23 ליולי

יום ראשון 23 יולי 2017 13:32

ספולינג- אם יש לי מצב שאני פונה לרכיב איטי, אני לא רוצה לתת ל-CPU להתעקב עליו. הרעיון הוא לפנות לרכיב מהיר יותר ואז לאגור את כל מה שהוא רוצה
למשל מדפסת- זה איטי. אז בהתחלה נכתוב לזכרון ורק אחכ לא על חשבון משהו אחר, נזרוק את זה למדפסת

Processes

פרוסס זה פשוט יחידת ריצה שמקבל את כל המשאבים והזכרון של המחשב בזמן שהיא רצה. תהליך כשהוא רץ- כל משאבי המחשב נתונים לו. משאבים הכוונה היא רגיסטרים, מרחב זכרון וכו'.
המשמעשמות של כל המשאבים נתונים להליך, זה אומר שבזמן שאני רץ אף אחד לא יכול להפריע לי.
זה אומר גם שמערכת ההפעלה צריכה להגן על התהלים, כך שבזמן שהוא רץ אף אחד לא יכול להפריע.

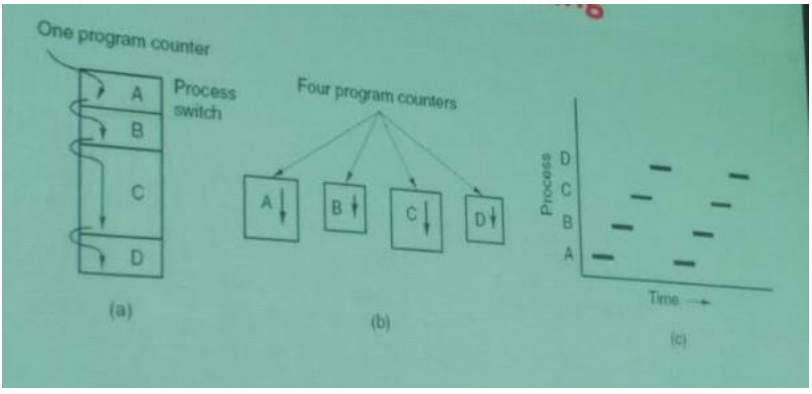
בלינוקס- איך פרוס מרחב הזכרון של תהליך?
מרחב הזכרון של כל תהליך שרץ מקבל את מרכב הזכרון מ-0 עד 4G
מרחב הזכרון ב-32 ביט זה 4G. מרחב הזכרון זה FFFFFFFF (הקסה - לחזור לזה). כלומר 2 בחזקת 32 פחות 1.

אם כך, כל מרחב הזכרון נתון לתהליך שרץ.
איך זה מחולק?
מ-0 עד גובה מסויים זה טקסט- כלומר הקוד שכתבנו (בשפת מכונה!!). הגודל של החלק הזה תלוי בתוכנה וכמה היא צריכה והגודל הזה קבוע בכל תכנית וזה יקבע בשלב הלינקינג.
בהמשך נמצא ה- data. מה יושב שם? כל המשתנים הגלובלים, הסטטים וכד', שיש להם ערך התחלתי.
אם עשיתי:
int g_i = 5 - זה יושב בדאטא.
הדאטא מחולק ל-2:
read only ו- read/ write
המשתנה למעלה יהיה ב- read write
מה יושב ב-read only?
אם יש סטרינגים שמאתחלים אותם- הם שם.
אם מנסים לשנות את ה-read only, מקבלים segmentation fault.
כשנגיע ל-memory management, נבין איך מסמנים read only

BSS- כל המשתנים שלא אותחלו. זה כולל גם את המשתנים שאיתחלתי ב- 0. הכל מחכתחילה מאופס, ולכן כשאנחנו נותנים למשתנה ערך 0, זה כאילו לא נתנו ערך התחלתי.
עד ה-heap- כל הגודל נקבע בזמן הלינקינג. הלאה משם, זה מרחבי זכרון שנקבעים באופן דינאמי בזמן הריצה

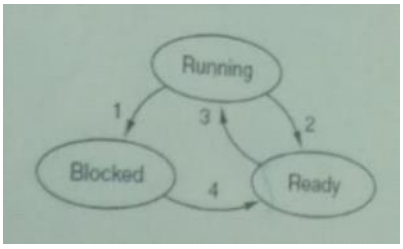
מ-4G, מלמעלה- ג'יגה שלם שייך למערכת ההפעלה.
זה כדי שיהיה לנו קשר למערכת ההפעלה. זה מרחב זכרון שלא מוקצה לתהליך שלנו. אלו שיטחי העבודה של מערכת ההפעלה- באפרים וכאלה.

אם כך, בין 3G לבין ה-BSS, זה השטח שמוקצה לסטאק ולהיפ. זה דינמי, תלוי בצורך של שניהם.
בהיפ- כל ההקצאות הדינמיות. בסטאק- כל פניה לפונקציה.



אם כך, בזמן נתון רק הליך אחד רץ כל פעם. כלומר, דיברנו על כך שיש תהליכים מקבילים. בפועל זה לא באמת מקביל, פשוט מערכת ההפעלה מייעלת את הריצה של כולם ביחד ואנחנו מקבלים את ההרגשה של ריצה כל הזמן.

כל פעם שאנחנו מפסיקים תהליך אנחנו שומרים את כל הסביבה ב-PCB- process control block
ואז כשחוזרים לרוץ זה לוקח משם.
בצורה כזו, בכל רגע נתון רץ תהליך אחד, אז יכול להיות שהוא נעצר, הכל נשמר, מתחיל תהליך אחר, אז הוא מסתיים או נעצר ויכול להתחיל תהליך אחר או לחזור לתהליך שנעצר קודם לכן.
שקף שממחיש



אמרנו שיש קוואנטום- סלייס זמן מסויים שבו תהליך רץ. לא מדובר בכמה זמן תהליך רץ אלא זמן החלטה של האם לתת לתהליך לרוץ או לא.
צריך להבין שמדובר בתהליך רנדומלי לחלוטין. אי אפשר לדעת איפה נקטעים (מבחינת איפה אני עומדת בתהליך שרץ), אף פעם אי אפשר לבנות על זה. כמו כן, אם רוצים לבדוק זמן בין שתי פקודות, אף פעם א אפשר לדעת את זה, בגלל הרנדומליות של הפעלת התהליכים השונה.

יש שלושה מצבים שבהם תהליך להיות בו:
ready- תהליך נמצא במצב זה (זה ממש קיו) כשהוא מוכן לריצה. כישיגע תור התהליך יעלה מה- PCB את הסביבה שלו והוא יעבור למצב של running.
משם זה יכול או לחזור ל- ready (עד הסיבוב הבא) או שהתהליך עצמו פנה ל-I/O, ואז זה יגיע לבלוק ויחכה לאינפוט. ברגע שמגיע האינפוט אפשר לעבור רק ל-ready, אי אפשר לעבור ישירות ל- running

צריך לזכור את שלושת המצבים האלה ואיך אפשר להגיע אליהם.
שורה תחתונה וחשובה: ל- running אפשר להגיע רק מ- ready

מתי נוצר תהליך?
דבר ראשון, באופן טבעי, כשמפעילים איזשהי פקודה - הרצה כלשהו.
תהליכים מתחילים לרוץ גם כשמערכת ההפעלה מתחילה לרוץ.
אפשר גם להפעיל תכנית בתוך תכנית.

מתי תהליך מפסיק?
מתוך תכנית עצמה ()exit מפסיק תכנית. להבדיל מ-return זה מוציא לגמרי מהתכנית. מתי עושים את זה? אם כקרה אישהו ארוע ממש לא טוב, צריך לצאת.
אפשרות אחרת - kill

איך מתוך תכנית שלי אני יכולה ליצור תכנית אחרת?
בלינוקס יש פקודה ()fork- מה שקורה בפקוה הזאת: בלנוקס יש היראכיה, כשעושים fork, עושים תכנית בן שהתכנית ממנה יצאתי זו תכנית האב. בהתחלה נוצרת תכנית שהיא זהה לזו שממנה יצאתי- שכפול מוחלט. הכל זהה לחלוטין. וכיוון שגם ה-PC- program counter זהה, הם ימשיכו מאותו מקום. הבן מקבל 0 כ- PID.
()fork זה system call שפונה למערכת ההפעלה
בוינדואס אגב זה עובד אחרת, אין היראכיה בין אב לבן. יש פקודה שנקראת create process ולא מדובר בשכפול, אלא בתהליך שרץ בנפרד.
כשבן מסיים, הוא מחזיר את ערך ההחזרה לאבא, ולכן האבא חייב לבצע wait כדי לקבל את הפקודה מהבן. בהמשך נראה מה קורה כשהוא לא עושה wait.

כאמור, יש היאראכיה בין תהליכים. לכל תהליך יש אבא אחד בלבד. לא יכול להיות תהליך שיש לו יותר.
כל ילד הוא בן של אבא מוגדר. כל בן יכול גם ליצור תהליכים מחדש. אין קשר בין נכד לאבא. לכל תהליך יש process ID.
בוינדוס כאמור אן היראכיה. ברגע שנוצר, אין שום קשר בין ה"אבא" ל"בן". זה משמעותי מבחינת מי יכול לשחרר, לעשות kill וכד'.

מה קורה בשלב ה-init?
כמשמערכת ההפעלה עולה התהליך הראשון שעולה נקרא **init** והוא למעשה מתחיל את כל התהליכים האחרים. וברור שהוא גם האחרון שיורד בסוף.
הבן מחזיר את ערכי ההחזרה לאבא, לכן האבא חייב להמתין לזה. מה קורה אם האבא לא ממתין? נוצר מצב שמערכת ההפעלה לא יכולה לסגור את המרחב הזכרון הזה. זה מצב שנקרא zomby - אם בן סיים והאבא לא קיבל את ערך החזרה. מערכת ההפעלה לא סוגרת אותו עד שהאבא סוגר אותו, או לחילופין אם האבא נסגר.
ברמת המערכת, שהיא האבא של כל התהליכים, היא קולטת את כל ה-"תהליכים"- היא קולטת את כל היתומים.
תהליך במצב זומבי- זה אומר שבן סיים והאבא לא סיים ולא עשה wait. אם האבא סיים, זה ישתחרר לבד.
התהליך עצמו, כשעושים PS, הוא לא נסגר.