Practicum 4: De scheduler (1)

Computerarchitectuur

Groep 11
Mitch De Wilde & Bart Middag
2de bachelor informatica
Academiejaar 2012-2013

1. *De instructies* pushad *en* popad

De instructies **pushad** en **popad** zetten alle 32-bit (push en pop **A**ll **D**ouble) general purpose registers en halen ze respectievelijk weer van de stapel. Als de stapel initieel leeg is, ziet ze er na een **pushad**-instructie als volgt uit:

0	edi
4	esi
8	ebp
12	esp (originele waarde)
16	ebx
20	edx
24	ecx
28	eax

2. *Equivalente code met* pushad

De volgende code is equivalent met één **pushad**-instructie:

```
1  push   eax
2  push   ecx
3  push   edx
4  push   ebx
5  push   esp
6  add    dword [esp], 20
7  push   ebp
8  push   esi
9  push   edi
```

We gebruiken hier `add dword [esp], 20` omdat we de originele waarde van esp willen – en na al de push-bewerkingen ervoor is deze al 5 keer verminderd met 4, dus we compenseren hiervoor door de waarde op de stack met 20 te vermeerderen.

3. *Annotatie van het schedulingmechanisme*

```
   ORIGINELE CODE                          ANNOTATIE
1  schedulerhandler :
2      pushad                              ;zet alle registers op de stapel
3      inc dword [Huidige_Tick]            ;begin aan de volgende kloktik
4      mov al, 0x20                        ;schrijf End of Interrupt naar al
5      out 0x20, al                        ;schrijf al naar de interr. contr.
6      sti                                 ;laat interrupts toe
7      mov ebx, [Huidige_Taak]             ;adres vd huidige taak naar ebx
8      mov dword [ebx], esp                ;esp naar die plaats vd array
9      mov dword [ebx + 4], 0              ;zet huidige taak.tik op 0
10     mov ecx, [Huidige_Tick]             ;schrijf huidige tik naar ecx
11     cli                                 ;laat interrupts niet meer toe
12     mov esp, 0                          ;zet esp op 0
13 .taakzoeklus:
14     add ebx, 8                          ;takenlijst_array_positie++
15     cmp ebx, takenlijst +               ;vergelijk ebx met start
```

```
(MAX_TAKEN * 8)                 takenlijst+array_positie
    jl                          ;als we nog niet aan einde array
.nog_niet_aan_het_einde         zijn, spring naar n_n_a_e
    lea ebx, [takenlijst]       ;zet ebx op beginadres van taken
.nog_niet_aan_het_einde:
    cmp dword [ebx], 0          ;als er hier geen taak staat
    je  .taakzoeklus            ;ga dan naar volgende taak
    cmp dword [ebx+4], ecx      ;als taak_entry.tik groter is dan
    jg  .taakzoeklus            ;de huidige, voer taak niet uit
    mov [Huidige_Taak], ebx     ;dit wordt de huidige taak
    mov esp, [ebx]              ;de esp van deze taak wordt esp
    popad                       ;haal alle registers van de stapel
    iret                        ;stop de interrupt, keer terug
                                naar onderbroken prog
```

4. Het scheduleralgoritme op een hoog abstractieniveau

De volgende pseudo-C-code stelt het scheduleralgoritme voor:

```
no_interrupts void handle_schedule() {
    int tik;
    save_all_registers();
    *huidige_tick++;
    write_to_port(interrupt_controller,0x20);
    allow_interrupts {
        huidige_taak->stapelwijzer = esp;
        huidige_taak->tik = 0;
        tik = *huidige_tick;
    }
    huidige_taak_index = (huidige_taak-takenlijst)/8;
    while(true) {
        if(++huidige_taak_index >= takenlijst[MAX_TAKEN]) {
            huidige_taak_index = 0;
        }
        if(takenlijst[huidige_taak_index] == NULL) continue;
        if(takenlijst[huidige_taak_index].tik > tik) continue;
        *huidige_taak = takenlijst[huidige_taak_index];
        esp = huidige_taak->stapelwijzer;
        load_all_registers();
        return_from_interrupt();
    }
}
```

De `no_interrupts` in deze code is in zekere zin gelijkaardig met het concept van `synchronized` in Java: net zoals `synchronized` (via een lock) zorgt dat enkel de `synchronized`-blok op dat ogenblik kan worden uitgevoerd (en alle objecten die het lock willen moeten wachten), zou `no_interrupts` ervoor moeten zorgen dat er geen interrupts meer zouden mogen worden uitgevoerd tot dit weer toegelaten wordt in de `allow_interrupts`-blok.

5. De positie van `sti` en `cli`

Dat er geen `sti` meer voorkomt na de `cli`-instructie, is geen fout. De `iret`-instructie die op het einde wordt uitgevoerd zet immers alle vlaggen terug zoals ze waren voor de interrupt – de IF (interrupt flag) inbegrepen.

6. Installatie op de timeronderbreking

```
1  ; installeer de schedulerhandler op de timeronderbreking en zet
   deze onderbreking aan (Opgave 6 van de voorbereiding)
2      push    schedulerhandler
3      push    32
4      call    install_handler
5      cli                         ; disable interrupts
6      in      al, 0x21
7      and     al, 0feh            ; mask 11111110
8      out     0x21, al
9      sti                         ; enable interrupts
```