we will use
    popa
    ret
```

```
;;; RestoreIntFunction: restore the previous assigned interruption
; Parameters:
;   - None
; Return
;   - None
RestoreIntFunction:
    ;; save values
    pusha
    cli             ; no ints
    les    di, [IntSegment]    ; es = interruption segment
    ;; Clock
    ;; retore interrupt
    xor    ax, ax
    mov    es, ax
    mov    dx, [es:60h*4]
    mov    cx, [es:60h*4+2]
    mov    [es:8h*4], dx
    mov    [es:8h*4+2], cx
    sti             ; ints agains
    ;; restore values
    popa
    ret
```

```
[SEGMENT .data]         ;Initialised data segment
    VideoSegment    DD  0B8000000h ; Video memory segment
    IntSegment      DD  000000000h ; Interrupts segment
    Process         DW  0          ; which process is active
    Processes       DW  0,0,0,0,0  ; processes state
    Stack_Main      DW  0          ; stack pointer
    Stacks          DW  0,0,0,0,0  ; processes stack
    N_Processes     DW  5          ; number of process we can held
    STR_PROC_1      DB  "String 1"       ; process' 1 string
    STR_PROC_2      DB  "String 2"       ; process' 2 string
    PROC_4_MUSIC    DW  1026,2274,0FFFFh ; music to play
```

```
    STR_PROC_5_RUN  DB  "RUNNING",0      ; represents running proc
    STR_PROC_5_PAU  DB  "PAUSED ",0      ; represents paused proc
    Author1         DB  "Gustavo Sverzut Barbieri",0
    Author2         DB  "Ivens Prates Telles Alves",0


[SEGMENT .bss]          ;Uninitialised data segment
    Stack_Area    resw    128*5 ; StackSize * N_Processes
```