המדריך המאוד לא שלם להתמודדות עם Segmentation Fault

כתבו: גדי אלכסנדרוביץ', דניאל ויינשנקר ועדי וולף

22 בנובמבר 2006

1 מה זה בכלל?

כל סטודנט במת"מ נתקל בהודעה החביבה Segmentation Fault יותר פעמים משירצה, ולפעמים גם פוגש את אחיה הקטן Bus error. שתי ההודעות הללו, בשורה התחתונה, אומרות שהתוכנית שרצה זה עתה ביצעה גישה לא חוקית לזיכרון.

1.1 מהו הזיכרון ואיך ניגשים אליו בצורה לא חוקית?

הזיכרון המדובר הוא הזיכרון הראשי של המחשב (להבדיל מהכונן הקשיח, Disk-on-key וכדומה). אפשר לחשוב עליו כעל מערך גדול של תאים שיכולים להכיל מידע. במהלך ריצת המחשב נעשה שימוש אינטנסיבי בזיכרון, וכל תוכנית שאנחנו מריצים מקבלת לשימוש חלק ממנו. האחראית על הקצאות הזיכרון היא מערכת ההפעלה שרצה על המחשב. בקורס "מערכות הפעלה" תלמדו הרבה יותר על הצורה שבה ניהול הזיכרון הזה מתבצע.

השורה התחתונה פשוטה: עבור כל תוכנית מחשב אפשר לחלק את איזורי הזיכרון לשניים: כאלו שהיא קיבלה במהלך ריצתה ממערכת ההפעלה וכאלו שהיא לא קיבלה. כל משתנה לוקלי שמוקצה בתוכנית מאוחסן בזיכרון, וגם קוד התוכנית עצמה מאוכסן בזיכרון. עם כל קריאה לפונקציה מאחסנים בזיכרון את כתובת קטע הקוד שאליו יש לחזור לאחר סיום הפונקציה, וכן הלאה. כל הקצאות הזיכרון הללו מתבצעות בצורה אוטומטית ואין חשש למעבר על החוק בכל הנוגע אליהן. הסיוט מתחיל (כמו שיודע כל פוליטיקאי) כשמכניסים למשחק את המצביעים.

1.2 עוד לא אמרת איך ניגשים אליו בצורה לא חוקית!

מצביע יכול להכיל כל כתובת שהיא, לכל מקום בזיכרון של המחשב. זה כולל את הזיכרון שבו מערכת ההפעלה משתמשת, והזיכרון שבו תוכניות אחרות משתמשות (זה לא מדוייק - האמת יותר מסובכת, כפי שתראו במערכות הפעלה). בפרט, הוא יכול להכיל כתובות שלא הוקצו לשימוש התוכנית שרצה כרגע. ברגע שבו מנסים לגשת לתוכן של תא שלא הוקצה לשימוש, יכול לקרות אחד משני דברים:

- הדבר הטוב: התוכנית שלכם תקרוס ותצעק "Segmentation Fault", או סתם תמשיך לרוץ אבל תוציא פלט מקושקש.
- הדבר הרע: התוכנית שלכם תמשיך לעבוד מצויין, תרוץ עד לסופה ותוציא את הפלט המבוקש.

חשוב להדגיש כבר בשלב מוקדם זה שהתוצאה השנייה היא <u>אסון</u>. מדוע? כי היא מונעת מכם להיות מודעים לכך שיש לכם בעיה בתוכנית. אם התוכנית הייתה קורסת הייתם מתאבלים למשך זמן מה, אך לאחר מכן הייתם משנסים מותניים ונוברים בנבכי הקוד בנסיון להבין מה קרה. לעומת זאת, אם התוכנית רצה עד תומה אתם תעלזו ותשמחו ותמהרו להגיש את העבודה ובא לציון גואל. לרוע המזל, מהר מאוד מתברר שאם ביצענו גישה לא חוקית לזיכרון, ריצה מוצלחת על קלט מסויים לא מבטיחה כלום על קלט אחר, או על מערכת הפעלה קצת שונה, או בשעה קצת יותר מאוחרת. הקריסה כמעט מובטחת תחת הבודק האוטומטי, בדיוק בריצה היחידה של התוכנית אי פעם שתשפיע לכם על הציון.

1.3 אז מה <u>בדיוק</u> גורם לשגיאות הללו, ומה זה Bus error?

כל גישה (קריאה או כתיבה) דרך מצביע שאינו מצביע על כתובת חוקית עשוייה לגרום לשגיאה. בפרט, אם המצביע אותחל להכיל NULL (שהוא פשוט המספר 0), גישה אליו עשוייה לגרום לשגיאה.

את ההודעה Segmentation fault תקבלו אם תנסו להריץ את התוכנית על Unix (למשל, על השרת T2, שהתוכנות שלכם אמורות לרוץ עליו) או על Linux. בחלונות זו לא בהכרח תהיה הודעת השגיאה שתקבלו - אבל השגיאה עצמה קיימת גם שם, כמובן.

ההבדל בין Segmentation fault ובין Bus error אינו משמעותי. Segmentation fault נגרם כאשר מערכת ההפעלה תופסת את התוכנית בקלקלתה. לעומת זאת Bus error נגרם כאשר הגישה הלא חוקית חמקה מעיניה של מערכת ההפעלה והחומרה עצמה התריעה שיש כאן בעיה. כאמור, גם מערכת הפעלה וגם החומרה הן לא יוצלחיות וייתכן מאוד שתתבצע גישה לא חוקית לזיכרון שאף אחד לא ישים אליה לב כל עוד הציון בעבודה שלכם אינו תלוי בה. לכן לעולם אל תסמכו עליהן.

2 אז על מי לסמוך ומה אפשר לעשות?

ראשית כל יש לסמוך על עצמכם. כתבו קוד בצורה זהירה. אתחלו כל מצביע ל-0 וודאו שאתם בודקים שהוא שונה מ-0 לפני שאתם ניגשים לתוכנו. לאחר שאתם משחררים זכרון שהקציתם למצביע, שימו בו שוב 0. עם זאת, אל תטעו לחשוב שביצוע ההוראות הללו מבטיח תסינות.

הדרך הטובה ביותר לאתר שגיאות שכאלו היא באמצעות תוכנה חכמה שמריצה את התוכנית בצורה מבוקרת ומתריעה על גישות לא חוקיות לזיכרון. תוכנה שכזו עבור לינוקס נקראת Valgrind (עוד מידע עליה באתר הבית שלה: valgrind.org). שימו לב - התוכנות הללו אינן צריכות שהתוכנה שלכם תקרוס בכדי להתריע על שגיאות - הן מתריעות על כל גישה לא חוקית, לא משנה מה תוצאותיה.

לפעמים לא תוכלו להשתמש ב- Valgrind ודומותיה, ועבור מקרים עצובים אלו, מומלץ לחזור להדפסות הדיבאג הישנות והטובות. אל תתקמצנו - קשה מאוד לנחש היכן מתרחשת שגיאת זכרון, ולפעמים לוקח הרבה זמן להאמין שהיא אכן איפה שכבר החלטתם שאינה. לכן עדיף לשים יותר מדי הדפסות מאשר מעט מדי. שימו לב שאם הדפסות הדיבאג שלכם נשלחות לקבצים, שגיאת זכרון תמורה עלולה למנוע מהקובץ להסגר וגרוע מכך, עלולה לגרום לסיפא מהדפסות הדיבאג שלכם להיעלם. אז כדאי להשתמש ב- fflush (זוהי פקודה ש"מכריחה" ביצוע של כתיבה לקובץ - עוד מידע עליה ניתן למצוא בחיפוש בגוגל).

כדי להיות מסוגלים לזהות בצורה יעילה מה גורם לשגיאה בתוכנית שלכם, כדאי להיות מודעים לכל הדרכים האפשריות שבהן שגיאות שכאלו עלולות לצוץ. הנה רשימה חלקית:

2.1 הקצאות זיכרון חלקיות

אם אתם מקצים 10 בייטים ואז ניגשים לבייט ה-11 מובטח טראת. למה שתעשו דבר שטותי כזה? הנה מספר אפשרויות:

2.1.1 הקצאת מחרוזת קצרה מדי

זוכרים שצריך להוסיף עוד בייט בשביל ה-0\ שבסוף? מתברר שלא כולם זוכרים, או שהם משוכנעים ש-Strlen מחזירה בתור אורך המחרוזת גם את הבייט הנוסף הזה. היא לא.

2.1.2 שימוש מופרך ב-sizeof

חשוב להבהיר ש-sizeof הוא אופרטור שמקבל משתנה או טיפוס של משתנה ולא שום דבר אחר. בפרט הוא לא צריך לקבל מספר. ישנם כאלו שמפעילים Strlen על מחרוזת, את התוצאה מעבירים ל-sizeof ואחר כך מחפשים בקוד של 4,000 שורות סיבות למה התוכנית שלהם כל הזמן קורסת על יוניקס למרות שהיא עבדה כמו חלום על ווינדוס. לא חבל?

2.1.3 שימוש עוד יותר מופרך ב-sizeof כשמקצים structs

בזכות שיטת הלימוד הברורה של מת"מ מאוד קל להבדיל בין מבנים ובין מצביעים למבנים. טיפוס של מבנה מכונה, למשל MyStruct_t (שימו לב ל-t שבסוף!). לעומתו, המצביע לאותו מבנה מכונה MyStruct. נכון שמאוד ברור וקל להבין ש-MyStruct הוא מצביע? לא? אז זה כנראה מסביר למה לעתים קרובות מעבירים ל-sizeof אותו במקום את המבנה עצמו.

למען הסר ספק, השורה הבאה היא שטות גמורה:

```
MyStruct myVariable = (MyStruct)malloc(sizeof(MyStruct));
```

ואילו השורה הבאה היא נכונה:

```
MyStruct myVariable = (MyStruct)malloc(sizeof(MyStruct_t));
```

קל לראות את ההבדל העצום בין שתי השורות, נכון? לכן כדאי לאמץ כלל אצבע: לוודא שמה שמופיע בתוך ה-sizeof תמיד שונה ממה שמופיע בשאר השורה (כלל אצבע נוסף - לסמן מצביעים באמצעות תוספת p לסוף שם הטיפוס).

2.2 "אפס אחד!" - גרסת המערך

אחד מהדברים הנפלאים ביותר בשפת C (ו-C++, ו-Java, ו-Perl, ועוד ועוד) הוא העובדה שמערכים מתחילים מ-0 ולא מ-1. באמת. אני לא צוחק. למשל, בזכות זה מאוד קל לתרגם את הקואורדינטה (x, y) למספר בודד ($x \cdot rowLength + y$). לרוע המזל אם מערך בגודל n מתחיל מ-0 הוא מסתיים ב-n-1, וזה מבלבל. אוהו, כמה שזה מבלבל. חשבו היטב לפני כל לולאת for שאתם כותבים האם תנאי העצירה שלכם בסדר או שהוא גדול קצת יותר מדי. בכל פעם שבה אתם ניגשים למערך שלא דרך משתנה האינדקס של לולאת for, חשבו היטב הרבה יותר האם המספר שאתם ניגשים דרכו אכן נמצא על הסקאלה 0...n-1 ולא על הסקלה 1..n. למשל, אם יש לכם מערך עבור חודשי השנה, ואתם מקבלים מהמשתמש את הקלט המספרי שמציין את חודש אוגוסט - 8 - עליכם לוודא שאתם מעבירים למערך את המספר 7 דווקא.

2.3 הזבל לסל ולא חסל

טוב, זה סעיף כמעט מיותר, ובכל זאת: אם השתמשתם ב-free כדי לחסל מצביע. אנא, אנא, אנא אל תמשיכו להשתמש בו אחר כך!

למעשה, זה לא טריוויאלי כפי שניתן לחשוב אם התוכנית שלכם סובלת מאחת הקללות הנפוצות של שימוש במצביעים: שימוש במספר מצביעים שונים כדי להצביע על אותה הכתובת. לפעמים יש לגישה הזו שימושים מוצלחים, אבל בקורס מת"מ לרוב היא אסון - בפרט כשלא מתכוונים אליה.

איך תאונה כזו יכולה להתרחש? למשל, כאשר יש לנו struct שאחד משדותיו מכיל מצביע, ואנחנו מעתיקים אותו ל-struct אחר באמצעות האופרטור =. כל השדות יועתקו על ידי השמה פשוטה, כולל השדה של המצביע. קיבלנו שני struct-ים שבשניהם יש מצביע לאותו המקום. אם נחסל את אחד ה-struct-ים בצורה שיטתית שתכלול את שחרור הזיכרון לכל המצביעים שבתוכו, נחסל גם את המצביע שב-struct השני, ויחד איתו - את כל התוכנית שלנו.

2.4 להערים על המחסנית

כאשר אנו מקצים זיכרון עם malloc, הזיכרון נלקח ממקום שמכונה "הערימה" (heap). לעומת זאת, הזיכרון של משתנים לוקאליים מוקצה במקום שונה לגמרי, שנקרא "המחסנית" (stack). למחסנית יש תכונה נפלאה: היא מתרוקנת מעצמה. ברגע שבו אנחנו יוצאים מפונקציה, החלק במחסנית שהכיל מידע על הפונקציה, ובכלל זה גם על המשתנים הלוקאליים שלה, מחוסל. פירוש הדבר הוא בפרט שכל מצביע למשתנה לוקאלי הופך לאויב המדינה ברגע שבו הפונקציה שבה המשתנה הלוקאלי חי מסתיימת. מאותו הרגע, הכתובת שבמצביע אינה חוקית וגישה אליה תחולל ניסים ונפלאות. לכן הכי טוב לאמץ את כלל האצבע הפשוט: לא להכניס למצביעים כתובות של משתנים לוקאליים. חריג מכלל זה הוא העברת פרמטרים לפונקציות by reference, אך במקרה זה הסכנה ממילא לא קיימת שכן המצביע עצמו מתחסל ביציאה מהפונקציה שאליה הועברו הפרמטרים - הרבה לפני שהפרמטרים עצמם מחוסלים.

כמובן שיש עוד חריגים - המטרה אינה למנוע מכם להצביע למשתנים לוקאליים, אלא לגרום לכם לעצור ולשאול את עצמכם בכל פעם שבה אתם עושים זאת "האם זה נתוץ?" ו"האם אנחנו נזהרים"? ודי בכך.

כשתלמדו C++ תצוץ בעיה דומה בכל הנוגע להחזרת references למשתנים לוקאליים. העקרון נותר אותו עקרון, ולכן לא נפרט עליו כאן.

2.5 מחסנית VS ערימה חלק 2: הנקמה

במובנים מסוימים, המחסנית מרושעת הרבה יותר מאשר הערימה, כמקום לעשות בו טעויות. קודם כל, כי סביב קטע זיכרון המוקצה בערימה, לעתים קרובות יהיה זיכרון שאינו מוקצה לתוכנית שלכם, ולכן גישה אליו תגרום לתעופה (כזכור, זה דבר טוב). לעומת זאת, אם הקצתם משתנה על ראש המחסנית, תוכלו לגשת בקלות לזיכרון בחלקים עמוקים יותר במחסנית, שהיות והם אכן מוקצים לתוכנית שלכם, לא יראו מוזר כלל וכלל למערכת ההפעלה. גם השטח הפנוי מעל למחסנית סביר להניח שכבר מוקצה לתוכנית, ולכן לא יגרום לאזהרה כלשהי.

כמובן, חלק מאותם משתנים בעומק המחסנית אינם שייכים לפונקציה הפעילה באותו הרגע, לכן כל תוכניתן סביר יאמר לכם שזה בלתי אפשרי בעליל שהם ישתנו, לכן אין בזמן הדיבאג טעם לבדוק את ערכם כלל; כפי שאולי הבנתם כבר, למילה "סביר" יש מעט ערך עבור תוכניתן בשפת C. אם לא הבנתם זאת עדיין, נזכיר לכם את האמירה המוקדמת יותר כי המחסנית כוללת גם ערכים מסתוריים יותר מאשר הנתונים שלכם, כגון המצביע לנקודה בקוד אליה צריך לחזור מהפונקציה הנוכחית. וכן כן, גם מצביע זה ניתן "לקדם" בטעות, עם תוצאות בלתי סבירות למדי. לנזהרים בשימושם במצביעים, הצורות הנוחה ביותר לשנות חלקים מעניינים במחסנית קשורות לרוב למערכים המוקצים על המחסנית, ולעובדה של-C אין כל עניין למנוע מכם לגשת לכל תא במערך, בין אם הוא שייך לו בפועל או לא, ובין אם מספר התא חיובי או שלילי...

2.6 אין מה להשוות

גורם התמותה מספר אחד בעולם שפת ה-C הוא ההבדל הקטן בין סימן השווה (=) וסימן השווה שווה (==). הראשון הוא השמה, השני הוא השוואה. לרוע המזל, הקלדה כפולה על אותו מקש היא משימה מעייפת והגוף נוהג להחליט שדי בהקלדה אחת. בשל כך, בפעמים רבות שבהן מבקשים לבצע השוואה, מתבצעת במקום זאת השמה. בזכות החלטה נבונה של מעצבי C כל מספר יכול להופיע בתור תנאי, ולכן הקומפיילר לא יצעק על הבלבול הזה (בשפת Java, למשל, הבעיה לא קיימת למרות ש-= עדיין משמש להשמה ו-== להשוואה).

דוגמה? אם יש לכם מצביע disastrousPointer ואתם רוצים לבדוק האם הוא לא מצביע על שום דבר, אתם רוצים לבצע את הבדיקה הבאה:

```
if (disastrousPointer==0) printf ("Bad pointer!\n");
```

הרי אתם תלמידים טובים ואתם מקשיבים לעצתנו מקודם - כל מצביע שלא מצביע על שום דבר מאותחל לאפס, ולכן זו ההשוואה שיש לבצע.

אבל אבוי, האצבע שלכם החליקה וקיבלתם את השורה הבאה (מצאו את ההבדלים):

```
if (disastrousPointer=0) printf ("Bad pointer!\n");
```

עכשיו, בואו נניח לרגע שהאדון disastrousPointer דווקא מצביע על משהו חוקי. מה קרה? גם איבדתם את המשהו החוקי, אבל חמור מכך - עכשיו הביטוי שמופיע בתוך הסוגריים של ה-if הוא המספר 0 (הערך של המצביע). כלומר, התנאי שבתוך הסוגריים לא מתקיים, לא יודפס "Bad pointer!" ואתם תחשבו ש-disastrousPointer הוא מצביע מאותחל וטוב לשימוש מיידי. את ההמשך אני משאיר לדמיונכם הפרוע.

שגיאה משונה עוד יותר שקרתה לי אישית מתרחשת כשמתבלבלים עם אופרטור האי-השוואה. למשל, כשעושים את הבדיקה הזו:

```
if (disastrousPointer=!0) printf ("Bad pointer!\n");
```

מה יקרה כאן? מכיוון שהאופרטור שבודק אי שוויון הוא =! ולא !=, מן הסתם לא ייבדק אי שוויון. במקום זה יוצב ב-disastrousPointer הערך של 0!, כלומר 1. נסו לחשוב מה יקרה כאשר הבדיקה מתבצעת במקרים מורכבים יותר...

הפתרון? טריק נפוץ כדי למנוע את ההשמה השגויה הזו היא להפוך את הסדר, כלומר לבצע את הבדיקה כך:

```
if (0==disastrousPointer) printf ("Bad pointer!\n");
```

מכיוון ש-0 הוא לא משתנה, אם בטעות נכתוב השמה במקום השוואה, הקומפיילר יזעק שאי אפשר לשים ערך בתוך 0. כמובן שזה פותר רק מקרים שבהם ההשוואה היא למשהו קבוע, ולא בין שני מצביעים. לכן חשוב להיזהר כשכותבים השוואות ולוודא שלא עשיתם את הטעות הזו (וכולם עושים אותה מתישהו), וכאשר אתם מבצעים debugging ורואים שערכו של מצביע (או סתם משתנה כלשהו) משתנה בלי הסבר למרות שלא היה אמור להשתנות, תזכרו את הבעיה הפוטנציאלית הזו.

3 תם ולא נשלם

לסיכום, חשוב להבין שתכנות אינו מדע (פרט לכך שגם בתכנות וגם במדע דברים מתפוצצים) אלא אמנות (כלומר - אין בו כללים חד משמעיים, והרבה ממנו מבוסס על הסגנון האישי של המתכנת), וכמו בכל אמנות, אם לא רוצים שהקהל ישליך עלינו עגבניות רקובות כדאי שנחפש אישור ממקור חיצוני לכך שהאמנות שלנו טובה. נסו להשתמש בתוכנה לחיפוש בעיות זכרון אם אתם מסוגלים - הדבר יחסוך לכם זמן וסבל רב.

אם אינכם מסוגלים, אין ברירה אלא לבצע debugging ארוך ומתיש. בתקווה, הבעיה שלכם היא אחת מבין הבעיות הנפוצות שתוארו לעיל, ובמקרה זה מציאתה תהיה קלה יחסית. אם אתם מצליחים להמציא סוג נוסף של שגיאה שמובילה לגישה לא חוקית לזיכרון, אתם מוזמנים לשלוח לי (gadial@tx.technion.ac.il) אותה ואעדכן.