To Boot or not to Boot?

הסרט "בלייד ראנר" (1982) עוקב אחר רוצח ברשיון שעבודתו היא להרוג אנדרואידים שהסתננו לכדור הארץ. האנדרואידים הם בני-אדם שיוצרו באופן מלאכותי ומשמשים כעבדים מסוגים שונים במושבות מרוחקות בחלל. יצרני האנדרואידים גילו שזכרונות שהאנדרואידים אוספים במהלך חייהם מייצרים אצלם תגובות רגשיות, ואלה מפריעות ליכולת לשלוט בהם ולנצל אותם. הפתרון של היצרנים אכזרי אך פשוט: זמן חיים מוגבל. כל אנדרואיד חי רק ארבע שנים.

כל תוכנה שרצה לאורך זמן, אוספת "זכרונות". זה יכול להיות נתונים בקבצי לוג; שינויים במידע הנצבר בבסיס נתונים; שאריות של "לכלוך" במקומות שונים ב-RAM ועוד ועוד. אחד הסיכונים שטמון בתופעה זו היא שבמקרים מסוימים זה יכול לגרום לתקלה במוצר. לפעמים זה עקב באג בקוד של המוצר עצמו: המפתח שכח לשחרר קבצים כמו שצריך ולאחר זמן ריצה ארוך הדבר מכלה את משאבי המערכת. במקרים אחרים זו אינטראקציה עם מערכת ההפעלה: קובץ הלוג גדל מעל גודל הקובץ המקסימלי שמערכת ההפעלה יודעת לפתוח; או שכל פתיחה של הקובץ לוקחת זמן רב וגורמת להפעלה של watchdog timer. ויש עוד הרבה תקלות שמופיעות רק לאחר הרצת המערכת לאורך ימים או שבועות.

אחת הדרכים להתמודד עם סיכונים אלה היא הרצת טסטים ארוכים מאוד. בקבוצות שבהם עבדתי, בדיקות אלה נקראו "Continuous Operation" או בקיצור "CO". בדיקות CO מפעילות את המערכת לאורך שעות רבות ולעיתים אף ימים ועוקבות שהמערכת מתנהגת באופן רצוי. אפשר לעקוב אחרי השימוש בזיכרון על מנת לתפוס זליגת זיכרון; אפשר לעקוב אחרי משאבים אחרים, כגון file handles או מצב ה-stack; אפשר לבצע מידי פעם פעולה פונקציונלית – לווידוא תקינות המערכת ואפשר למדוד ביצועים על מנת לזהות ירידה במהירות העבודה או עלייה בצריכת ה-CPU או הזרם.

באגים אחרים ש-CO מתוכנן לתפוס הם אלה הנובעים משגיאה מצטברת: עיגול תוצאה שמתחיל להשפיע לאחר מיליוני חישובים; טעות חישובית שמשפיעה על חישובי המשך לאורך זמן; סטיה מצטברת של שעונים עקב חוסר דיוק בחומ שמייצרת את תדר השעון הבסיסי וכדומה.

מעבר למטרה של זיהוי באגים שמופיעים רק לאחר הרצה ארוכה מאוד, מטרת בדיקות CO הינה זיהוי "באגים סטטיסטיים": כאלה המופיעים בצירוף מקרים נדיר. למשל, קריסה המופיעה כשתהליך כלשהו של מערכת ההפעלה פועל בדיוק כאשר המערכת שלנו מנסה לכתוב קובץ לוג ספציפי. כאן פשוט משחקים עם הסיכויים: אם התהליך הקריטי מופעל פעם בדקה ל-5 מילי שניות, ואילו אנו כותבים ללוג פעם בשעה, הרי שצריך מזל די רע כדי שהכתיבה תיפול בדיוק כשהתהליך הקריטי רץ. אבל אם נריץ את המערכת שלנו 200 שעות, יש סיכוי לא רע שזה יקרה לפחות פעם אחת.

בבדיקות CO יש שתי בעיות עיקריות: הראשונה היא שקשה להגדיר כמה זמן נחשב "ארוך מספיק". נניח שהחלטתם על 48 שעות. למה קבעתם את המספר הזה? איזה הגיון יש מאחורי ההחלטה? יותר הגיון מ-24 שעות? יש צורך עסקי ביותר מ-10 שעות? ללא דרישה ברורה ומנומקת לאור שעות פעולה רצופה, הרי שמי שבפועל קובע אר הגבול אלה הבודקים - או אולי מנהל המוצר - ע ידי ההחלטה כמה שעות נריץ CO. הרבה פעמי ההחלטה נלקחת מתוך שיקולים של לוגיסטיקה, ניחוש מושכל או סתם מספר מצוץ מהאצבע שנראה טוב.

הבעייה השנייה היא שהמחיר של בדיקות אלה הוא גבוה. אני לא מדבר על עלות ישירה: כמה כבר עולה לשים מחשב ולתת לו לרוץ שבוע? העלות היא עקיפה. קודם כל, מדובר בבדיקה ארוכה. אם מתגלה באג לאחר הרצה של 15 שעות, יתכן ששיחזור הבאג גם ידרוש 15 שעות... ולא תמיד מצליחים בהרצה הראשונה... בנוסף: אם יש תנאי שאי אפשר לשחרר גירסה ללא תוצאות CO, יתכן שבדיקה זו תקבע את אורך הזמן המינימלי שסבב בדיקות לוקח.

אבל לא בזה אני רוצה לדון היום.

במקום לשבור את הראש איך להבטיח שנגלה באגים שקורים רק אחרי זמן עבודה רב, יש כיוון אחר שצריך לדעתי לבחון ברצינות עבור כל מוצר – וודאי אם המוצר הוא תוכנה במערכת משובצת מחשב (embedded). הרעיון הוא פשוט – ממש כמו בבלייד ראנר: נאלץ את התוכנה לאתחל (לעשות reset, "לרסט") מחדש כל פרק זמן מסוים. לצורך הדיון נניח "כל 24 שעות" – למרות שזה מספר שרירותי לגמרי; בפרויקט אמיתי צריך להשקיע מחשבה ולהגדיר זמן התואם את המוצר והשימוש בו . הרעיון, אגב, אינו שלי; שמעתי אותו בכנס בפריס לפני למעלה מעשר שנים ואני יודע שיש מערכות שעושות זאת. ניר גלנר הכיר לי את הביטוי "Reboot בריאות".

יש תקלות שמופיעות רק לאחר הרצת המערכת לאורך ימים או שבועות

קודם כל צריך לשאול את עצמנו מי קבע ולמה, שהמערכת שלנו צריכה לרוץ תקופות ארוכות ללא אתחול. לפעמים, זו אכן הדרישה: אם המערכת מבקרת פעילות קריטית, שבה יכול להתרחש מאורע רע תוך שניות (למשל: קוצב לב), הרי שהיא לא מתאימה לריסט תקופתי. לעומת זאת, יש המון מערכות שבהם אתחול זריז בשעות שאינן בשימוש יכול לחסוך הרבה כאבי לב: המשפחה שלי, למשל, יודעת שאני לא מוכן לקבל טלפון של "אבא, למה אין אינטרנט" לפני שכיבו והדליקו את הנתב בתור "תפעול מעצור ראשון". גם מערכות מסובכות, כמו CT, אפשר (מבחינה תפעולית) לרסט כל יממה. צריך כמובן לוודא שהמכונה לא סורקת מישהו כרגע, אבל אחרי שעברו 24 שעות והסריקה האחרונה הסתיימה, המערכת יכולה לסרב להמשיך לסריקה הבאה אם לא בוצע אתחול.

אז בהנחה שהמערכת שלנו מתאימה, שהקוד המתאים הוכנס למערכת ושדאגנו לבדוק שאכן הוא עובד נכון - אנו משיגים מספר יתרונות בהחלטה לרסט כל 24 שעות.

אתחול זריז בשעות שהמערכת אינה בשימוש יכול לחסוך הרבה כאבי לב

1) אם קודם קבענו את אורך הרצת CO ל-72 שעות כשילוב של אינטואיציה, תקווה חסרת בסיס ואילוצי הפרויקט, הרי עכשיו יש בסיס עובדתי להחלטה להריץ 24 שעות: בשטח המערכת לא תרוץ יותר מזה לפני שתעבור אתחול.

מיכאל שטאל
ארכיטקט בדיקות תוכנה באינטל, ישראל
עוסק בעיקר בבדיקות מערכות משובצות מחשב. במסגרת תפקידו, מיכאל מגדיר שיטות בדיקה ומתודולוגיות עבודה, עוסק בהדרכה ולפעמים אפילו מרשים לו לבדוק משהו (שזה הכי כייף).
מיכאל מציג תכופות בכנסים בארץ ובחו"ל ומלמד בדיקות תוכנה בפקולטה למדעי המחשב באוניברסיטה העברית.
ניתן לראות חלק מהמצגות והמאמרים שלו באתר www.testprincipia.com.

מחפש צרות | מיכאל שטאל

2) חסכון בזמן (זמן הרצה; זמן שחזור ודיבוג).

3) מניעת הופעה של באגים מסוימים, עקב העובדה הפשוטה שבאגים אלה צריכים יותר מ-24 שעות על מנת להתפתח (למשל שגיאות מצטברות).

הערה על יתרון (3): נוצרת קצת הרגשה שמשהו כאן לא מוסרי. אנחנו מתעלמים מהסיכון שיש באג בתוכנה, על ידי הטענה ש"דאגנו שהוא לא יופעל גם אם הוא קיים". האם זו אכן בעיה? לדעת לא. אנחנו משחררים תוכנות לשטח מתוך ידיעה ש באפשרותנו למצוא את כל הבאגים ולוקחים את הסיכון שחלק מהם יקרו. כאן מצבנו עדיף: אנחנו מגדירים מעטפת פעולה שמונעת מהבאגים להופיע בכלל. גם בחיים הרגילים זו גישה מקובלת: בכבישים מניחים מעקה הפרדה מבטון בין נתיבים. זה לא פותר בעיות של סטייה מהנתיב עקב מהירות מופרזת או שתייה, זה פשוט מונע את האפשרות של התנגשות חזיתית.

אבל אין ארוחות חינם, הרי יש עוד סוג של באגים שהיו על הכוונת: הבאגים הסטטיסטיים. אם עכשיו נריץ רק 24 שעות, הרי שהורדנו את הסיכוי לתפוס אותם! לבעייה זו יש פתרון שמתאפשר עקב ההחלטה לרסט כל 24 שעות: נריץ מספר מערכות במקביל, כל אחת ל-24 שעות. הרצת שלוש מערכות כאלה במקביל היא שוות ערך להרצה של מערכת אחת ל-72 שעות – לפחות מבחינת הסיכוי להופעת באגים סטטיסטיים.

אם נראה לכם שכדאי לשקול אתחול תקופתי של המוצר שלכם, יש עוד קצת עבודה לפני שרצים קדימה. צריך לנתח לעומק עד כמה האתחול שלכם באמת מאתחל את המערכת באופן מלא. האם בכל זאת נשאר משהו מההרצה הקודמת? למשל, רשומות בבסיס נתונים; ערכים שנרשמו בזמן ההרצה לזיכרון בלתי נדיף (non-volatile) ויהיו שם גם לאחר האתחול; קבצי לוג שלא נמחקו וכו'. יתכן שתגיעו למסקנה שיש לעשות עוד כמה צעדים בקוד על מנת להגיע לאתחול נקי מספיק. למשל, שינוי שם קבצי לוג, על מנת שבהרצה החדשה הלוג יאותחל לקובץ חדש. אני מניח שכמעט ואין מקרים בהם כיבוי והדלקה פשוטים מחזירים את המערכת ממש ממש למצב בו היתה קודם, ויש מקרים רבים בהם אנו בכלל לא רוצים שהמערכת תשכח כל מה שהיה לפני האתחול. צריך לחשוב טוב מה לשמור גם באתחול, להעריך עד כמה הדבר ישפיע על המשך תקין של עבודת המערכת או כמה מאמץ להשקיע בלאפס דברים.

ולסיום שתי דוגמאות התומכות ברעיון:

דני אלמוג הזכיר לי את הבאג שהתגלה במערכות טיל הפטריוט במלחמת המפרץ הראשונה. טעות מצטברת באחד החישובים גרם להפרש בין שעונים במערכת, ולטעות בזיהוי הצורך לשגר טיל יירוט. בצה"ל זיהו את הבעייה ומצאו דרך עקיפה לבעיה עד שהבאג יתוקן: אתחול של המערכת מידי יומיים. צבא ארה"ב לא עשה זאת, והתוצאה הייתה פגיעה קטלנית של טיל סקאד ב-28 חיילים.

במערכת שבדקתי לפני שנים הוכנס, כחלק מהארכיטקטורה, watchdog timer שבדק אם חלק מסוים במערכת תקוע יותר מ-2 מילי שניות. אם התברר שכך הוא, וממש לא משנה למה, המערכת ביצעה אתחול פנימי של המעגלים הרלוונטיים. הפעולה הייתה מהירה והמשתמש כלל לא הבחין שקרה משהו חריג. זה לא תיקן את הבעיות שגרמו לתקיעה (אם כי עקבנו אחרי הלוגים של מקרי האתחול ולאט לאט תיקנו אותם). אבל גם לפני התיקונים קיבלנו מערכת שכמעט אף פעם לא נתקעה.

לסיכום

- בדיקות לאורך זמן מוצאות תקלות חשובות, חלקן כתוצאה ישירה מההפעלה הארוכה וחלקן סטטיסטיות
- במוצרים מסוימים ניתן לעקוף את הבעיות הנובעות מהרצה ארוכה על ידי קביעת משטר אתחול כחלק מהגדרות המוצר
- על מנת להמשיך ולגלות תקלות סטטיסטיות ניתן להריץ מספר מערכות במקביל לזמן קצר יחסית

היתרון בקביעת משטר אתחול הוא קביעה מושכלת וניתנת להצדקה של זמן בדיקות ה-CO, הפחתה בעלויות של הרצת הבדיקות ותיקון הבאגים שהן מגלות.

תודות

מאמר זה נכתב לאחר דיון בפורום בדיקות ואיכות תוכנה ב-Facebook ודרך דוא"ל עם מספר חברים. השתתפו ותרמו (לפי סדר א"ב): יונית אלבז, דני אלמוג, יאן ברון, ניר גלנר, קובי הלפרין, יאן ואן-מול, דביר יוסקוביץ, נועם כפיר, טל פאר, ירון צוברי ויונתן קליין.

אפשר לקבל רעיונות איך לבצע ולהפיק תועלת מקסימלית מבדיקות לאורך זמן במאמרים של פרופ' קם קנר (Cem Kaner) בנושא של "High-volume Automated Testing" (חפשו בגוגל).