***file descriptor:***

יכולים להתייחס לקבצים, directories, sockets ועוד כמה אובייקטי נתונים.
<u>הערה</u> : בדיזיין המקורי של יוניקס (מ1970) כל דבר יורש מתהליך או מקובץ בשנות ה90 נוסף ללינוקס /PROC – מערכת קבצים המציגה את כל התהליכים כקבצים ועכשיו בלינוקס "הכל" זה קובץ (כולל תהליכים). אנחנו לא למדנו (וגם לא נלמד במסגרת הקורס הזה על מערכת קבצים /PROC) ולכן להגיד ש"הכל זה קובץ" זה יכול לבלבל. אני אשאר עם הניסוח שכתוב למעלה למרות שהוא לא מדויק כל זמן שתדעו שהוא לא מדויק. נצר.

לכל תהליך (PROCESS) הרץ במערכת מחזיקה מערכת ההפעלה טבלה המפרטת את הקבצים הפתוחים

על ידי אותו תהליך.

הטבלה מנוהלת על ידי אינדקסים. כלומר תן לי אינדקס (מספר שלם לא שלילי) ובאמצעותו אגיע לכניסה

המתאימה בטבלה. אינדקס זה נקרא:file descriptor

מקובל שבכל פעם שתהליך נוצר,

יוצרת עבורו מערכת ההפעלה שלושה file descriptor באופן אוטומטי: stdin, stdout stderr

המקבלים את ערכי ה file descriptor -הבאים בהתאמה: 2, 1, 0

בעצם, הכניסות הראשונות שמורות ומוקצות עבור standart input, standart error, standart output.

לדוגמה:

| FD | Name | Other information |
|---|---|---|
| 0 | Standard Input (stdin) | … |
| 1 | Standard Output (stdout) | … |
| 2 | Standard Error (stderr) | … |

כאשר קוראים לFORK אז אז הוא משכפל את עצמו, למעשה כל file descriptors עוברים (מועתקים) לילד והם מצביעים לאותם אובייקטים שהיו בטבלה ברגע שהורה עושה fork.

חשוב להבהיר – זו העתקה DEEP – כלומר אין לנו מצביע לאותה טבלה באבא ובילד אלא שתי טבלאות שונות שמכילות (ברגע היצירה) את אותם קבצים. אם האבא או הבן יוצרים קובץ נוסף לאחר היצירה – הכניסה הזאת לא תופיע בתהליך השני

אזיש מצביע לאובייקט בטבלת הקבצים.

ניתן להעתיק אותו DEEP (כלומר להעתיק את התוכן ולא להעתיק את האינדקס!) על יד dup(2) Syscall העתקה עמוקה. זוהי העתקה עמוקה. נוצר מצביע חדש המצביע לאותו קובץ בדיוק.המצביע החדש יהיה הכניסה הפנויה הראשונה בטבלה (האחרון +1 אם אין חורים או ש"נמלא חור")–

הפונקציה dup2(2) מאפשרת לציין את האינדקס החדש למקום ידוע מראש (להבדיל מהמקום הפנוי) . בעזרת הפקודה הזאת אנחנו נדע בדיוק שהוא מעתיק לאן שנרצה ואם האינדקס החדש היה פתוח כבר, אנחנו נסגור אותו ונציב שם חדש.

קריאת המערכת דומה לdup הרגיל. (כלומר העתקה עמוקה).

signals:

הודעות הדרך של יוניקס לממש פסיקות – הודעות של מערכת ההפעלה לתהליך (בדר"כ עקב פעולות לא חוקיות.)

synchronous - קשור להרצת התוכנית כמו חלוקה ב-0 כלומר קשור לקוד (גם נגיעה בזכרון שאסור לגעת בו. הוראת אסמבלר לא מוגדרת או לא חוקית וכדומה)

Asynchronous -תוצאה מאירועים לא תלויים בקוד כמו בקשה מהתהליך להסתיים. ארוע חיצוני וכדומה, יציאה מהמערכת זמן שנגמר

דוגמאות signals :

sigfpe - הודעה שחילקו באפס.- -

ctrl+z - sigtstp

ctrl+Q - Sigabrt

ctrl+z - sigint

sigterm-סיום תהליך

פסיקה וסיגנל:

- פסיקה זה השם שנשתמש בו לנושאי חומרה (לדוגמא הגיע פקט בכרטיס רשת או תשובה לבקשת DMA מהדיסק) מטפלת בקרנל

- סיגנל זה מה שקוראים פסיקת תוכנה. הודעה ממערכת ההפעלה לתהליך, מאפשר לתהליך לסיים את עניניו הארציים, כלומר לסגור קבצים, להתנתק ועוד.

להבדיל sigkill שזה "תמות עכשיו אני רציני" (כלומר אי אפשר לתפוס את הסיגנל והוא תמיד יגרום לתהליך להסתים מיד) , sigint או sigterm או sigabrt מאפשר למשל לסגור סוקטים, לשמור את כל הקבצים בצורה מסודרת ועוד.

sigkill הורג את התהליך (נדרש כאשר תהליך נתקע). בדרך כלל נשלח sigabrt ואם התהליך לא מסתיים תוך מספר שניות אז sigkill.

5 פעולות ברירת מחדל אפשריות:

| | |
|---|---|
| -מאלץ את התוכנית לצאת. | exit |
| מאלץ את התהליך לצאת וליצור קובץ ליבה (כאשר נפל לדוגמה תהליך ורוצים את הגופה של התהליך ולמה זה נפל) | core |
| -לעצור את התהליך | stop |

| | |
|---|---|
| | |
| מתעלם מהסיגנל ללא כל פעולה אחרת | ignore |
| -המשך ביצוע של תהליך שנפסק | continue |

תפיסת signal

לכל signal יש מספר, לכל אחד יש ביט אם צריך להתעלם ממנו או לא, במידה ולא- לאיזה פונקציה צריך לקרוא(שולחים פוינטר לפונקציה).

יש 2 סיגנלים שלא ניתן לטפל בהם:

sigkill- מערכת ההפעלה הורגת את התהליך.

sigstop - עוצר את התהליך- ניתן לחזור עליו אבל הוא יצא מעיבוד כרגע. (מיד)

זה שניתן לחזור זה בדיוק ההבדל בין STOP לKILL

Signal handlers:

Signal(2) -מקבל מספר של 2 פרמטרים, איזה סיגנל לתפוס ומצביע לפונקציה.

הפונקציה שsignal(2) מקבל מקבלת int מספר הסיגנל ומחזירה VOID (בעקרון לא ניתן לקבל ערך ישירות מהפונקציה אבל ניתן לשנות משתנה גלובלי למשל.)

sigaction(2) - מבנה המתאר מה הפעולה ומה אני רוצה לעשות ,גמיש יותר.

sigprocmask() - כל הסיגנלים שאני רוצה למסך.

real time signal - מערכות בזמן אמת.

למערכות זמן אמת יש התנהגות צפויה – לדוגמא אם יש לנו משימה דחופה (למשל שמירה על יציבות של מטוס קרב) ויש משימה לא דחופה תמיד המשימה הדחופה תקבל מעבד.

זה לא התנהגות רצויה אצלנו כי בתחנת עבודה – אם יש משימה שנתקעה (למשל בלולאה אין סופית) אם היא בעדיפות גבוהה ואנחנו בתיעדוף זמן אמת – היא לא תצא לעולם.

יש דרכים לטפל בסיגנלים גם במערכות זמן אמת – מעבר לזה – מחוץ לסקופ

לא בחומר

סוגית הבטחה אבטחה: מנהל המערכת יכול לשלוח

סיגנל לכל התהליכים, כל משתמש אחר יכול רק לתהליכים שהוא יצר.

Process group ID-קבוצת תהליכים הם כולם מאותו אבא ולכן שייכים לאותה קבוצה, ניתן לשלוח signal לקבוצה מסוימת כלומר לכל התהליכים הנמצאים באותה קבוצה.

יש פונקציות שונות הקשורות ל idול group-

getpid(2)-מחזיר את ה pid-מזהה של התהליך

Getpgrp(2)-מחזיר את ה pid-מזהה של הקבוצה.

- setpgrp(2)מחזיר את ה pid-של התהליך להיות עם ה pid-שלו.

setpgrp(int pid1, int pid2)-מגדיר את pid1 pgid של התהליך להיות שווה ל pid -שלpid2