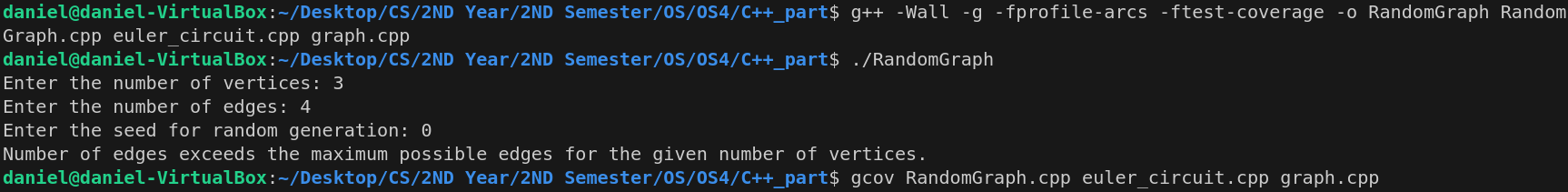
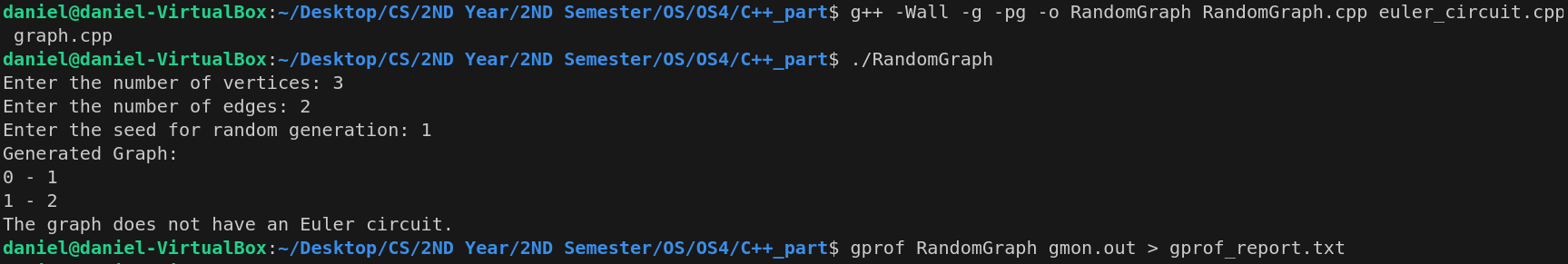
מטלה 4

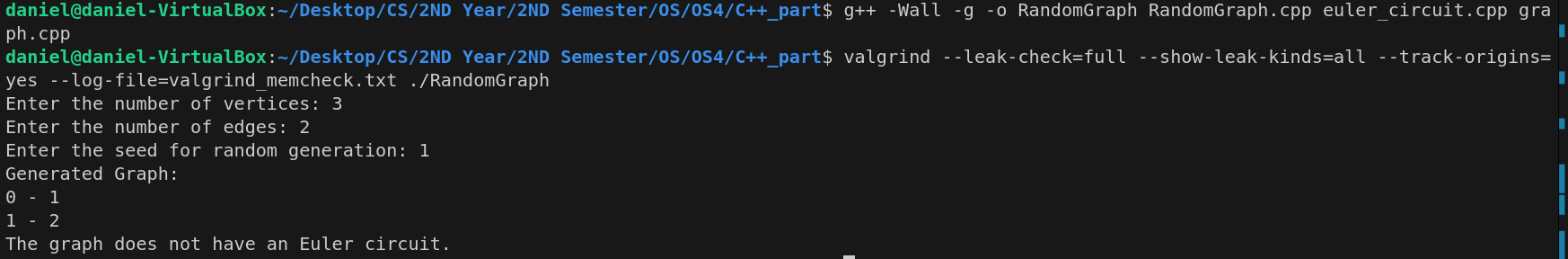
אורי קיאלי, 318375318, דניאל צדיק 209307727, הילה שמיר -314906983

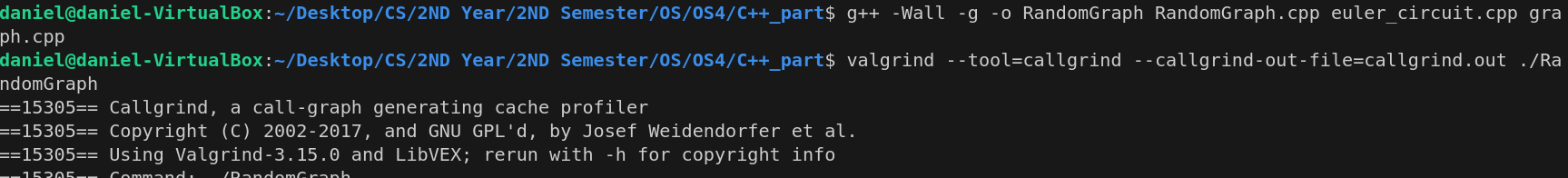
שאלות 1-4:

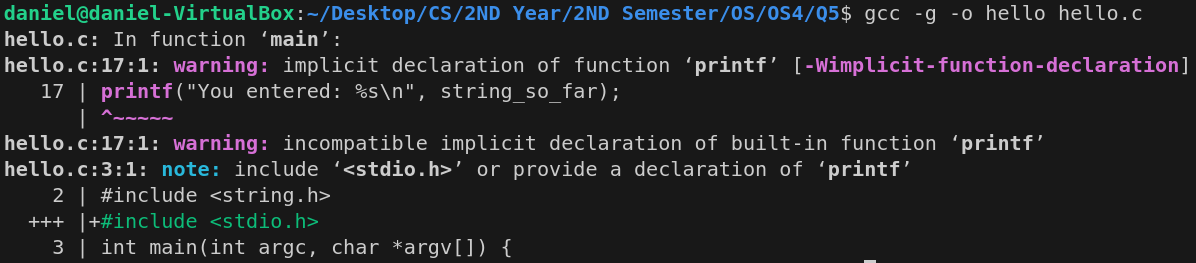
בצילומי המסך הבאים ניתן לראות איך ייצרנו את קבצי הgcov, gprof, valgrind\_memcheck וה-callgrind. כל דו"ח שמופיע בתוך הקוד כולל קלט שונה:   
עבור גרף עם 3 קודקודים ושתי צלעות, קליקה בגודל 3 וגרף עם שלושה קודקודים ועם 4 צלעות (ואז הגרף לא חוקי, אז הוא לא נבנה כלל). ניתן לראות פה למטה איזה פקודה יוצרת כל קובץ שמעניין אותנו.









שאלה 5:  
אחרי קמפול הקובץ הנתון, קיבלנו אזהרה:  


האזהרה מציינת שהקומפיילר זיהה את הפונקציה printf בקוד, אך לא מצא את ההכרזה המתאימה עבורה. זה קורה בגלל שלא נכלל קובץ הכותרת <stdio.h>, שבו נמצאת ההצהרה של הפונקציה printf.

בנוסף, לאחר הרצה של Valgrind/memcheck, קיבלנו את השגיאות הבאות:

**ערכים לא מאותחלים:**

* **מיקום בקוד:** שורה 10, שורה 11.
* **זיהוי באמצעות Valgrind** התגלו מספר מקרים של שימוש בערכים לא מאותחלים, מה שגרם להתנהגות לא מוגדרת בקפיצות תנאי ובפעולות על הזיכרון.

**גישה לא חוקית לזיכרון:**

* **מיקום בקוד:** שורה 14, שורה 11.
* **זיהוי באמצעות Valgrind** זוהו קריאות וכתיבות לא חוקיות בזיכרון עקב ניהול לא נכון של גודל ההקצאה בזיכרון.

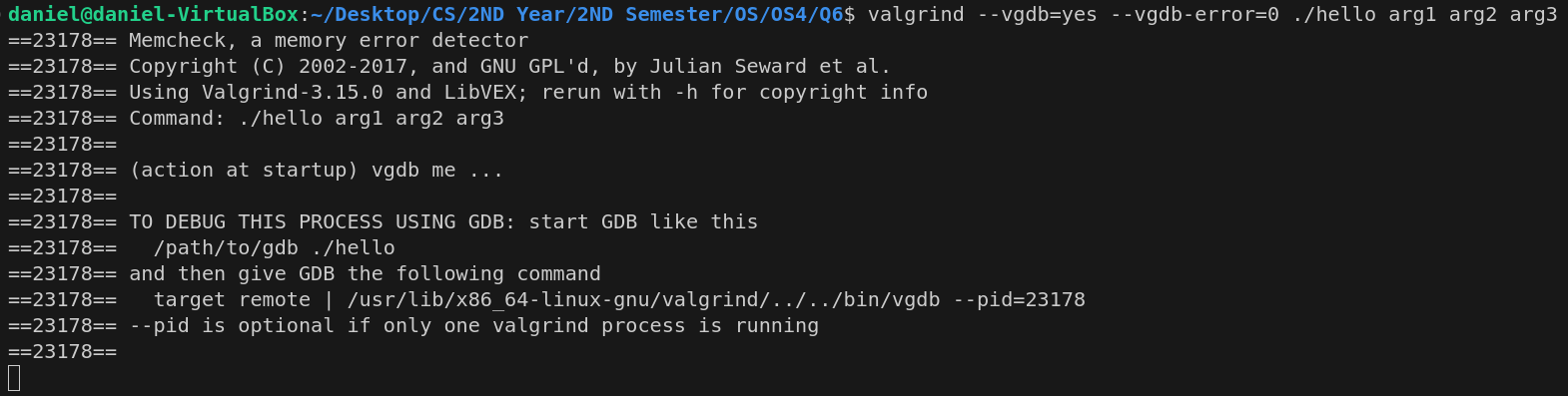
**דליפות זיכרון:**

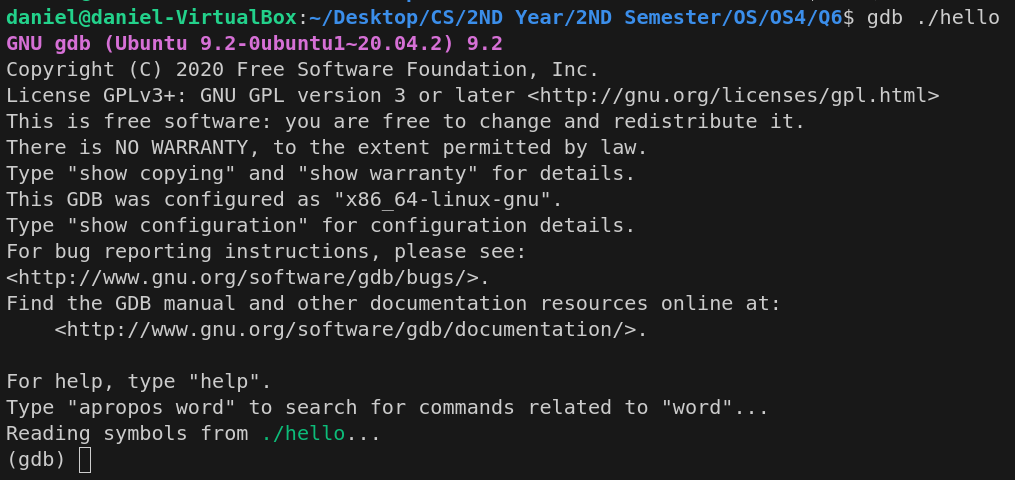
* **מיקום בקוד:** שורה 8.
* **זיהוי באמצעות Valgrind** הזיכרון שהוקצה באמצעות malloc לא שוחרר, מה שהוביל לדליפות זיכרון.

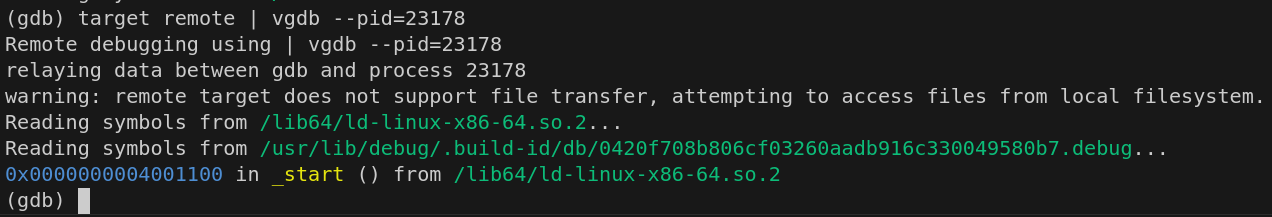
בעיות אלו מצביעות על ליקויים משמעותיים בניהול הזיכרון בקוד, שעלולים להוביל לקריסות, להתנהגות לא מוגדרת, ולשימוש לא יעיל במשאבים.

שאלה 6:

פתיחת טרמינל של VALGRIND:



פתיחת טרמינל DEBUGGER:  


שיוך בעזרת pid:  


ולאחר מכן ניתן לדבג חופשי בטרמינל דיבאגר.

כמובן שכל פעולה בתוך טרמינל הדיבאגר גורמת לחיווים בטרמינל הVALGRIND הראשון שפתחנו.

מצורף בתוך הקוד דוגמה לשני קבצי טקסט, אחד מהטרמינל של הVALGRIND והשני של הדיבאגר. כל פעולה שביצענו בדיבאגר מקבלת התייחסות בטרמינל של הVALGRIND, ואכן כשמגיעים לשורות עם דליפות זיכרון ושגיאות אחרות מקבלים על זה חיווי בטרמינל.

שאלה 7:

מצורף בקוד פירוט השגיאות שקיבלנו על ידי הרצה של Helgrind. להלן הניתוח שלו:  
פירוט הבעיות שהתגלו:

בעיית מרוץ אפשרית בזמן קריאה:

מיקום בקוד: שורה 9 בפונקציה square (race.c:9).

תהליך מעורב: תהליך מספר 3.

מה הבעיה: Helgrind זיהה בעיית מרוץ אפשרית כאשר תהליך מספר 3 ניסה לקרוא את המשתנה הגלובלי accum.

קונפליקט: הקריאה הזו מתנגשת עם כתיבה קודמת על אותו משתנה שבוצעה על ידי תהליך מספר 2, גם כן בשורה 9.

בעיית מרוץ אפשרית בזמן כתיבה:

מיקום בקוד: שורה 9 בפונקציה square (race.c:9).

תהליך מעורב: תהליך מספר 3.

מה הבעיה: Helgrind זיהה בעיית מרוץ אפשרית כאשר תהליך מספר 3 ניסה לכתוב למשתנה הגלובלי accum.

קונפליקט: הכתיבה הזו מתנגשת עם כתיבה קודמת על אותו משתנה שבוצעה על ידי תהליך מספר 2, גם כן בשורה 9.

מה גורם לבעיית המרוץ:

משתנה גלובלי מעורב: accum (משתנה שכל התהליכים משתפים בו שימוש).

תהליכים מעורבים: כמה תהליכים שמתחילים לרוץ בלולאה, וכל אחד מהם מבצע את הפונקציה square באותו זמן.

איך נגרמת בעיית המרוץ:

סיבה: בעיית המרוץ מתרחשת כי מספר תהליכים ניגשים ומשנים את המשתנה הגלובלי accum באותו זמן, בלי להשתמש במנגנון סינכרון כמו mutex.

השפעה: זה יכול לגרום לתוצאות לא עקביות או שגיאות, כי הסדר שבו התהליכים מבצעים את הפעולות על accum לא קבוע.

פירוט ההודעות מ-Helgrind:

פעולות מתנגשות:

קריאה של תהליך מספר 3 מתנגשת עם כתיבה של תהליך מספר 2.

כתיבה של תהליך מספר 3 מתנגשת עם כתיבה של תהליך מספר 2.

מיקום בזיכרון: כל הפעולות המתנגשות נעשות על אותו מיקום בזיכרון שבו נמצא המשתנה accum.

סיכום השגיאות:

סה"כ שגיאות שהתגלו: 38 שגיאות בשני מקומות שונים בקוד.

שגיאות שהודחקו: 902 שגיאות שלא דווחו על ידי Helgrind.

מסקנות:

הבדיקה עם Helgrind זיהתה בעיית מרוץ בתוכנית race.c. הבעיה נובעת מכך שמספר תהליכים ניגשים ומשנים את המשתנה הגלובלי accum בו-זמנית בלי סינכרון, מה שיכול לגרום לשגיאות בתוצאות.

המלצה: כדי למנוע את בעיית המרוץ, כדאי להשתמש במנגנון סינכרון כמו mutex כדי להבטיח שרק תהליך אחד יוכל לשנות את accum בכל רגע. כך נמנעות התנגשות בין פעולות הקריאה והכתיבה.

החלק של הסינגלטון והMUTEX מופיע בקוד עצמו.