בעיות ופתרונן

# בעיית פסיקות שעון

## הבעיה

ע"מ להתחיל בכלל את תכנון המערכת, היה צורך לפתור בעיה בסיסית: בשביל preemption יש צורך לחקות את הTimer Interrupt של הBIOS שהSMARTS המקורי התבסס עליו. יש צורך בפתרון שכל זמן קבוע יפסיק את ריצת התוכנית, ישמור את המצב הנוכחי, יקפוץ לשגרה לטיפול בinterrupt ואח"כ ישחזר את המצב וימשיך לרוץ. במערכות מודרניות הגישה לInterrupt vector וממילא לפסיקה של השעון חסומה (ובצדק) ע"י מערכת ההפעלה.

נקודות לציון: מכיוון שמדובר פה במערכת שמדמה לצרכי לימוד, ולא במערכת אמתית, יש ערך גדול יותר לפתרון קל ליישום ולהבנה גם אם לא 100% מדויק, על-פני פתרון מדויק לגמרי אבל שלא ניתן לשימוש בקלות אצל תלמידים.

## פתרונות אפשריים

|  |  |
| --- | --- |
| **פתרון** | **בעיה** |
| כתיבת דרייבר (בWindows). דרייברים רצים בkernel mode, ויש להם גישה לinterrupt vector | מסובך מאוד  יצריך התקנה של הדרייבר על כל מחשב של סטודנט שרוצה להשתמש |
| SIGALRM סיגנל שמבקשים ממערכת ההפעלה לשלוח כל X מילי-שניות | עובד רק במערכות POSIX |
| SIGALRM עם שכבת התאמת POSIX ל Windows | חלק מהמתאמים לא תומכים בSIGALRM  חלק שולחים את הסיגנל ב thread נפרד  כולם מצריכים התקנה מסובכת או קימפול עם dll שלהם וקומפיילר שלהם – לא נייד כ"כ |
| Thread נפרד שילך לישון ל X זמן ואח"כ יקרא לפונקציה של ה handler | קורא ב context של ה thread של הטיימר, לא של הscheduler וכל התהליכים שלנו (אנחנו רצים בthread אחד) |

## פתרון נבחר

במערכות POSIX נשתמש בSIGALRM. בWindows נשתמש בטיימר בthread נפרד (הצעה אחרונה בטבלה), כאשר במקום לקרוא לפונקציה, נשתמש בAPI של Windows לעצור את הthread הראשי ולשנות לו את הIP, כך שירוץ ממקום שאנחנו נקבע. המקום הזה הוא פונקציית assembly שתשמור את כל הרגיסטרים, תקרא לhandler, תשחזר את המצב ותחזור לנקודה שבה הפסקנו.

יתרונות הפתרון הזה על-פני פתרונות אחרים או הפתרון של SMARTS המקורי הם:

1. ניתן לקבוע את משך הזמן של ה"פסיקה" (שלא כמו במקורי).
2. לא מצריך התקנה מיוחדת או קומפיילר מיוחד
3. יחסית לפתרון של כתיבת דרייבר הוא פשוט להבנה
4. הטיימר הוא פתרון כללי שלא קשור דווקא לSMARTS ויוכל לשמש כל תכנית שצריכה לסמלץ פסיקת שעון

חסרון הפתרון הוא שהוא לא 100% מדויק (הוא תלוי בחסדי הscheduler של Windows)

# בעיית גיבוי נתונים

## הבעיה

לצורך ביצוע Context switch, נדרשים כמה גיבויים

* Instruction pointer
* Flags
* Registers
* Stack pointer

בפתרון הקיים קיימת האפשרות להגדיר פונקציה כ Interrupt, מה שגורם לקומפיילר "לשתול" גיבוי של ערכי הרגיסטרים והדגלים למחסנית של התהליך המופרע, ובסוף הפונקציה מתבצע שחזור של ערכי הרגיסטרים מתוך המחסנית, וכך בביצוע פסיקת שעון, ערכי הרגיסטרים והדגלים מגובים מיד.

במערכת שלנו לא ניתן להגדיר פונקציה כ Interrupt כיוון ש..

## פתרון

לשם כך, נדרשנו לגבות "ידנית" את הנדרש לצורך שחזור התהליך שהופסק, בפעם הבאה שיקבל זמן CPU. כדי למנוע שינוי של ערכי הרגיסטרים אנו נדרשים לגרום ל thread לבצע את פקודות הגיבוי מיד לאחר ההפרעה ע"י הסבת IP (Instruction Pointer) לפונקציית הגיבוי.

**להלן מתוארת סדר הפעולות לצורך Context Switch כפי שמתבצעת במערכת שלנו.**

1. הרדמת ה thread של SMART.
2. שמירת ה- IP של ההוראה שהייתה אמורה להתבצע ע"י דחיפתה למחסנית.
3. דחיפת מצביע ל SMART למחסנית (פרמטר ל Interrupt handler).
4. דחיפת כתובת ה- Interrupt handler למחסנית.
5. טעינת כתובת הפונקציה doTimerAsm ל IP.
6. הערת ה thread של SMART.

כעת המחסנית של הtask נראית כך:

|  |  |
| --- | --- |
|  |  |
| Interrupt handler pointer | מצביע לפונקציה המטפלת בפסיקת שעון |
| SMART \* | מצביע ל instance של SMART של המערכת |
| Instruction Pointer | של ההוראה שהייתה אמורה להתבצע ב Task שהפרענו לו |
|  |  |

לאחר שינוי IPלכתובת הפונקציה doTimerAsm, ה task המופרע ימשיך בביצוע הפונקציה doTimerAsm המיועדת לגבות את הרגיסטרים והדגלים, המשך פעולת Context Switch:

1. דחיפת ה Flags למחסנית.
2. דחיפת ערכי 8 רגיסטרים למחסנית.

כעת המחסנית של הtask נראית כך:

|  |  |
| --- | --- |
|  |  |
| General purpose registers |  |
| Flags |  |
| Interrupt handler pointer | מצביע לפונקציה המטפלת בפסיקת שעון |
| SMART \* | מצביע ל instance של SMART של המערכת |
| Instruction Pointer | של ההוראה שהייתה אמורה להתבצע ב Task שהפרענו לו |
|  |  |

לאחר גיבוי הרגיסטרים והדגלים אנו נדרשים לקרוא ל Interrupt handlerלצורך קריאה ל scheduler.

1. דחיפת מצביע ל SMART (משמש כפרמטר ל Interrupt handler).
2. קריאה ל Interrupt handler.
3. קריאה לפונקצייה timerHandler של SMART

כעת המחסנית של הtask נראית כך:

|  |  |
| --- | --- |
|  |  |
| Return address | \_\_stdcall timerHandler |
| Return address | doTimerAsm |
| SMART \* | מצביע ל instance של SMART של המערכת |
| General purpose registers |  |
| Flags |  |
| Interrupt handler pointer | מצביע לפונקציה המטפלת בפסיקת שעון |
| SMART \* | מצביע ל instance של SMART של המערכת |
| Instruction Pointer | של ההוראה שהייתה אמורה להתבצע ב Task שהפרענו לו |
|  |  |

השלב הבא הוא לבצע החלפה מה task המופרע ל task של ה scheduler ע"י הפונקציה Task::switchTo שמפעילה את פונקציית contextSwitch.

1. קריאה לפונקציה switchTo עם מצביע למחסנית של ה scheduler כפרמטר.
2. קריאה לפונקציה contextSwitch עם שני פרמטרים, מצביע למחסנית של התהליך המופרע, ומצביע למחסנית של ה scheduler.

כעת המחסנית של הtask נראית כך:

|  |  |
| --- | --- |
|  |  |
| Return address | Task::switchTo |
| &taskPtr | מצביע למחסנית של ה Task שאותו מפסיקים |
| tsk | מצביע למחסנית של ה scheduler |
| Return address | WinSMARTS::timerHandler |
| myContext | מצביע למחסנית של ה scheduler |
| Return address | \_\_stdcall timerHandler |
| Return address | doTimerAsm |
| SMART \* | מצביע ל instance של SMART של המערכת |
| General purpose registers |  |
| Flags |  |
| Interrupt handler pointer | מצביע לפונקציה המטפלת בפסיקת שעון |
| SMART \* | מצביע ל instance של SMART של המערכת |
| Instruction Pointer | של ההוראה שהייתה אמורה להתבצע ב Task שהפרענו לו |
|  |  |

כעת הושלם כל הנדרש לצורך ביצוע פונקציית contextSwitch, לכן נבצע את הפעולות הבאות:

1. דחיפת ה Flags למחסנית.
2. דחיפת ערכי 8 רגיסטרים למחסנית.
3. שמירת כתובת ה- SP העדכנית במחסנית של ה task המופרע.
4. טעינת כתובת ה- SP של ה task של ה scheduler.

כעת המחסנית של הtask המוקפא נראית כך:

|  |  |
| --- | --- |
|  |  |
| General purpose registers |  |
| Flags |  |
| Return address | Task::switchTo |
| &taskPtr | מצביע למחסנית של ה Task שאותו מפסיקים |
| tsk | מצביע למחסנית של ה scheduler |
| Return address | WinSMARTS::timerHandler |
| myContext | מצביע למחסנית של ה scheduler |
| Return address | \_\_stdcall timerHandler |
| Return address | doTimerAsm |
| SMART \* | מצביע ל instance של SMART של המערכת |
| General purpose registers |  |
| Flags |  |
| Interrupt handler pointer | מצביע לפונקציה המטפלת בפסיקת שעון |
| SMART \* | מצביע ל instance של SMART של המערכת |
| Instruction Pointer | של ההוראה שהייתה אמורה להתבצע ב Task שהפרענו לו |
|  |  |