שאלה 1

1.א

נוסיף Signal Handler אשר יתפוס את הסינגל SIGALARM בכל שניה. נחזיק מונה שיספור עד 7, כאשר הגיע ל7 נשלח את התהליך הנוכחי לישון. הזמן של המערכת FCFS יוציא אותו מפעולה ויביא את התהליך הבא שמובטח שירוץ 7 שניות, התהליך שהלך לישון באותו הרגע יחזור לסוף התור וכך נממש את הזמן RR.

הSignal Handler

void SigAlarmHndl(int signum){  
 if(++count == 7){  
 count = 0;  
 sleep(1);  
 }  
}

כאשר Count משתנה גלובלי שמאותחל ל0.

כמו כן בפונקציית המיין נוסיף את השורה:

signal(SIGALARM,SigAlarmHndl);

בתחילת התוכנית.

1.ב

נוסיף משתנה גלובלי, אשר ישמור את מספר החוט הנוכחי והחוט הבא אשר צריך לעבוד. לפי הנתון כל שניה נקבל סיגנל, לכן נתפוס וננהל את החוטים על ידי signal handler אשר תופס את SIGALARM. הsignal handler:

int current\_thread = 0;  
  
  
void SigAlarmHndl(int signum){  
 int reg = current\_thread;  
 if(++current\_thread == 3) current\_thread = 0;  
 if(setJmp(b[reg] == 0)){  
 longJmp(b[current\_thread],1);  
 }  
}

כל פעם נכניס למשתנה לוקאלי reg את החוט אשר אנו נמצאים בו עכשיו (ואם חוזרים אליו אז הערך שsetJump יחזיר יהיה 0. נקדם את המונה של החוט ונשמור אותו בין 0 ל2. לאחר מכן נבדוק אם זו פעם ראשונה שהחוט רץ setJmp יחזיר 0 ונשלח לlongJmp את החוט המקודם.אחרי סיבוב כאשר נשלח לlongJmp את החחוט הראשון שוב, הערך המוחזר לא יהיה 0 הוא יהיה אחד והחוט הראשון ירוץ עוד שניה עד הsignal הבא. מכיוון שאנו רוצים שתהליך זה יתפוס את הסיגנל נפריד להתליכים שונים, ובפקונציית הmain נוסיף שורה בה אנו מכניסים את הhandler שלנו ונוסיף fork כדי שהsignal יתפס על ידי החוטים בלבד.

שאלה 2.

2.א –

לא נכון- חוטים משתפים את מרחב הזיכרון ביניהם. לשניהם גישה לאותה המחסנית לכן כאשר ניגש לאותו משתנה משני חוטים שונים נצטרך להגן עליו באמצעות נעילה.

2.ב –

ייתכן שיהיה deadlock, אם כל חוט אחרי השורה הראשונה יעשה context switch נקבל ששלושת המנעולים נעולים וכל התהליכים ילכו לישון אחרי השורה הבאה.

שינוי לקוד אשר יתקן:

החוטים צריכים לנעול באותו הסדר את המנעולים ( כאשר זה 3 חוטים ניתן להגדיר שינעלו בסדר עולה). לדוגמה:

Thread 1:  
1. LOCK(L1);  
2. LOCK(L2);  
3. // critical section  
4. UNLOCK(L2);  
5. UNLOCK(L1);  
  
Thread 2:  
1. LOCK(L1);  
2. LOCK(L3);  
3. // critical section  
4. UNLOCK(L3);  
5. UNLOCK(L1);  
  
Thread 3:  
1. LOCK(L2);  
2. LOCK(L3);  
3. // critical section  
4. UNLOCK(L3);  
5. UNLOCK(L2);

3.

sem\_t vodka\_s = {};

sem\_t orange \_s = {};

sem\_t umbrela\_s = {};

void init(){

    sem\_init(&vodka\_s, 0, 0);

    sem\_init(&orange\_s, 0, 0);

    sem\_init(&umbrela\_s, 0, 0);

}

void put\_vodka(){

    g.add(vodka);

    up(vodka\_s);

    down(orange \_s);

    down(umbrela\_s);

    serve\_if\_ready();

    up(umbrela\_s);

    up(orange \_s);

}

void put\_oragne(){

    down(vodka\_s);

    g.add(orange);

    up(orange \_s);

    down(umbrela\_s);

    down(orange \_s);

    serve\_if\_ready();

    up(orange \_s);

    up(umbrela\_s);

}

void put\_umbrella(){

    g.add(umbrela);

    up(umbrela\_s);

    down(orange \_s);

    down(umbrela\_s);

    serve\_if\_ready();

    up(umbrela\_s);

    up(orange \_s);

}

4

.א. יכול לקרות DEADLOCK, בצורה הבאה:

בקריאה ל recursiveFunction, בהמנעול LL היה פתוח אז נעלנו אותו בכניסה לפונקציה

Condition ==True

נכנסנו לקריאה הרקורסיבית

שוב אנחנו מנסים לתפוס את אותו המנעול כאשר הפעם הוא נעול, אבל הוא לא יפתח עד שהפונקציה החיצונית לא תגמר, ולכן נשאר תקועים.

4.ב לא ברור האם ניתן להשתמש ב pthread\_self

5. א

הבעיה יכולה לקרות בתסריט הבא:

עבור N=10, נניח וכבר 8 חוטים ממתינים על שורה 22 כאשר threads\_inside==8, ואז מגיע חוט מספר 9 שמגיע לשורה 20 ואחרי ביצוע הup(s1) מתבצעת החלפת הקשר לחוט מספר 10, threads\_inside עדיין לא עודכן ל 9, ולכן במקום להעיר את כולם, חוט 10 ילך גם לחכות על שורה 22, ונגיע למצב שאנחנו עדיין מחכים, אפילו שכבר נכנסו 10 חוטים לBarrier

5.ב לא מצאתי בעיה..

ג. פה יש בעיה, יכול להיווצר dead-lock, עבור N=10, כאשר התהליך הראשון קורא ל EnterBarrier, ונועל את s1 בשורה הראשונה, הוא יכנס להמתנה על שורה 21 על s2 אבל אז כל תהליך נוסף שיקרא לEnterBarrier יתקע על שורה 12 כי s1 עדיין נעול והוא ישאר נעול כי התהליך הראשון שאמור לשחרר אותו תקוע עך שורה 21