1. א. נכון: G הוא בהכרח DAG.

הוכחה: יהי הגרף G = (P, E) , נניח בשלילה שהגרף לא DAG (גרף מכוון ללא מעגלים), הגרף שלנו מכוון ולכן בהכרח G הוא גרף שבו יש לפחות מעגל אחד. נסתכל על המעגל הקצר ביותר בגרף (מעגל זה מורכב לפחות משני תהליכים שונים, אחרת המעגל היינו חץ עצמי: אבל אף תהליך אינו יוצר את עצמו ואפילו לא init או idle. כי ההגדרה לאבא: הוא מי שעשה עליי fork – ואף תהליך לא יכול מבחינה לוגית או פיזית לעשות fork לעצמו- כי כדי לעשות fork התהליך צריך להיות קיים בעצמו. בכל מקרה הנקודה שאנחנו מנסים להעביר שאין חצים עצמיים ולכן המעגל הקטן ביותר מכיל לפחות 2 תהליכים שונים). במעגל זה ישנם לפחות 2 תהליכים שונים, ונסמן בp_0 את התהליך שנוצר הכי מוקדם מבין התהליכים במעגל זה. מכיוון שזה מעגל, ישנו תהליך אחר, נסמנוP_1 שהוא מצביע לp_0 במעגל זה (כל אחד מהתהליכים מופיע עד פעם אחת במעגל זה, אחרת היינו יכולים לקצר אותו עוד).

אבל מכיוון של p_1 יש חץ אל p_0, אזי בהכרח הוא האב האמיתי שלו- כלומר מתקיים כי p_1 עשה fork ל p_0 ולכן בהכרח p_1 נוצר לפני p_0 בסתירה לכך שp_0 הוא התהליך שנוצר ראשון מבין תהליכי מעגל זה. ולכן הגרף בהכרח DAG.

ב. הטענה שגויה.

על מנת שG תהיה עץ עליה להיות גרף קשיר חסר מעגלים (עבור מעגלים לא מכוונים). מספיק שנראה כי דוגמא נגדית לגרף G שבו יש יותר מרכיב קשירות אחד.

נבנה גרף כזה: עבור 'G גרף תהליכים כזה, שנגיד שהוא עץ נוכל להראות שעל ידי מספר פעולות פשוטות נוכל להפוך את הגרף הזה ללא קשיר. יהי P אחד מהתהליכים הפעילים כרגע, נעשה forl ממנו- והוא ייצור בן חדש Q- נשים קשת בין P לQ. כעת P יעשה fork נוסף ויצור בן חדש W – ולכן נוסיף את הpid W לגרף ונוסיף חץ מP האבא אל הבן שלו W. כעת נהרוג את התליך האבא P באמצעות exit , ונקבל כי ישנם שני תהליכים Q,W בגרף (נסתכל עליו כאל גרף לא מכוון)- שאין ביניהם שום מסלול מאחד לשני. ולכן הגרף החדש הוא גרף חוקי (קודקודיו הם התליכים הפעילים כרגע: שאלו כל התיכים שהיו פעילים ב'G, ללא התהליך P ועם שני תהליכים חדשים W,Q – ולכל תהליך עדיין יש חץ לאבא האמיתי שלו אם אבא שלו עדיין בגרף). אזי הגרף החדש הוא גרף תהיכים חוקי, אבל לא קשיר (יש לפחות 2 רכיבי קשירות) ולכן גרף זה אינו עץ. נסמן את הגרף הזה בתור G.

ג. הטענה נכונה

על מנת להוכיח זאת, עבור הגרף G נסתכל על גרף חדש K : שהוא יהיה הגרף של כל התהליכים שהיו אי פעם עד כה במערכת. (עבור שני תהיכים עם אותו pid כי אחד כבר מת והשני קיים כעת נמספר אותם לפי מי היה הראשון וכן הלאה). והקשתות בין התהליכים יהיו כמו בגרף G המקוקי: רק אם תהליך 1 הוא (או היה) האבא האמיתי של תהליך 2 יהיה לו חץ אליו. נוכיח באינדוקציה שגרף זה הוא עץ – על k מספר התהליכים בגרף K:

עבור k=1 נקבל שיש רק תהליך אחד- שהוא init והוא עץ כי זה רק צומת אחד.

עבור k כלשהו נניח, ונוכיח עבור k+1 את נכונות הטענה.

יהי הגרף K עם k+1 תהליכים בו- נסתכל על D התהליך שנוסף אחרון אל הגרף. אזי בהכרח אין לו בנים- כי אחרת לא היה האחרון שנוסף. ובמילים אחרות: יש לו רק שכן אחד בגרף והוא אבא שלו. נוריד את התהליך D ונותרנו עם גרף 'K – בלי התהליך D ובלי החץ שלו לאביו. גרף זה מתאר את המצב לפני הוספת התילך D אל התמערכת- ולכן גם גרף זההוא גרף תהליכים- לפי הנחת האינדוצקיה מכיוון שיש לו רק k תהליכים- הוא עץ. כעת נוסיף את D אל הגרף במקום שבו היה מחובר קודם לכן- מתקיים בהכרח כי הגרף K המקוקי, הוא קשיר: כי כל kהתהליכים הראשונים הם מקושרים ביניהם- והתהליך D מחובר אל אביו ולכן הוא גם חלק מרכיב הקשירות הכולל. ובגרף זה אין מעגלים: בהוספת D הוספנו קשת אחת בלבד- ולכן אם יש מעגל היה חייב להיות גם טרם הוספת D בסתירה

לכך שגם בלי D הגרף הוא עץ! ולכן גרף K הוא עץ. נשים לב כי הגרף G מוכל בגרף K – עם פחות תהליכים ופחות קשתות ולכן כפי שבגרפK אין מעגלים, גם בגרף G אין- כלומר הוא יער כנדרש.

ד. הטענה נכונה:

BFS מבטיח לצבוע ("לבקר") בכל צומת שלקודקוד 0 יש מסלול אליה בגרף. אבל כפי שהראינו בסעיף קודם גרף הוא יער ולא בהכרח עץ ולכן ייתכנו צמתים שנמצאים ברכיב קשירות אחר מזה של הקודקוד 0 (ושם זה היה עבור גרף לא קשיר- ולכן בהכרח גם עבור G הגרף הקשיר הם בהכרח לא נמצאים באותו רכיב קשירות- כי עבור הגררחסר הכיוונים הם לא נמצאים באותו רכיב קשירות).

ולכן גם אם נפעיל BFS מהקודקוד 0, יתכנו תהליכים שאינם עוד פעילים במערכת- ולכן כבר לא בגרף וכך הם מבטלים את הקשירות של הגרף G. ולכן הפעלת BFS מהקודקוד 0 אינה תוכל לצבוע/לבקר גם התיכים שנמצאים ברכיבי קשירות שונים מרכיב הקשירות של 0. ובכך יתכנו קודקודים שלא נבקר בהם בBFS מהקודקוד 0.

שאלה 2:

א. הטענה שגויה.

הסבר: שני התהליכים קיימים במערכת בו זמנית, ולכן לא ייתכן קיבלו את אותו הpid, מכיוון שהpid הוא המספר הייחודי של תהליך- והוא ילווה את התהליך מהרגע שייווצר עד הרגע שייאסף מwait או עד הרגע שinit ישחרר אותו (כלומר עד שהתהליך משוחרר סופית הPID שלו ייחודי רק לו מבין כל התהליכים שנמצאים גם במערכת בזמן זה). בשל היחודיות של הpid לכל תהליך פעיל במערכת- לא ייתכן ששני התהליכים שוייכו לאותו המספר.

הייחודיות של הpid מאפשרת למערכת ההפעלה להבדיל בין התהליכים, ולזהות אותם. ברשימת התהליכים, כל תהליך נשמר לפי הpid שלו- ולכן לא ייתכן שיש שני תהליכים עם אותו הpid – אחרת לא היינו יכולים להבדיל ביניהם. והרעיון: של מספר ייחודי לא היה ממומש כאן.

ב. הטענה נכונה.

הסבר: אותו ההסבר כמו בשאלה הקודמת- pid הוא יחודי ולכן לא ייתן שהם חולקים את אותו המספר (***** בהינתן ששני התהליכים קיימים באותו הזמן במערכת).

***הערה: אם ייתכן ששני התהליכים לא קיימים באותו זמן: ואחד בדיוק נגמר רגע לפני שהשני נוצר ואז הוא מקבל את המספר pid שלו- אז התשובה לשאלה זו צריכה להיות: "הטענה שגויה". אבל שני התהליכים קיימים בו זמנית ולכן חייבים להיות להם pid שונים.

****הערה חשובה נוספת: אנו יוצאים מתוך הנחה ש p1 וp2 הם שני תהליכים שונים שקיימים בו זמנית! אם מדובר באותו תהליך אז בוודאי יש להם את אותו pid- כי זה אותו תהליך. אם הכוונה בשאלה ששני התהליכים יכולים בעצם להיות אותו תהליך אז התשובה לשאלה זו היא: הטענה שגויה.

לגבי סעיפים ג וד':

כפי שלמדנו, תהליך P מופיע בזיכרון בצורה הבאה: text אחריו data (שם שמורים המשתנים הגלובאליים של תהליך) ואז הheap והstack שמתקדמים אחד לקראת השני. ולכן חשוב להדגיש כי גם אם שני תהליכים הגדירו

משתנה גלובאלי X, לא ייתכן שזה באמת יהיה אותו משתנה גלובאלי- כי כל אחד מהXים ישמר במקום אחר בזיכרון: בdata של התהליך שבו הוא נמצא. ולכל תהליך את אזור זיכרון שהוא רק שלו ואינו משותף בינו לבין שאר התהליכים (ללא קשר לזיכרון משותף בין תהליכים- זה דיון אחר).

ולכן נקודה חשובה: **לשני התהליכים הללו יש X שונה משלהם- ולכל X כזה כתובת פיזית אחרת-** בהתאם למיקום הDATA של התהליך בתוך הזיכרון הפיזי.

אבל נקודה חשובה לציין כאן היא שx& מתוך תהליך, אינה בהכרח נותנת את הכתובת הפיזית של המשתנה הגלובאלי X בתוך הזיכרון הפיזי.

ולכן התשובה לד' היא: טענה שגויה.

הפלט של שני התהליכים אינו בהכרח שונה: כל אחד מהם מדפיס את x& של המשתנה הגלובאלי X שלו. מערכות הפעלה רבות (בניהן גם לינוקס) תומכות ברעיון הווירטואליזציה- אחד הרעיונות הבסיסיים ביותר בניהול זיכרון: לפי רעיון זה כל תהליך מקבל אשליה שכל הזיכרון שלו על ידי סט של כתובות וירטואליות שהוא מקבל- ובכך הוא חושב שאלו כל הכתובות בזיכרון הפיזי: min עד MAX הכתובות, אבל בפועל זוהי רק אשליה כי הכתובות הווירטואליות שלו ממורות לכתובות פיזיות שמגדירות את המקום בזיכרון שמוקצה עבור תהליך זה בזמן הקיום שלו. ולכן על אף שלXים של שני התהליכים לא תיתכן אותה כתובת פיזית- x& אינו נותן את הכתובות הפיזית של X אלא את הכתובת הווירטואלית שלו, כפי שהיא נראית מהעניים של התהליך שבו X מוגדרת. ואין שום מניעה, ואף יש סיכוי (לפי הקוד של התוכנית) שבה שני התהליכים יתנו את אותה הכתובת הווירטואלית עבור הX ים שלהם, ועבור כל תהליך היא תומר לשתי הכתובות הפיזיות השונות. ולכן ייתכן מצב שהפלט (שהא x& הכתובת הווירטואלית של x בכל תהליך) תהיה זהה.

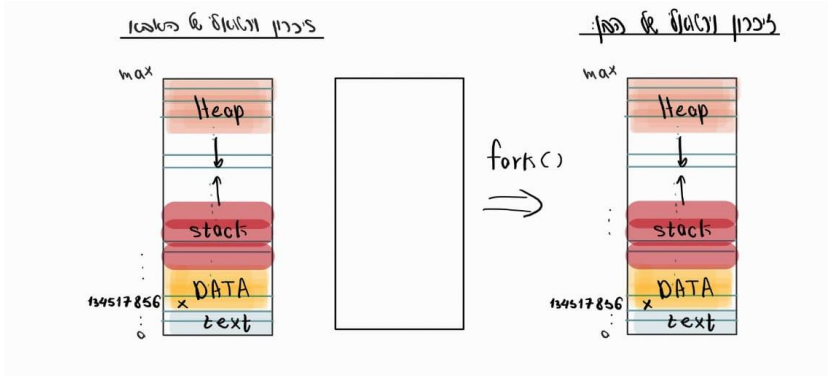
ואפילו על מנת להמחיש זאת ניתן דוגמא נגדית:

אם ישנו תהליך אבא בעל משתנה גלובאלי x, והאבא עושה ()fork מתקיים כי כל הזיכרון של הילד מועתק בדיוק מהזיכרון של האבא. ולאבא יש בחלק בזיכרון שמוקצה לו בDATA שמור המשתנה הגלובאלי x. ולכן אותו העתק בדיוק של המשתנה הגלובאלי הזה של האבא, יעבור גם אל הבן. במערכת הפעלה כמו לינוקס, שתומכת בזיכרון וירטואלי- ומכיוון שהתוכנית של האבא והבן הן זהות – מתקיים כי כל הזיכרון של האבא מועתק כפי שהוא אל הזיכרון של הבן ומוכנס בסדר כזה או אחר ששומר על אותה החוקיות: textואז data ואז מחסנית וheap שמתקדמות אחת לקראת השנייה. במצב כמו בדוגמא המצורפת (סביר מאוד שבחלק מהמקרים אם לא בכולם) הזיכרון של האבא יועתק אל הילד בדיוק כפי שהופיע אצל האבא: ביט אחרי ביט. אבל מכיוון שיש תמיכה במערכת וירטואלית של זיכרון: כל אחד מהתהליכים רואה את המקום שלו בזיכרון, כמו כל הזיכרון: וחושב שהוא יושב על כל הכתובות: 0 ועד MAX. אבל מכיוון שהזיכרון של האבא הועתק בדיוק ביט אחרי ביט אל הזיכרון של הבן: הכתובת הווירטואלית של הDATA אצל האבא והבן הן יכולות להיות זהות, בשל התמיכה בזיכרון הווירטואלי (אלא אם כן יש איזשהו שינוי מכוון כמו הASLR שהזכרנו בהמשך) ולכן יוצא מצב שהכתובת הווירטואלית של שני העתקי X אצל האבא והבן יושבות באותה כתובת וירטואלית והפלט יהיה זהה.

```
int x = 0;

int main()
{
    if (fork() > 0)
        wait(NULL);
    printf("%d\n", &x);
    exit(0)
}
```

```
134517856
134517856
```



לגבי ג': הטענה גם היא שגויה

הסבר: יתכנו מספר רב של סיטואציות בהן הפלט יהיה שונה ולא זהה:

1. בהינתן שמערכת ההפעלה תומכת בוירטואליזציה (שזה מאוד הגיוני, כי זו תכונה בסיסית), ייתכן מצב שבו הכתובות הוירטואליות של שני הXים יהיו שונים. וזה תלוי בהמון פרמטרים שאת חלקם לא למדנו בקורס (לדוגמא ASLR אשר אחראיות על מיקום הספרייה במקום "רנדומלי" בזיכרון).
2. כמו כן ייתכן מצב שבו הקוד של שני התהליכים אינו זהה. אם אחד משני התהליכים יש יותר ממשתנה גלובאלי אחד (לדוגמא בנוסף לx=0 הגדיר גם y= 3), אז המיקום של x בזיכרון ממש לא מובטח להיות אותו המיקום הוירטואלי שאותו מעניק התהליך השני- כי אין שום סיבה לכך שהם כן יהיו מוגבלים לשבת באותו מיקום בזיכרון הוירטואלי (זה שלשניהם קוראים x, או שהם מאותחלים לאותו לערך כלשהו לא אומר שהם ישבו במקום ספציפי בDATA, במיוחד אם יש עוד משתנים גלובאליים שמוגדרים).
3. הסבר נוסף (אבל פחות הגיוני) אם יש מערכת הפעלה שאינה תומכת בזיכרון וירטואלי (למרות שזו תכונה די בסיסית של מערכות הפעלה). במצב כזה x& אינו מוגדר להיות כתובת וירטואלית כי אין כזו- אלא הוא בהכרח הכתובת הפיזית של המשתנה x בזיכרון. מכיוון שאלו שני משתנים גלובאליים שונים, הם יושבים בכתובות שונות בזיכרון ולכן בהכרח הפלט עבור x& גם הוא יהיה שונה.