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 תקורה גדולה יותר להחלפה ובעקבות זאת הפרשים לא נכונים.
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**. נשים לב כי אם כל התהליכים מנצלים את מלוא הקוונטום שניתן להם, הם רצים את אותו הזמן הווירטואלי ןלכן ההפרש בין הזמן הווירטואלי המקסימלי לזמן הווירטואלי המינימלי הינו 0. זאת משום שאם נניח כי לכל התהליכים אותו המשקל, זמן הריצה הווירטואלי של כל התהליכים הוא זהה שכן הקוונטום של כל התהליכים הוא ועל כן היחס בין כל זמני הריצה הווירטואלים הינו 1.

(בפועל זה לא נכון אבל משום מה אמרו שניתן להניח שהמשקלים של כולם זהים... עבור משקל מקסימלי ומשקל מינימלי נקבל הפרש מקסימלי גדול מאפס...)

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: