תוכניות דוגמא c_intr1d.c, c_intr2d, c_intr3d.c

התוכניות הללו הן תוכניות C, להמחשה של אספקטים של נושא הפסיקות.

התוכנית c_intr1d.c

התוכנית c_intr1d.c היא תוכנית Turbo C טהורה, להמחשה ראשונית של נושא הפסיקות. ההדגשה היא "תוכנית Turbo C" ולא סתם "תוכנית C" משום שהתוכנית הזו תתקמפל רק ע"י קומפילר של C עבור DOS (ביחוד Turbo C ו- Microsoft C) משום שהיא מישמת רעיון שבכל מערכות ההפעלה של המחשב האישי שבאו אחרי DOS אינה נתמכת יותר: שתוכנית אפליקציה משתלטת על פסיקה. תחת DOS הדבר היה אפשרי והרבה תוכניות אפליקציה עשו זאת, ותוכניות האמולציה (הדמיה) של DOS בדרך כלל משתדלות לתמוך בהדמיה כזו או אחרת של הטכניקה הזו. בגלל זה ניתן להריץ את התוכניות הללו בחלונות ה-command וה-cmd של WINDOWS אם כי ישנם תוכניות דומות (למשל כזו המסתמכת על מקש ה-Prnt-Scr) שיעבדו אך ורק תחת DOS האמיתי.

כפי שנאמר, תפקידה של c_intr1d.c לשמש המחשה ראשונית של מוסג הפסיקה. שימו לב למבנה הבא של התוכנית:

```
volatile int Ctrl_Break_Flag;

   ....

void main()
{

    ....

  Ctrl_Break_Flag = 0;

  while(Ctrl_Break_Flag == 0)
       ;

    ....

} /* main */
```

על משמעות מילת המפתח volatile נרחיב בהמשך. בשלב זה נתעלם ממנו. על פי הכללים של שפת C, Ctrl_Break_Flag הינו משתנה גלובלי שלם. במהלך הריצה של התוכנית הראשית, אנחנו מציבים לתוכו אפס ומשם נכנסים לתוך לולאה while שאינה עושה דבר למעט לבדוק אם Ctrl_Break_Flag עדיין שווה לאפס ואם כן לחזור על הלולואה. לכאורה לולאת ה-while היא בחזקת לולאה אינסופית ומבוי סתום, שכן היא תלויה בערכו של משתנה שאותו אין היא משנה תוך כדי ביצוע הלולאה. ואכן הרצת התוכנית תתקע את המחשב (במקרה של DOS) או את חלון האמולציה עד שהמשתמש ילחץ על מקש ה-Ctrl-Break.

מה שצריך להיות ברור הוא שסיום לולאת ה-while כרוך בהפרת העקרון שבו כל פקודת מכונה שמבצע המחשב נקבע ע"י פקודת המכונה שלפניה. אנחנו רואים לפי הריצה של התוכנית שזה אכן מה שקורה. הדרך שאנחנו עושים זאת היא על ידי השתלטות על פסיקת מספר 27 (1Bh בהקסה), פסיקה שמתרחשת בכל רגע שנלחץ המקש Ctrl-Break. בפקודה

```
setvect(27, Ctrl_Break_Handler);
```

קבענו את הרטינה המיוחדת Ctrl_Break_Handler בתור רטינת הטיפול בפסיקה של פסיקה 27. מרגע זה ואילך הרטינה הזו היא שתתבצע עם כל לחיצה של המקש Ctrl-Break ולא זו הרגילה (הגורמת בתנאים מסוימים חזרה ל-DOS). מאחר ואחד הפקודות של הרוטינה הזו היא

```
  Ctrl_Break_Flag = 1;
```

בפעם הראשונה שהרוטינה תתבצע הערך של המשתנה הגלובלי Ctrl_Break_Flag ישתנה ועם החזרה מפסיקה התוכנית אכן תשתחרר מלולאת ה-while. setvect היא רוטינה פנימית של Turbo C המאפשרת השתלטות על פסיקה נתונה, תוך ציון הרוטינה החלופית לפסיקה.

מההדפסות הנראות בפלט

```
C: Ctrl-Break has been pressed.
C: Terminating ...
```

צריך להיות ברור שלחיצת Ctrl-Break אכן גרמה הסתעפות לרוטינה Ctrl_Break_Handler ושעם סיומו חזר לתוכנית הראשית ומשם לסיום.

נקודות נוספות שצריך לשם לב עליהם:

- שינוי טיפול בפסיקה היא פעולה שאינה משתחזרת או מתבטלת עם סיום התוכנית. לכן בכדי שהמערכת תמשיך לתפקד עם סיום התוכנית הראשית, יש לשמר את הטיפול המקורי בפסיקה ולשחזר אותה לפני סיום. לשם כך קיימת הקריאה ל-getvect בפקודה

```
Int27Save = getvect(27);
```

getvect היא רוטינה פנימית של Turbo C המחזיר את כתובת הרוטינה הנוכחית שמטפלת בפסיקה שמספרה היא מקבלת כפרמטר. היא כמובן אינה משנה אותו. Int27Save הינו משתנה 32 ביט בזכרון שעוד נתיחס איך מגדירים אותו, אבל בסופו של חשבון הוא פשוט 32 ביט המאחסנים את התוכן הכניסה. התוכנית הראשית פשוט משמרת את כתובת המטפל הנוכחי לפני הקריאה ל-setvect ולפני הסיום משחזרת אותו ע"י הפקודה

```
setvect(27, Int27Save);
```

- בשביל לכתוב רוטינה שתטפל בפסיקה, היא חייבת להיות מוגדרת כפונקציה void interrupt עם רשימת פרמטרים void. לפיכך ההגדרה של Ctrl_Break_Handler הוא

```
void interrupt Ctrl_Break_Handler(void);
```

המילה interrupt היא מילה שמורה של Turbo C. משמעות ההגדרה הזו שהרוטינה משמרת את ערכי כל האוגרים ומסיימת את ריצתה ע"י פקודת המכונה IRET.

- ההגדרה של Int27Save

```
void interrupt (*Int27Save)(void);
```

הינו הכרזה ש- Int27Save הוא **פוינטר לפונציה מסוג interrupt void**.
מלבד הקצאת שטח מתאים (32 ביט) Turbo C יסכים לקמפל קריאה ל-setvect
ו-getvect רק למשתנים מהסוג הזה.

- המילה volatile המופיעה בהכרזה

```
volatile int Ctrl_Break_Flag;
```

מנחה את הקומפילר שהמשתנה הזה עשוי לקבל ערכים באופן לא צפוי ולכן יש לגשת לזכרון כל אימת שצריך לעמוד על ערכו. הדבר נחוץ כי אחרת הקומפילר עשוי לעשות אופטימיזציה נוסך קריאת הערך פעם אחת לתוך אוגר ולסמוך על כך שכיוון שהתוכנית לא שינתה את המשתנה, אפשר לסמוך על הערך שנמצא באוגר ולחסוך בכך גישה לזכרון. המילה volatile היא מילה שמורה של שפת C הסטנדרטית והיא רלונטית לא רק כאן.

- פקודת הקימפל של התוכנית

```
tcc -ml -r- c_intr1d.c
```

מנחה את הקומפילר לקמפל את התוכנית
א. במודל Large ע"י ההנחיה -ml
ב. ללא שימוש במשתני אוגר ע"י ההנחיה -r-.

עבור תוכניות המשתלטות על פסיקות בצורה הזו הדבר הכרחי ממספר סיבות, שלא ניכנס להם כאן.

<u>התוכניות c_intr2d.c, c_intr3d.c</u>

התוכניות הללו הן גירסאות של c_intr1d.c רק שהרטינות getvect ו-setvect מוחלפים במימושים באסמבלי. הקוד באסמבלי ממומש כאן בשיטה של inline assembly כלומר באופציה של כתיבת קוד אסמבלי בתוך תוכנית C. הדבר נעשה משיקולים של נוחות של תצוגה, השימוש בטכניקה הזו היא לאו דווקא קשור לפסיקות. בשיטה הזו ממשים קוד באסמבלי בבלוקים מהסוג

```
asm {
        קוד אסמבלי
    }
```

או בשורות קוד מהסוג

```
asm פקודת אסמבלי אחת
```

באסמבלי מהסוג הזה יש מגבלות (למשל אין תמיכה בפקודות הסתעפות מותנות). ההערות לפי המוסכמות של שפת C.

התוכנית c_intr2d.c

- בתוכנית c_intr2d.c הפעולות getvect ו-setvect נעשות ע"י גישה ישירה לזכרון לפי כתובת, כאשר התוכנית מציבה 0 לתוך ה-ES על מנת שתצביע על השטח שבו נמצאת טבלת ה-IV ובהיסטים קבועים יחסית אליו. למשל הפקודה

```
MOV AX,ES:[27*4]
```

הינו קריאה של שדה ההיסט של הכניסה מספר 27 ב-IV. הכפל הזה נעשה ע"י האסמבלר בזמן אסמבלי כלומר רק לקבועים ניתן לבצע פעולות אריתמטיות (לא ניתן לכתוב 5*BX בתוך הסוגריים המרובעים, למשל). באופן שקול ניתן היה לכתוב

```
MOV AX,ES:[108]
```

- המימושים של setvect כרוכים ביותר מפקודת מכונה אחת ולכן מקדים אותם הפקודה CLI ולאחריהם פקודת המכונה STI. הדבר נעשה על מנת למנוע תקלה כתוצאה מטיפול בבדיוק הפסיקה הזו לאחר שהחל השינוי של הכניסה ב-IV ולפני ההשלמה שלו. כמובן שהסכוי שתקלה כזו תתרחש היא אפסית, אך תמיד חייבים להתיחס לזה. למשל במימוש של setvect(27,Ctrl_Break_Handler)

```
CLI
MOV WORD PTR ES:[27*4], OFFSET Ctrl_Break_Handler
MOV WORD PTR ES:[27*4+2], SEG Ctrl_Break_Handler
STI
```

אילו לא ביצענו את ה-CLI ואם יתרחש פסיקה מספר 27 בדיוק לאחר ה-MOV הראשון ולפני הביצוע ה-MOV השני ה-CPU יסתעף לכתובת של מספר סגמנט לפי רוטינת הטיפול המקורי ואילו ההיסט לפי רוטינת הטיפול החדש. בקיצור לא פה ולא שם. העקרון הכללי הוא פשוט למנוע טיפול בפסיקות בזמן שמבנה הנתונים של ה-IV התחיל לעבור שינויים והשינויים הללו עוד לא הושלמו. מובן שאם היינו שוכחים לכתוב את CLI ו-STI סביר להניח שהשגיאה הזו לא מורגשת או באה לידי ביטוי. אך באופן כללי, חייבים להקפיד על עקרונות כאלו.

- האופרטור SEG נותנת אפקט דומה לאופרטור OFFSET רק שכאן זה קבוע של מספר סגמנט. הבדל אחר הוא שלא כל כך קל לממש אותו שכן מספר הסגמנט נקבע בזמן ריצה. יש לזה פתרון שלא נדון בו כאן.

- המשתנה Int27Save מוגדר כאן כ-long int על מנת להדגיש את העובדה שבכל התוכניות Int27Save הוא בסך הכל משתנה 32 ביט.

<u>התוכנית c_intr3d.c</u>

זוהי תוכנית כמעט זהה ל-c_intr2d.c רק ש-setvect ו-getvect ממומשים ע"י רוטינות DOS:

getvect ממומש ע"י

INT 21h, AH = 35h

"החזר ב-ES:BX את תוכן הכניסה ב-IV שמספרה מועבר אליך ב-AL".

setvect ממומש ע"י

INT 21h, AH = 25h

"הצב לכניסה ב-IV שמספרה מועבר אליך ב-AL את התוכן המועבר אליך ב-DS:DX".

```
/* c_intr1d.c - Change Ctrl-Break interrupt handler.
   Pure C version.   */

#include <stdio.h>
#include <dos.h>

volatile int Ctrl_Break_Flag;

void interrupt (*Int27Save) (void);  /* Pointer to function */

void interrupt Ctrl_Break_Handler(void)
   {
     Ctrl_Break_Flag = 1;
     printf("C: Ctrl-Break has been pressed.\n");
   } /* Ctrl_Break_Handler */

void main(void)
 {
   Int27Save = getvect(27);                /* Preserve old pointer */
   setvect(27, Ctrl_Break_Handler);           /* Set entry to new handler */
   printf("C: Press Ctrl-Break to continue.\n");

   Ctrl_Break_Flag = 0;

   while (Ctrl_Break_Flag == 0)
       ;   /* Do nothing */

   printf("C: Terminating ...\n");

   setvect(27,Int27Save);                /* Restore old pointer */

 } /* main */
```

```
E:\users\eytan\asm>tcc -ml -r- c_intr1d.c
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
c_intr1d.c:
Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4112144

E:\>c_intr1d.exe
C: Press Ctrl-Break to continue.
C: Ctrl-Break has been pressed.
C: Terminating ...

E:\>
```

```
/* c_intr2d.c - Change Ctrl-Break interrupt handler.
   C + ASM version.   */

#include <stdio.h>
#include <dos.h>

volatile int Ctrl_Break_Flag;
long int Int27Save;  /* 32bit used to hold old ISR address */

void interrupt Ctrl_Break_Handler(void)
   {
     Ctrl_Break_Flag = 1;
     printf("C+InASM 1: Ctrl-Break has been pressed.\n");
   }

void main(void)
 {
   /* Int27Save = getvect(27);    Preserve old pointer */
   asm {
         PUSH ES           /* Preserve registers  */
         PUSH AX
                           /* Preserve pointer of old interrupt handler */
         MOV AX,0          /* Set ES to point to Interrupt vector segment */
         MOV ES,AX
         MOV AX,ES:[27*4]    /* Read offset entry of Ctrl-Break ... */
         MOV WORD PTR Int27Save,AX  /* ... interrupt handler */
         MOV AX,ES:[27*4+2]    /* Read segment entry of Ctrl-Break ... */
         MOV WORD PTR Int27Save+2,AX  /* ... interrupt handler */

/*   setvect(27,Ctrl_Break_Handler);     Set new interrupt handler */
         CLI              /* Disable interrupts while IV is being changed */
              /* Set offset of new interrupt handler  */
         MOV WORD PTR ES:[27*4],OFFSET Ctrl_Break_Handler
              /* Set segment number of new interrupt handler  */
         MOV WORD PTR ES:[27*4+2],SEG Ctrl_Break_Handler
         STI              /* Re-enable interrupts (modification finished) */
         POP AX           /* Restore registers  */
         POP ES
       }

   printf("C+InASM 1: Press Ctrl-Break to continue.\n");

   Ctrl_Break_Flag = 0;

   while (!Ctrl_Break_Flag)
       ;     /* Do nothing */

/*   setvect(27,Int27Save);     Restore old pointer */
   asm {
         PUSH ES           /* Preserve registers  */
         PUSH AX
         MOV AX,0          /* Set ES to point to Interrupt vector segment */
         MOV ES,AX
                          /* Restore old interrupt handler */
         CLI              /* Disable interrupts while IV is being changed */
         MOV AX,WORD PTR Int27Save  /* Restore the offset ...*/
         MOV ES:[27*4],AX          /* ... of old interrupt handler */
         MOV AX,WORD PTR Int27Save+2   /* Restore the segment number ... */
         MOV ES:[27*4+2],AX  /* ... of old interrupt handler */
         STI               /* Re-enable interrupts (modification finished) */
         POP AX            /* Restore registers  */
         POP ES
       }
 } /* main */
```

```
E:\>c_intr2d.exe
C+InASM 1: Press Ctrl-Break to continue.
C+InASM 1: Ctrl-Break has been pressed.

E:\>
```

```
/* c_intr3d.c - Change Ctrl-Break interrupt handler.
   C + ASM version, use DOS INT 21h AH=35h, 25h.   */

#include <stdio.h>
#include <dos.h>

int Ctrl_Break_Flag;
long int Int27Save;  /* 32bit used to hold old ISR address */

void interrupt Ctrl_Break_Handler(void)
   {
     Ctrl_Break_Flag = 1;
     printf("C+InASM 2: Ctrl-Break has been pressed.\n");
   }

void main(void)
 {
   /* Int27Save = getvect(27);    Preserve old pointer */
   asm {
         PUSH DS           /* Preserve registers  */
         PUSH ES
         PUSH AX
         PUSH BX

         MOV AH,35h          /* Set DOS interrupt handler */
         MOV AL,27           /* Set interrupt vector entry number */
         INT 21h             /* Invoke DOS  */
                             /* Preserve offset of old interrupt handler */
         MOV WORD PTR Int27Save,BX
         MOV AX,ES           /* Preserve segment number of old ... */
         MOV WORD PTR Int27Save+2,AX  /* ... interrupt handler */

   /* setvect(27, Ctrl_Break_Handler); Set new interrupt handler */
                        /* Set offset of new interrupt handler */
         MOV DX,OFFSET Ctrl_Break_Handler
         MOV AX,SEG Ctrl_Break_Handler   /* Set segment number of new ... */
         MOV DS,AX                    /* ... interrupt handler */
         MOV AH,25h                   /* Set DOS interrupt handler */
         MOV AL,27                    /* Set interrupt vector entry number */
         INT 21h                      /* Invoke DOS  */

         POP BX                       /* Restore registers  */
         POP AX
         POP ES
         POP DS
       }

   printf("C+InASM 2: Press Ctrl-Break to continue.\n");

   Ctrl_Break_Flag = 0;
   while (!Ctrl_Break_Flag)
        ; /* Do nothing */

/*   setvect(27,Int27Save);     Restore old pointer */
   asm {
         PUSH DS           /* Preserve registers  */
         PUSH AX
                          /* Restore number old interrupt handler */
         MOV DX,WORD PTR Int27Save
         MOV AX,WORD PTR Int27Save+2   /* Restore the number ... */
         MOV DS,AX                /* ...  of old interrupt handler */
         MOV AH,25h               /* Set DOS interrupt handler */
         MOV AL,27                /* Set interrupt vector entry number */
         INT 21h                  /* Invoke DOS  */
         POP AX             /* Restore registers  */
         POP DS
       }
 } /* main */
```

```
E:\>c_intr3d.exe
C+InASM 2: Press Ctrl-Break to continue.
C+InASM 2: Ctrl-Break has been pressed.

E:\>
```

תוכניות דוגמא mainint3.c, ctrlbrkl.asm

התוכנית mainint3.c כמעט זהה ל-c_intrld.c, למעט העובדה שהרוטינה Ctrl_Break_Handler אינה ממומשת שם אלא מוגדרת כ-extern ומצפה שהיא תהיה ממומשת בפועל בקובץ אחר. בפועל היא ממומשת פרוצדורה בשפת אסמבלי טהורה בקובץ ctrbrhl.asm. לפיכך מה שאנחנו רואים כאן להבדיל מ- c_intrld.c היא השורה

```
extern void interrupt Ctrl_Break_Handler(void);
```

שורת הקימפול של התוכנית המשלובת היא

```
tcc -ml -r- mainint3.c ctrlbrhl.asm
```

והתוכן של הקובץ ctrlbrhl.asm. אשר לזה האחרון כדאי לשים לב לעובדה שהחזרה של הרוטינה ממומשת ע"י פקודת המכונה IRET, הרוטינה משמשרת/משחזרת את כל האוגרים בפקודות

```
  PUSH AX
  PUSH BX

  ...

  PUSH BP
```

ומשחזרת אותם בפקודות

```
   POP BP
   POP DI

   ...

   POP AX
```

מכיוון שהתוכנית מתקמפלת במודל LARGE, הקריאה ל-printf מחיבת דחיפה למחסנית של פוינטר מלא (גם מספר סגמנט וגם היסט) הממומשים בפקודות

```
   MOV AX,SEG printf_str
```

```
    ....
    PUSH AX
    PUSH OFFSET printf_str
```

בנוסף ישנה פקודה

```
  MOV DS,AX
```

בכדי להבטיח שה-printf של מודל LARGE יהיה לו את הערך שהוא מצפה לו. זאת מאחר שמדובר ברוטינת טיפול בפסיקה איננו יכולים לסמוך על ערכי העוגרים, שכן איננו יודעים בודאות מהכן הגיע ה-CPU לכאן.

```
/* mainint3.c - Main for pure assembler interrupt handler  */

#include <stdio.h>
#include <dos.h>

volatile int Ctrl_Break_Flag;
void interrupt (*Int27Save) (void);  /* Pointer to function */

extern void interrupt Ctrl_Break_Handler(void);

void main(void)
 {
   Int27Save = getvect(27);                /* Preserve old pointer */
   setvect(27, Ctrl_Break_Handler);
   printf("C: Press Ctrl-Break to continue.\n");

   Ctrl_Break_Flag = 0;

   while (Ctrl_Break_Flag == 0)
          ;  /* Do nothing */

   printf("Terminating ...\n");

   setvect(27,Int27Save);                /* Restore old pointer */

 } /* main */
```

```
E:\>tcc -ml -r- mainint3.c ctrlbrh1.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
mainint3.c:
ctrlbrh1.asm:
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland International

Assembling file:   ctrlbrh1.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  418k

Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4112112

E:\>mainint3.exe
C: Press Ctrl-Break to continue.
ASM: Ctrl-Break has been pressed
Terminating ...

E:\USERS\EYTAN\ASM>
```

```
;  ctrlbrh1.asm  - pure assembler Ctrl-Break handler.
;
        .MODEL LARGE
        PUBLIC  _Ctrl_Break_Handler
        EXTRN   _Ctrl_Break_Flag:WORD
        EXTRN   _printf:FAR
   ;
   ;    void interrupt Ctrl_Break_Handler(void)
   ;
        .DATA
printf_str   DB 'ASM: Ctrl-Break has been pressed',10,0
        .CODE
_Ctrl_Break_Handler  PROC    FAR
        PUSH    AX              ; Preseve all registers ...
        PUSH    BX              ; ... other than FLAGS and SP.
        PUSH    CX
        PUSH    DX
        PUSH    ES
        PUSH    DS
        PUSH    SI
        PUSH    DI
        PUSH    BP
        MOV     AX,SEG _Ctrl_Break_Flag       ; Set ES to segment ...
        MOV     ES,AX                       ; ...  of Ctrl_Break_Flag
   ;
   ;       {
   ;         Ctrl_Break_Flag = 1;
   ;
        MOV     ES:[_Ctrl_Break_Flag],1      ; Set global flag to 1
   ;
   ;         printf("ASM: Ctrl-Break has been pressed\n");
   ;
        MOV     AX,SEG printf_str           ; Set DS to segment ...
        MOV     DS,AX                       ; ...  of printf_str
        PUSH    DS                          ; Store complete address
        PUSH    OFFSET printf_str           ; ... of printf_str
        CALL    _printf
        ADD     SP,4                        ; Free parameter space
   ;
   ;       }
   ;
        POP     BP        ;  Restore all registers ...
        POP     DI        ;  ... other than FLAGS and SP.
        POP     SI
        POP     DS
        POP     ES
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        IRET            ; Return from interrupt
_Ctrl_Break_Handler  ENDP
        END
```