מערכות הפעלה – יבש 2

חלק א

1. הפקודה מדפיסה למסך מחרוזת בלולאה אינסופית עד שהורגים את התהליך עצמו. הארגומנטים שהיא מקבלת היא המחרוזת להדפסה string וכן את הדגלים הבאים:

* help - עזרה
* version - לקבלת הגרסה.

1. הסיבה שהשתמשנו בפקודה yes '' היא על מנת לשלוח לmake oldconfig תו ירידת שורה "\n" וזאת על בשביל לדמות לחיצה על מקש "enter". כך שנדחה כל שינוי שמבקשת הפקודה make oldconfig. למעשה נקבל את ערכי ברירת המחדל של הפקודה. הבעיה בהרצה ידנית של הפקודה make oldconfig היא שנאלץ להתייחס להרבה "בקשות" מצד הפקודה באופן ידני, מה שייקח הרבה זמן.
2. זה הזמן שניתן לנו כמשתמשים לבחור איזו גרסה של מערכת ההפעלה תטען.
   1. יתרונות – בהינתן שיש רשימה ארוכה או רשימה מאוד מפורטת של גרסאות זמינות למערכת ההפעלה אשר ניתן לטעון בזמן עליית המחשב אזי נקבל מספיק זמן לבחור את הגרסה הנכונה (מקל על קריאות).
   2. חסרונות – מעקב את עליית מערכת ההפעלה בעת עלייה אוטומטית כאשר יודעים בוודאות מה הגרסה שבדרך כלל רוצים להעלות ולכן אין צורך בהמתנה ארוכה.
3. כאשר הפונקציה run\_init\_process() רצה, המערכת נמצאת בהרשאות kernel, ולכן קוראת לפונקציה של מערכת ההפעלה do\_execve שכן אין צורך לקרוא לקריאת מערכת ממצב kernel, ובפרט ל- execve.
4. Syscall() היא פונקציית הספריה "sys/syscall.h" אשר מקבלת לכל היותר 7 ארגומנטים, כאשר:

* הראשון הינו מספר קריאת המערכת
* 6 ארגומנטים נוספים אפשריים בהתאמה למספר הארגומנטים הנדרשים לקריאת המערכת.

1. הקוד הנתון מדפיס את הpid של התהליך אשר מריץ את תכנית main הנתונה.

#include <iostream>

#include <unistd.h>

int main () {

pid\_t pid = getpid();

std::cout << "process pid " << pid << std::endl;

return 0;

}

1. בודק שהתוכנית המתינה "timeout" זמן עד כדי שגיאה זניחה על ידי בדיקת הפרשי "vruntime" לפני ואחרי המתנה.
2. Taskset היא פקודה שמורה למערכת ההפעלה להריץ תהליך מסויים אך ורק על רשימת המעבדים שניתנה כארגומנט. במהלך ריצת התוכנית עלולה להתרחש החלפת הקשר בעקבותיה עלול התהליך להתחיל לרוץ על מעבד אחר, מה שיגרור לכך שהתהליך יעבור למעבד אחר עם עץ אדום שחור משלו ויקבל בו את הmaxvruntime (כפי שראינו בתרגול) ואז הטסט יחזיר שגיאה כיוון שהוא יקבל הפרש גדול מידי בשביל בדיקת הassert.
3. התוכנית הבאה מהווה בדיקה המריצה שני תהליכים רגילים בעלי עדיפות NICE=0 על אותה ליבת מעבד למשך 5 שניות ובודקת שהם חלקו שווה בשווה את זמן המעבד:

#include "hw2\_test.h"

#include<sys/wait.h>

#include<sys/resource.h>

int main() {

double timeout = 5.0;

double vtime = 0;

double measured\_vruntime\_first = 0;

double measured\_vruntime\_second = 0;

int which = PRIO\_PGRP;

id\_t pid = getpid();

int priority = 0;

setpriority(which, pid, priority);

int firstPipe[2];

int secondPipe[2];

pipe(firstPipe);

pipe(secondPipe);

Stopwatch stopwatch;

pid\_t first = fork();

if (first < 0) {

perror("Error while forking");

return -1;

}

else if (first == 0) { /// First child process

vtime = get\_vruntime();

while (stopwatch.Read() < timeout); // spin

measured\_vruntime\_first = get\_vruntime() - vtime;

close(firstPipe[0]);

dup2(firstPipe[1], 1);

close(firstPipe[1]);

write(1, &measured\_vruntime\_first, sizeof(measured\_vruntime\_first));

exit(0);

}

else { /// Father

pid\_t second = fork();

if (first < 0) {

perror("Error while forking");

return -1;

}

else if (second == 0) { /// Second child process

vtime = get\_vruntime();

while (stopwatch.Read() < timeout); // spin

measured\_vruntime\_second = get\_vruntime() - vtime;

close(secondPipe[0]);

dup2(secondPipe[1], 1);

close(secondPipe[1]);

write(1, &measured\_vruntime\_second, sizeof(measured\_vruntime\_second));

exit(0);

}

else { /// Father

close(firstPipe[1]);

close(secondPipe[1]);

read(firstPipe[0], &measured\_vruntime\_first, sizeof(measured\_vruntime\_first));

close(firstPipe[0]);

read(secondPipe[0], &measured\_vruntime\_second, sizeof(measured\_vruntime\_second));

close(secondPipe[0]);

while (wait(NULL) > 0){}

AssertRelativeError(measured\_vruntime\_first, measured\_vruntime\_second);

cout << "===== SUCCESS =====" << endl;

}

}

return 0;

{

חלק ב

1. נשים לב כי תחת ההנחות הנתונות, ההפרש הגדול ביותר האפשרי בין הזמן הווירטואלי המקסימלי והזמן הווירטואלי המינימלי של התהליכים בסוף ה-Epoch הינו **0**. עבור כל תהליך שרץ ב-Epoch, זמן הריצה הווירטואלי שלו גדל באופן הבא:

*וזהו גודל שאינו תלוי ישירות במשקלו של התהליך ה-i, אלא במשקל כל התהליכים. על כן, השינוי בזמן הווירטואלי של כל התהליכים זהה ללא תלות במשקלם ולכן ההפרש המקסימלי יהיה 0.*

1. נשים לב להנחות הנתונות:
   1. אלגוריתם CFS שואף שהזמן הווירטואלי בין כל התהליכים יהיה זהה ובפרט בסוף כל Epoch לכל התהליכים יהיה את אותו זמן וירטואלי.
   2. עבור כלשהו.
   3. גודל ה-Epoch הוא קבוע ומקיים

ולכן מתקיים:

2. נכתוב חסרון עבור כל שינוי שאמיר ביצע:
   1. במידה ותהליך החוזר מהמתנה יחזור עם אותו הזמן הווירטואלי שהיה לו לפני כן, אלגוריתם ה-CFS יתעדף אותו לריצה על פני תהליכים אחרים מה שיכול לגרום להרעבה של יתר התהליכים. למשל, שני תהליכים A, B הרצים זה לצד זה ותהליך B יוצא להמתנה של 10 שניות. כאשר תהליך B מתעורר הוא נכנס לעץ עם זמן ווירטואלי הקטן ב-10 שניות מזה של A ולכן אלגוריתם ה-CFS יתעדף את B שירוץ ב-10 השניות הבאות וירעיב את A.
   2. במידה ונבחר להשתמש במבנה נתונים של רשימה מקושרת חד כיוונית, זמן ההכנסה, הוצאה וחיפוש של תהליך ברשימה נעשה בסיבוכיות זמן O(N) כאשר בעץ אדום-שחור זה נעשה בסיבוכיות זמן O(logN).
   3. במידה ונסיר את המינימום על גודל הקוונטום, אם במערכת הנתונה ישנו מספר גבוה של תהליכים, המערכת עלולה לסבול מהחלפות הקשר תכופות ופגיעה בביצועים שכן חלק גדול מזמן הריצה ילך להחלפות הקשר במקום ללכת לטובת ריצת התהליכים.
   4. נשים לב כי ה-Epoch עלול לגדול באופן משמעותי ועל כן זמן התגובה של המערכת ירד משמעותית בהתאם.
   5. נשים לב כי ייווצרו הפרשים בזמני הריצה הווירטואלים בסוף כל Epoch (שכן התהליכים בעלי משקל נמוך יוסיפו הרבה זמן הווירטואלי שלהם ואלו שיש להם משקל גבוה יותר יוסיפו מעט) שהאלגוריתם ישאף לצמצם ועלולה להיווצר הרעבה של תהליכים בעלי משקל נמוך.
   6. *במידה ותהליך חדש מגיע למערכת עם זמן וירטואלי השווה לזמן הווירטואלי הממוצע של כל התהליכים במערכת, אלגוריתם ה-CFS ישאף להריץ תהליכים עם זמן ריצה וירטואלי נמוך מזה של התהליך שהתווסף לתור ועל כן תיווצר הרעבה של תהליך זה.*
3. *בכדי "לגנוב" זמן מעבד, תהליך יכול ליצור לעצמו חוטים נוספים שיבצעו חלקים מעבודתו. כאשר תיעשה החלפת הקשר, תהליכים אלו יקבלו תיעדוף על פני תהליכים אחרים וירעיבו את התהליכים האחרים.*
4. *ננתח את הקשר בין ערך ה-nice של תהליך לבין משקלו.*

*לפי הגרף לעיל ניתן לשים לב כי הקשר המתמטי בין ערך ה-nice ללוג המשקל התהליך הינו קשר* ***לינארי*** *ומכאן ניתן להסיק כי הקשר בין ערך ה-nice לבין משקל התהליך הינו* ***מעריכי****.*