ליאור דניאל
305257347

שאלה 3 – צילום מסך

הסבר: לאחר הרצת הפקודה 'make' המקמפלת את הקובץ set_policy, נותר להריץ את הקובץ a.out ולצרף אליו את מספר המדיניות ואת מספר הפריוריטי לשינוי. הקוד בנוי בצורה הבאה:

1. מדפיס את מספר הזהות של התהליך הנוכחי.
2. מקבל את מספר העדיפות של התהליך ושומר אותו לתוך המשתנה priority שבתוך ה-struct שיצרנו הנקרא sp ומדפיס את העדיפות.
3. מקבל את מספר המדיניות הקיים אצל אותו תהליך ומדפיס אותו.
4. משנה את העדיפות והמדיניות עבור התהליך הנוכחי.
   אם הפעולה לא הצליחה והתקבלה שגיאה – התוכנית תצא ותחזיר '1'.
6. אם השינוי הצליח, נדפיס את הערכים העכשוויים.

Select lior@DESKTOP-8CP0JPA: /mnt/d/Compuer Science/Current/Operation System/Assignments/Final Work/Q3

```
lior@DESKTOP-8CP0JPA:/mnt/d/Compuer Science/Current/Operation System/Assignments/Final Work/Q3$ make
gcc set_policy.c
lior@DESKTOP-8CP0JPA:/mnt/d/Compuer Science/Current/Operation System/Assignments/Final Work/Q3$ sudo ./a.out 1 5
pid: 258

current priority: 0
current policy: 0 - SCHED_OTHER

CHANGES SUCCESSFULL!!!

current priority: 5
current policy: 1 - SCHED_FIFO
lior@DESKTOP-8CP0JPA:/mnt/d/Compuer Science/Current/Operation System/Assignments/Final Work/Q3$
```

ליאור דניאל
305257347

שאלה 3 – סיכומון.

כחלק מסיכום מורחב שסיכמתי עבור הקורס, לקחתי את החלקים הרלוונטים ממנו עבור שאלה 3, הכוללים את הפקודות הרלוונטיות ואת האלגוריתמים השונים לתזמון.

פקודות רלוונטיות לשאלה 3:

- ps -xl – צפייה בתהליכים הפתוחים עם הפריוריטי שלהם.
- nice – מאפשר לקבוע את הפריוריטי של התהליך לפני הרצתו (הפקודה זמנית ורלוונטית עבור אותה הרצה בלבד).
  - דוגמא לפקודה: nice -n <priority> pico <name of file>
- renice – מאפשר לקבוע את הפריוריטי של התהליך בזמן ריצה (הפקודה זמנית ורלוונטית עבור אותה הרצה בלבד).
  - דוגמא לפקודה: renice <priority> -p <pid>
- renice – מאפשר לקבוע את הפריוריטי של התהליך בזמן ריצה (הפקודה זמנית ורלוונטית עבור אותה הרצה בלבד).
  - דוגמא לפקודה: renice <priority> -p <pid>
- taskset – מאפשר לראות או לקבוע את הליבה או את הליבות שעליהן התהליך ירוץ.
  - דוגמא לפקודה המציגה על איזה ליבה\ות התהליך רץ: ps -cp <pid>
  - דוגמא לפקודה לשינוי הליבה של תהליך: pid -p <cpu number> <pid>
- chrt – מאפשר לראות ולקבוע את הפריוריטי של התהליך ואת הפוליסה (מנגנון התזמון) שלו.
  - דוגמא לפקודה: chrt -p <pid>
  - דוגמא לפקודה: chrt <-policy_symbol_char> -p <priority> <pid>

אלגוריתמים שונים של תזמון

FCFS – First Come First Served

המעבד מטפל בתהליך הראשון שמבקש. הדרך הפשוטה ביותר למימוש האלגוריתם היא על ידי תור FIFO, המתזמן התהליכים לפי אופן הגעתם ל-Ready Queue.
זהו אלגוריתם Non-Preemptive, וזמן הממוצע שלו לרב ארוך, כי תלוי בזמן ההגעה של התהליכים.

דוגמא:

Process	Burst Time
P1	24
P2	3
P3	3

ניתן לראות כי זמן ההמתנה של P1 = 0, של P2 = 24 ושל P3 = 27.
לכן זמן ההמתנה הממוצע הוא : (0+24+27)/3 = 17.

לעומת זאת אם הסדר היה P1 -> P3 -> P2

ניתן לראות כי זמן ההמתנה של P2 = 0, של P3 = 3 ושל P1 = 6.
לכן זמן ההמתנה הממוצע הוא : (0+3+6)/3 = 3.

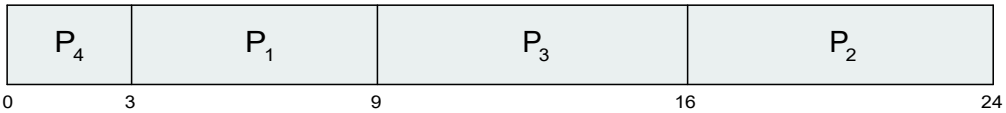
SJF – Shortest Job First

אלגוריתם אופטימאלי הנותן זמן המתנה מינימאלי. צריך לשים לב כי אם יש הרבה תהליכים קצרים ואחד ארוך, ייתכן כי הוא יחכה המון זמן. לכן, ה-Turnaround Time לא אופטימאלי כלל.

דוגמא:

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

שימוש באלגוריתם זה יתזמן את התהליכים באופן הבא: P4 -> P1 -> P3 -> P2.



זמן ההמתנה הממוצע יהיה (3+16+9+0)/4 = 7.

הבעיה : לא נוכל לדעת את אורך התהליך. איך אפשר "לנחש"?

פתרון : לפי הנוסחה $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n.$

ה-α משמש כמקדם כך ש: $0 \le \alpha \le 1$ (בד"כ 0.5)

הניחוש שלנו לתהליך הבא.

האורך האמיתי שהיה לתהליך הקודם ב-CPU.

הניחוש שקיבלנו לתהליך הקודם.

- $\alpha = 0$: $\tau_{n+1} = \tau_n$ - נתייחס לניחוש שקיבלנו בתהליך הקודם ולא לאורך האמיתי.
- $\alpha = 1$: $\tau_{n+1} = \alpha\, t_n$ - נתייחס רק לאורך האמיתי שלקח לתהליך הקודם.
- הנוסחה הרקורסיבית :

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\alpha\, t_{n-1} + \ldots + (1-\alpha)^j \alpha\, t_{n-j} + \ldots + (1-\alpha)^{n+1}\tau_0$$

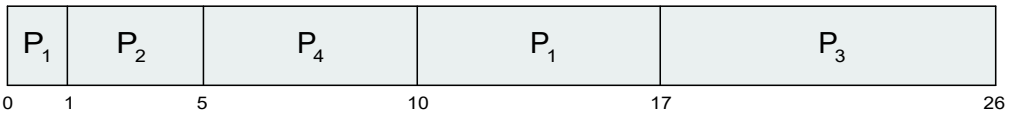
הסיבה שה-α בין 0 ל-1, נשים לב שעל כל שפעה קטנה יותר מאשר על אלו

Shortest Remaining Time First

גרסת ה-Preemptive של ה-SJF.
באלגוריתם זה, גם אם התחלנו לטפל בתהליך מסוים, ברגע שקיבלנו תהליך עם זמן קצר יותר – נפסיק ונעבור אליו.
כאן נוסיף פרמטר נוסף – זמן הגעת התהליך לתור ההמתנה לפי יחידות זמן.

דוגמא:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5



הסבר: התחלנו עם P1 אבל אחרי יחידת זמן אחת קיבלנו את P2.
כרגע P1 = 7 ו-P2 = 4, לכן P2 קצר יותר והמתזמן עוצר את P1 ומטפל ב-P2.
P3 ו-P4 הגיעו תוך כדי הטיפול ב-P2, אך הם עדיין ארוכים יותר ממנו ולכן P2 מסיים את התהליך. המתזמן עובר ל-P4 הקצר אחריו, משם ל-P1 ואז P3.

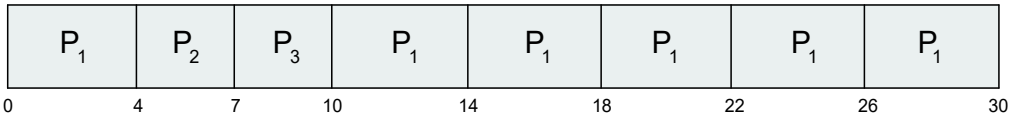
זמן ההמתנה הממוצע יהיה [(10-1) + (1-1) + (17-2) + (5-3)] / 4 = 26 / 4 = 6.5
P1 P2 P3 P4

Round-Robin Scheduling

אלגוריתם Preemptive.
כל תהליך מקבל זמן קבוע לריצה, הנקרא Quantum (בד"כ בין 10-100 מיל" שניות).
האלגוריתם מתייחס ל-Ready-Queue כאל תור מעגלי.
מתזמן המעבד עובד על התור ומקצה Quantum פרק זמן עיבוד בכל פעם לתהליך אחר, כאשר הזמן עבור התהליך מסתיים, התהליך עובר לסוף התור.
אם יש n תהליכים ב-Ready Queue אז אין תהליך שיחכה יותר מ (n-1)*Quantum יחידות זמן.
נשים לב שאם ה-Quantum יהיה גדול מידי נקבל FCFS.
לעומת זאת, נשים לב שה-Quantum לא קטן מידי מאחר וכל החלפת תהליך דורשת Context Switch ש"המחיר" עבורו גבוהה.

דוגמא:

Process	Burst Time
P1	24
P2	3
P3	3



במקרה זה ה-Quantum = 4.
המתזמן מתחיל טיפול ב-P1 ולאחר 4 יחידות זמן מפסיק ועובר ל-P2. מסיים את P2 לאחר 3 יחידות זמן ועובר ל-.P3 מסיים אותו לאחר 3 יחידות זמן וחוזר ל-P1 עד שמסיים אותו.

* זמן ההמתנה הממוצע – לבדוק *

Priority Scheduling

אלגוריתם Non-Priority.
אלגוריתם זה נותן זמן ריצה לתהליך בעל העדיפות הגבוהה ביותר.
ככל שמספר ה-Priority נמוך יותר, כל העדיפות עבור התהליך גבוהה יותר.

בעיה: תהליכים עם עדיפות נמוכה עלולים לא לקבל זמן עיבוד אף פעם.

פתרון: Aging (תהליך הזדקנות) – ככל שעובר זמן ותהליך מסוים לא קיבל עדיפות, נעלה לו את העדיפות באופן יזום.

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4

דוגמא:

P4	1	5
P5	5	2

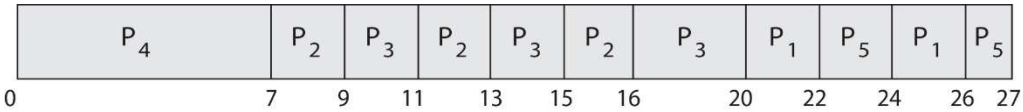
זמן הממוצע הוא 8.2 = 5 / (0+1+6+16+18+19).

Priority Scheduling w/ Round-Robin

נריץ את Priority אלגוריתם, אך במידה וניתקל בשני תהליכים עם אותה העדיפות, נתזמן ביניהם עם אלגוריתם Round-Robin.

דוגמא:

Process	Burst Time	Priority
P1	4	3
P2	5	2
P3	8	2
P4	7	1
P5	3	3



Multilevel Queue

ניצור תור תהליכים שונה עבור כל רמת עדיפות. עבור כל תור נוכל לבחור אלגוריתם תזמון שונה.
מתחילים לטפל בתור של הרמה הכי נמוכה וכשמסיימים עוברים לרמה הבאה אחריה.

Multilevel Feedback Queue

אלגוריתם הבנוי על בסיס Multilever Queue אך כאן נוסיף אינטרקציה בין התורים על מנת להימנע מתהליכים שיישכחו.
כך נוכל לבצע Aging – הגדלת העדיפות של תהליך באופן יזום עבור אחד כזה שנשכח.

EDF – Earliest Deadline First

לכל תהליך יש זמן ריצה, זמן הגעה לתור ו-Dead Line. כלומר, אפליקציה יכולה להגדיר שהיא רוצה שיטפלו בתהליך בתוך זמן מסוים. ככל שה-Dead Line יותר קרוב, כך העדיפות גדלה.
יש פרמטר נוסף שהתהליך מעביר איתו ומסמל כמה "כסף" האפליקציה מוכנה לשלם למערכת במקרה שהתהליך יטופל בזמן, אך האלגוריתם אינו מתייחס לערך זה ולכן הוא טוב עבור תהליכים שערכם בפרמטר זה שווה.

Linux Scheduling

עובד עם CFS – Completely Fair Scheduler.

nice value – עדיפות התהליך. ככל שיותר קטן העדיפות גבוהה יותר.
target latency – הזמן שבו תהליך אמור לרוץ לפחות פעם אחת (יכול להשתנות לפי כמות התהליכים).
virtual run time – כמה זמן באמת התהליך קיבל זמן עיבוד (ככל שקיבל יותר כך המספר יותר נמוך).

במימוש נכניס את התהליכים לעץ אדום שחור כך שמצד שמאל יהיו התהליכים עם ה-virtual run time הכי נמוך ובימין הכי גבוה.
במקרה בו יש כמה מעבדים, נוודא שתהליך שהתחיל במעבד מסוים גם ימשיך ויסיים איתו.

Windows Scheduling

בנוי על Preemptive ומתזמן לפי עדיפות (לכל רמת עדיפות תור משלה).
אם אין תהליך שירוץ, מריצים תהליך סרק.
תהליכים שהם real-time יכולים לעקוף כאלה שלא.

יש 32 רמות עדיפות :
0 – תהליכים של ניהול זיכרון.

Variable Class – 1-15
Real-Time Class – 16-32

Solaris

מתזמן לפי עדיפות.
בעל 6 Class'ים, כך שלכל Class יש אלגוריתם תזמון משלה. כל תהליך יכול להיות ב-Class אחד בו זמנית.

.Fixed Priority (FP), Fair Share (FSS), System (SYS) ,Time Sharing (TS), Interactive (IA), Real Time (RT)