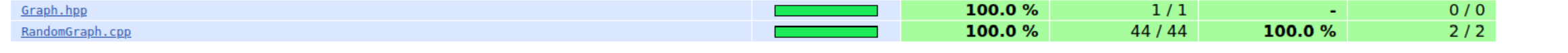
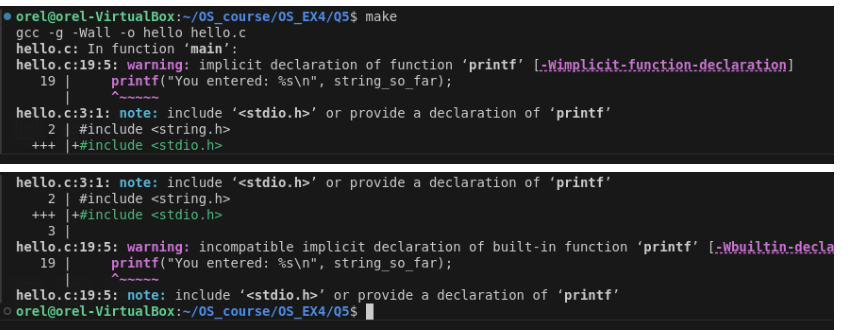
מטלה מספר 4 – מערכות הפעלה   
 שאלה מספר אחת :  
בשאלה זו השתמשנו בטכניקה שנלמדה בקורס בדיסקרטים בשביל לדעת האם הוא גרף הוא אויילרני אם כל צלע הדרגה שלה היא זוגיות ובכך בדקנו האם הגרף אוליינרני או לא .   
יש לנו אוביקט של גרף ויש לנו את הקובץ של randomngraph אשר מהווה עבורנו כקובץ main   
חלק שלוש אנחנו מגנרט'ים מספרים אקראים בטווח שבו אנחנו חסומים בו ולאחר מכן מרצים את האלגוריתם   
  
בתוך הקישור לgithub ניתן לצפות במידע אשר מתקבל מכל אחד מהבאים   
gprof,gocv,valgrind,valgrind callGraph   
<https://github.com/nissan259/OS_EX4/tree/main/Ex4_files>

1. זה ה-gprof הסיבה שאני מצרף אותו זה כי בgithub אי אפשר לפתוח את זה בפורמט html וזה רושם את זה כקוד   
   תמונה שמכילה טקסט, צילום מסך

   התיאור נוצר באופן אוטומטי  
   תמונה שמכילה טקסט, צילום מסך

   התיאור נוצר באופן אוטומטי  
     
     
     
     
     
     
     
   שאלה 4   
     
     
   ניתן לראות כי ישנה בעיה מכיוון שלא נעשה include <stdio.h>#  
   בנוסף לכך ניתן לשים לליקויים הבאים בקוד ניתוח על ידי Valgrind/memcheck   
   סוג הבעיה: שימוש בערך לא מאותחל (Uninitialized Value)

איפה היא נמצאת: השגיאה מתרחשת בשורה 10 בקובץ hello.c. שם נעשית השוואה בין המשתנה string\_so\_far לבין (char \*) 0, אך המשתנה לא מאותחל לפני כן.

איך לפתור: צריך לאתחל את המשתנה string\_so\_far לפני שמשתמשים בו. ניתן לאתחל אותו לערך מתאים כמו NULL או מחרוזת ריקה, כדי להבטיח שאין גישה לערכים אקראיים בזיכרון.

סוג הבעיה: דליפת זיכרון (Memory Leak)

איפה היא נמצאת: Valgrind מדווח על דליפה של 9 בתים בזיכרון. המשמעות היא שזיכרון שהוקצה במהלך התוכנית לא שוחרר כראוי לפני סיום התוכנית.

איך לפתור: יש לוודא שכל זיכרון שהוקצה דינמית (באמצעות malloc או פונקציות דומות) משוחרר בסיום השימוש בו באמצעות הפונקציה free().

סוג הבעיה: כתיבה או קריאה לא חוקית בזיכרון (Invalid Read/Write)

איפה היא נמצאת: בשורות 11 ו-14 יש גישה לא חוקית לזיכרון, כלומר, כתיבה או קריאה מחוץ לתחום הזיכרון שהוקצה למשתנה. כאן כנראה הוקצו 9 בתים, אבל התוכנית מנסה לגשת ליותר מזה.

איך לפתור: יש לוודא שהקצאת הזיכרון מספיקה עבור הנתונים המיועדים. אם מדובר במחרוזות, חשוב לוודא שהוקצה מספיק מקום גם עבור תו הסיום \0.

בסעיף מספר שש ניתן לראות שבאמצעות הdebuger אנחנו מקבלים חשיפה לבעיה בצורה הרבה יותר נוחה תוך הבנה של מה באמת קורה ואיפה הקוד נפל לדוגמא   
תמונה שמכילה טקסט, צילום מסך, תוכנה, אתר

התיאור נוצר באופן אוטומטי  
  
התהליך שבוצע הוא פתחה של שני טרמנילים במקביל שמהמטרה שיוקצה pid לטובת הvalgrirnd   
ואילו הטרמניל השני יאפשר לנו לדאבג את הקוד בחופשיות ובכך לראות היכן נמצאות הבעיות כפי שניתן לראות בתמונה הנ"ל כמובן שהמידע על valgrind בתור קובץ output נמצא כקובץ בתוך הגיטאהב שמשיוך לחלק 6  **שאלה מספר 7 – בנוסף יש קובץ בשאלה שנקרא file.txt   
שקשור לשאלה זאת**

בעיית מרוץ מתרחשת כאשר שני תהליכים או יותר ניגשים בו-זמנית למשתנה משותף בזיכרון, כאשר לפחות אחד מהם כותב ערך חדש. אם הפעולות האלו לא מתואמות כמו שצריך, סדר הפעולות עלול להשתנות באופן אקראי, דבר שיכול לגרום לתוצאות שגויות או לא צפויות בתוכנית.

ניתוח על פי Helgrind:

Helgrind הוא כלי בתוך Valgrind שנועד לאתר בעיות הקשורות לסנכרון בין תהליכים (כמו בעיות מרוץ). הדוח המוצג מתאר מקרה שבו תהליכים מרובים ניגשים למשתנה משותף בשם accum מבלי להשתמש במנגנון סינכרון מתאים.

פירוט הבעיות שהתגלו:

1. בעיית מרוץ אפשרית בזמן קריאה:
   * מיקום בקוד: שורה 9 בפונקציה square בקובץ race\_condition.c.
   * תהליך מעורב: תהליך מספר 4.
   * מה הבעיה: Helgrind מזהה שתהליך 4 ניסה לקרוא את המשתנה accum בזמן שתהליך אחר (מספר 3) כתב אליו באותו הזמן, מה שגורם להתנגשות.
   * קונפליקט: הקריאה הזו מתנגשת עם כתיבה שנעשתה על ידי תהליך 3 גם כן באותה שורה, מה שמוביל לבעיית מרוץ.
2. בעיית מרוץ אפשרית בזמן כתיבה:
   * מיקום בקוד: גם כן שורה 9 בפונקציה square בקובץ race\_condition.c.
   * תהליך מעורב: תהליך מספר 4.
   * מה הבעיה: Helgrind מזהה שתהליך 4 ניסה לכתוב ערך חדש ל-accum, מה שגורם להתנגשות עם כתיבה קודמת שביצע תהליך 3 לאותו משתנה.
   * קונפליקט: הכתיבה מתנגשת עם כתיבה קודמת שבוצעה על ידי תהליך 3, שניהם מנסים לגשת לאותו מיקום בזיכרון.

איך נגרמת בעיית המרוץ?

* משתנה מעורב: המשתנה הגלובלי accum הוא המשתנה הבעייתי, מכיוון שהוא משותף לכמה תהליכים בו-זמנית.
* מה גורם לבעיה: תהליכים מרובים ניגשים לאותו משתנה בו-זמנית, כל אחד מהם מנסה לקרוא ולכתוב לתוך אותו מקום בזיכרון בלי תיאום ביניהם.
* ללא סינכרון: הפעולות על המשתנה אינן מסונכרנות, מה שגורם לכך שהסדר של הפעולות (קריאה וכתיבה) משתנה בכל ריצה של התוכנית.

השפעה אפשרית:

* תוצאות לא עקביות: מכיוון שהגישה למשתנה accum נעשית בו-זמנית על ידי מספר תהליכים, התוצאות יכולות להשתנות מריצה לריצה, מה שעלול להוביל לשגיאות בתוצאה הסופית של התוכנית.
* סיכון לשגיאות קשות: אם אין מנגנון סינכרון מתאים, ייתכן שתהליכים יכתבו ערכים שונים לאותו מקום בזיכרון בו-זמנית, מה שעלול לגרום לאובדן נתונים או שגיאות בזיכרון.

איך לפתור:

כדי למנוע את בעיית המרוץ, צריך להשתמש במנגנון סינכרון כמו mutex (מנעול) או כל מנגנון אחר שמתאים לגישה מסונכרנת לזיכרון משותף:

* שימוש ב-mutex: נועל את הגישה למשתנה המשותף כך שרק תהליך אחד יוכל לגשת אליו בכל רגע נתון, והשאר יצטרכו לחכות עד שהתהליך הנוכחי יסיים את העבודה על המשתנה.
* סינכרון: יש לדאוג שכל תהליך שמנסה לגשת למשתנה accum יעשה זאת רק כאשר הוא מחזיק במנעול (mutex).

סיכום:

Helgrind זיהה בתוכנית בעיות מרוץ הקשורות למשתנה גלובלי accum. התהליכים ניגשים למשתנה זה בו-זמנית ללא סנכרון, מה שגורם להתנגשויות בזמן קריאה וכתיבה לאותו מקום בזיכרון. כדי למנוע את הבעיה, חשוב להוסיף מנגנון סינכרון כמו mutex כך שרק תהליך אחד יוכל לגשת למשתנה בכל פעם.

תבניות עיצוב חלק זה נמצא בקישור בגיטאהב שנמצא בעמוד מספר אחת   
ניתן לראות את המימוש של בתוך תקיית Design Patterns   
