**מערכות הפעלה מטלה 3:**

2 מגישים:

איתמר בבאי 206847584

שריאל משה 322772880

**שאלה 1:**

|  |
| --- |
| * התבקשנו לממש קמור, נחזיר struct עם וקטור שיכיל נקודות, גודל ושטח. * נקבל מהטרמינל את מספר הנקודות ואת הנקודות עצמן על פי הפורמט x,y * נעזר באלגוריתם של אנדרו לחישוב הקמור מהפוליגון שהתקבל. * תחילה נמיין על פי x ואז על פי y , לאחר מכן נבין את כיוון הפנייה של השלשה. נכניס את החלק העליון ולאחריו את התחתון, לבסוף נסיר את הנק הכפולות. * לאחר סינון הקמור מהפוליגון נחשב את השטח של מה שנותר ונחזיר אותו. |
|  |
| דוגמת הרצה בטרמינל: |
|  |

**שאלה 2:**

|  |
| --- |
| * התבקשנו לבצע profiling בין 2 אלגוריתמים או מבני נתונים , בחרנו להשוות בין 2 אלגוריתמים. * פונקציה אחת מוצאת את הקמור עם האלגוריתם של אנדרו והשנייה בעזרת גרהם. * נבצע gprof לאחר הרצה של 100,000 נקודות, את המידע נשמור בקובץ analsys.txt . * בנוסף נעזר בchrono כדי לזהות את זמן הריצה של כל אלגוריתם ונדפיס אותו. |
|  |
|  |
|  |
|  |
|  |

**חלק 3:**

|  |
| --- |
| * בחלק זה נאפשר קבלת פקודות מstdin ונבצע אותם לפי האלגוריתם שמימשנו. * נטפל בשגיאות קלט (חוסר בפסיק) ושגיאות לוגיות כמו הסרה של קודקוד לא קיים. |
|  |
|  |
|  |

**שאלה 4**:

* בחלק זה התבקשנו לאפשר קבלת תקשורת מלקוח ולתחזק גרף משותף לכל הלקוחות (ללא mutex).
* בתוכנית שיצרנו השרת מאזין מעל פורט 9034 כמתבקש וממתין להתחברות TCP , במקביל השרת מאזין גם לstdin ובעזרת select מזהה האם ישנה בקשה מהלקוח (או לקוחות) או שהיא הגיע מstdin.
* הגרף מאוחסן אצל השרת ומשותף לכל הלקוחות המחוברים, כל אחד מהם יכול להקיף גרף חדש, להוסיף קודקוד למחוק ולמצוא קמור.

|  |  |  |  |
| --- | --- | --- | --- |
| * בדוגמה הקרובה נשים לב לגרף שנבנה אצל השרת ונוספים לו עוד קודקודים אצל הלקוחות * לאחר מכן נמחק קודקוד שנוסף מלקוח 1 ע"י לקוח 2 * לבסוף נבצע CH מ2 הלקוחות והשרת ותראה שזהים. | | | |
| * יצירת הגרף בשרת והתחברות 2 לקוחות | | | |
|  | | | |
| * הקמת לקוח 1 והוספת קודוקוד לגרף שנוצר בשרת | | | |
|  | | | |
| * הקמת לקוח 2 והוספת קודקוד לאותו גרף בדיוק | | | |
|  | | | |
| לקוח 1 | | לקוח 2 | |
|  | |  | |
| לקוח 1 | לקוח 2 | | שרת |
|  |  | |  |

**שאלה 5** :

בשאלה זו התבקשנו לייצר ספרייה של תבנית עיצוב בשם reactor, יצרנו אותו בcpp באופן כזה שאחרי שניצור אובייקט מסוג reactor נוכל בעזרתו להפעיל את כל הפונקציות הנדרשות (את ההפעלה שלהן נציג בשאלה הבאה).

כעת נציג ונסביר על מספר חלקים בספרייה

|  |
| --- |
|  |
| בתמונה לעיל ניתן לראות את קובץ הhpp של מחלקת Reactor ואת המתודות אותן ניתן להפעיל, אין צורך בשליחת פוינטר לreactor מכיוון שזה האובייקט עצמו.  כמו כן הגדרנו שדה mutex שיגן על הmap של fd |
|  |
| מימוש של בנאי – הפעלה של משתנה בוליאני שמסמן שהreactor רץ מימוש של מפרק – קורא ל stop Reactor  מימוש של stopReactor – נועל את העבוד מ threads אחרים ,מכבה את הreactor מנקה את map של fd. |
|  |
| בפונקציה לעיל נפעיל את ה reactor ונתחיל לבחון fd "חמים" תחילה נשים ערך true ב running  ניכנס ללולאה אינסופית שתחילה תמלא את הקבוצה של read\_fds בfd שצריך לבדוק אם קיבלנו דרכם מידע.  לאחר מכן נבחן מהו ה fd המקסימלי לטובת בדיקה של select  נמתין בselect עד שימצא fd שיש בו מידע אם הfd "חם" נפעיל את הפונקציה שנמצאת באיבר השני בpair (שחולץ מה map) .  נבצע נעילה של mutex לפני כל פעולה שביצענו בה שינוי ב fdmap. |

**שאלה 6** :

מטרתנו היא להשתמש בreactor כדי לאפשר לשרת לנהל מספר לקוחות במקביל בצורה של תבנית עיצוב ללא פתיחת tread חדש לכל לקוח.

התהליך יחסית דומה לשאלה 4 למעט השימוש באובייקט reactor.

נחלק את הטיפול בכל סוג לקוח לפונקציה נפרדת על מנת שנוכל לשלוח את המצביע של אותה פונקציה ל reactor , כאשר ל stdin יש פונקציה משלו ולכל לקוח אחד יש פונקציה אחרת.

נשים לב שגם הfd של הsocket המאזין הוא fd שצריך לבדוק ולכן נבנה גם עבור השרת פונקציה ונכניס גם את הfd שלו אל reactor לטובת קבלת לקוחות חדשים.

ברגע שלקוח סוגר תקשורת עם השרת נכניס אותו לרשימת ההסרה של fd וכך נדע להפסיק לבדוק אותו בselect .

|  |
| --- |
| * צירוף הספרייה בmakefile |
|  |
| * הוספת הfd של השרת ושל stdin אל הreactor ולאחר מכן נפעיל את הלולאה של הreactor (startReactor) שהצגנו בשאלה 5. |
|  |
| * בעת התחברות לקוח חדש הfd של הserver הופך ל"חם" ומפעיל את הפונקציה של הserver. * בפונקציה זו נבצע accept עם לקוח חדש ונוסיף אותו לרשימה של ה reactor כך שה select יבחן גם אותו בלולאה הרשאית. |
|  |
| * כאשר לקוח שולח פקודה ה fd שלו הופך ל"חם" ומפעילים את הפונקציה שמשויכת אליו ב reactor . |
|  |
| * אותו דבר לגבי stdin |
|  |
| * נציג תהליך של יצירת גרף ע"י stdin * הוספת נקודות ע"י לקוח 1 * הוספת נקודה ע"י לקוח 2 * הסרת הנקודה שלקוח 1 הוסיף ע"י לקוח 2 * CH ע"י כולם. |
| * טרמינל של השרת |
|  |
|  |
| * לקוח 1 |
|  |
| * לקוח 2 |
|  |

**שאלה 7** :

* נממש את שאלה 4 בעזרת Threads באופן הבא,
* נפתח thread עבור stdin
* ניכנס ללולאה הראשית שלנו בה נפתח thread נפרד לכל לקוח חדש שיכנס, נמתין בrecv כדי לקבל מידע מהלקוח ומשם נפעיל את פונקציה ה handle\_request
* לפני כל הפעלה של פונקציה הhandle נפעיל נעילה של mutex .

|  |
| --- |
| * יצירת ה thread עבור stdin |
|  |
| * יצירת thread עבור על לקוח אחר |
|  |

**שאלה 8**:

* התבקשנו לעדכן את הספרייה על מנת שתוכל לעבוד גם בצורה של proactor , ניצור מחלקה חדשה בשם proactor שדרכה נממש את תבנית העיצוב.

|  |
| --- |
| * נשתמש במבנה הבא שמכיל fd של socket פוינטר לפונקציה ומשתנה בוליאני שיצביע אם הproactor עדיין רץ. * נשתמש במבנה הזה כשפה פרטי של Proactor |
|  |
|  |
| * נייצר גם פונקציית startProactor שמטרתה היא להריץ את הלולאה העיקרית עם Thread חדש. * פונקציה זו מקבלת fd של הלקוח החדש ופוינטר לפונקציה. * מגדירה את השדות של האובייקט * מגדירה thread חדש ושולחת אליו כפרמטרים את מספר הזיהוי שלו, את הפונקציה שהוא צריך להריץ (הלולאה הראשית ) ואת הארגומנטים שמכילים את המידע על fd. |
|  |
| * ניצור פונקציה שתהווה הלולאה הראשית , אותה כל thread יריץ וממנה יטופל הקלט של הלקוח. * תחילה נקבל את הארגומנטים ונחלץ אותם לחלקים. * נשחרר את הזיכרון עבור המצביע שהוקצה בstartProactor * נרוץ בלולאה וברגע שנקבל connect מהלקוח נפעיל את הפונקציה שקשורה אל אותו fd (במקרה שלנו תהיה פונקציה שמטפלת בקלט מהלקוח) בthread חדש. |
|  |
| * בנינו פונקציה נוספת שתעצור את הproactor * הפונקציה יוצרת sockets לכל thread במטרה להפעיל את הaccept (עם connect) שלו ואז לבצע לו join. * מכיוון שהלולאה הראשית של proactor חסימתית אנחנו חייבים לעצור אותה באופן זה. |
|  |

**שאלה 9** :

* בשאלה זו התבקשנו לבצע את שאלה 7 (מימוש תקשורת שרת לקוח וstdin) בעזרת threads רק הפעם בשימוש proactor.
* לאחר יצירת socket נפעיל את הפונקציה startPreactor עם הפרמטרים של הפוינטר לפונקציה שמטפלת בלקוח, וה fd שלו, משם נוצר thread חדש עבור אותו לקוח בפונקציה הלולאה של proactor , אותו thread יעבוד על הפונקציה שמטפלת בקלט מלקוח כך שהחל מרגע זה כאשר הלקוח ישלח מידע מסוים בsocket המידע יתקבל בפונקציה on\_client\_socket ויפעיל את handle .

|  |
| --- |
|  |
| * כמו כן ניצור thread חדש עבור stdin עם פונקציה הטיפול הישירה שלו. * כאשר מתקבל מ stdin exit אז runningServer הופך ל 0 וכך משתחררת הלולאה של main . * נפעיל על ה thread של הserver וכך נפעיל connect כדי לשחרר מהלולאה את הserver. * לבסוף נבצע join כדי לאחד בין הthreads השונים למניעת זומבים. |

**שאלה 10:**

* בשאלה זו התבקשנו להוסיף בדיקה על האם שטח הקמור עבר את סף ה100 מהפעם האחרונה שביקשנו חישוב קמור.
* נעזרנו בcondition variable , ניצור עבורו thread משלו ונעיר אותו בכל חישוב CH . כמו כן עבור הפעלת משתנה זה נשתמש בmutex נפרד.
* נשמור משתנה בוליאני שיאמר האם בקריאה האחרונה ל CH האחרון היינו מעל או מתחת ל100 ונדפיס את ההודעה המתאימה.

|  |
| --- |
| שרת |
|  |
| לקוח ( ההדפסה על השינוי היא בשרת בלבד) |
|  |
| שרת (הרבה הדפסות אך העיקר הדפסת המעבר לשטח מעל 100 בקריאה לCH) |
|  |
| שרת – הסרת נקודה וחישוב CH לאחר ירידה מ 100 (יש לשים לב שהפעלה נוספת של CH לא מביאה הדפסה על שינוי בשטח) |
|  |