מערכות הפעלה
מטלה 3

מגישות :
לין טיבי 318232139
קורל סרב 207972282

FCFS:

נעבור על מערך התהליכים הממוין ובכל איטרקציה בה נמצא תהליך אשר הגיע עד הזמן הנוכחי (time >= processes[i].arrival) נבצע הרצה לתהליך ונסיים אותו (ראה שלבים 2.2-2.4).

במידה ולא נמצא תהליך כנ"ל נקדם את הזמן לזמן הקרוב ביותר בו יגיע תהליך.

שימו לב: בעת <u>סיום הרצה לתהליך</u>, אנו נקדם את הזמן הנוכחי בזמן הריצה שלו (burst) ונחשב את זמן ההרצה שלו באמצעות חיסור בין זמן סיום ההרצה (הזמן הנוכחי המעודכן) לזמן ההגעה (arrival).

<u>FCFS(processes, length)</u>

```
1        time ← 0, sum ← 0, turnaround ← 0
2        for i 0 → length
2.1          if time < processes[i].arrival
2.1.1                time ← processes[i].arrival
2.2          time ← time + processes[i].burst
2.3  turnaround ← time - processes[i].arrival
2.4  sum ← sum + turnaround
3        return sum/length
```

LCLS non preemptive:

באלגוריתם זה השתמשנו במבנה נתונים מחסנית (stack) מבוססת lifo אשר באמצעותה נוכל להוציא ולהשתמש בתהליך האחרון שנכניס אליה.

נעבור על מערך התהליכים הממוין ובכל איטרקציה נמצא את כל התהליכים אשר הגיעו עד הזמן הנוכחי (time >= processes[i].arrival) ונכניס אותם למחסנית.

במידה ולא נמצאו תהליכים כנ"ל (המחסנית ריקה) נקדם את הזמן לזמן הקרוב ביותר בו יגיע תהליך.

אחרת, נוציא את התהליך האחרון שהוכנס (שהגיע) מהמחסנית, נבצע לו הרצה ונסיים אותו (ראה שלבים 3.3.2-3.3.4).

שימו לב: בעת <u>סיום הרצה לתהליך</u>, אנו נקדם את הזמן הנוכחי בזמן הריצה שלו (burst) ונחשב את זמן ההרצה שלו באמצעות חיסור בין זמן סיום ההרצה (הזמן הנוכחי המעודכן) לזמן ההגעה (arrival).

<u>LCLS non preemptive(processes, length)</u>

```
1       time ← 0, sum ← 0, turnaround ← 0
2       i ← 0
3       while i < length or stack not empty
3.1         while i < length and processes[i].arrival <= time
3.1.1               stack.push( processes[i] )
3.1.2               i ← i + 1
3.2         if stack is empty
3.2.1               time ← processes[i].arrival
3.3         else
3.3.1               p ← stack.pop()
3.3.2               time ← time + p.burst
3.3.3               turnaround ← time - p.arrival
3.3.4               sum ← sum + turnaround
4       return sum/length
```

LCLS preemptive:

באלגוריתם זה השתמשנו במבנה נתונים מחסנית (stack) מבוססת lifo אשר באמצעותה נוכל להוציא ולהשתמש בתהליך האחרון שנכניס אליה.

נעבור על מערך התהליכים הממוין ובכל איטרקציה נמצא את כל התהליכים אשר הגיעו עד הזמן הנוכחי (time >= processes[i].arrival) ונכניס אותם למחסנית.

במידה ולא נמצאו תהליכים כנ"ל (המחסנית ריקה) נקדם את הזמן לזמן הקרוב ביותר בו יגיע תהליך.

אחרת, נוציא את התהליך האחרון שהוכנס (שהגיע) מהמחסנית, ונבדוק האם התהליך יסיים את פעילותו בסוף מחזור השעון הנוכחי (ראה שלב 3.3.2). אם כן, נבצע לו הרצה ונסיים אותו (ראה שלבים 3.3.2.1-3.3.2.3). אחרת, נבצע לו הרצה למשך מחזור שעון אחד (נקטין את זמן הריצה שלו) ונחזירו למחסנית (ראה שלבים 3.3.3.1-3.3.3.3).

שימו לב: בעת <u>סיום הרצה לתהליך</u>, אנו נקדם את הזמן הנוכחי בזמן הריצה שלו (burst) ונחשב את זמן ההרצה שלו באמצעות חיסור בין זמן סיום ההרצה (הזמן הנוכחי המעודכן) לזמן ההגעה (arrival).

```
LCLS preemptive(processes, length)
1        time ← 0, sum ← 0, turnaround ← 0
2        i ← 0
3        while i < length or stack not empty
3.1          while i < length and processes[i].arrival <= time
3.1.1                stack.push( processes[i] )
3.1.2                i ← i + 1
3.2          if stack is empty
3.2.1                time ← processes[i].arrival
3.3          else
3.3.1                p ← stack.pop()
3.3.2                if p.burst <= 1
3.3.2.1                      time ← time + p.burst
3.3.2.2                      turnaround ← time - p.arrival
3.3.2.3                      sum ← sum + turnaround
3.3.3                else
3.3.3.1                      time ← time + 1
3.3.3.2                      p.burst ← p.burst - 1
3.3.3.3                      stack.push( p )
4        return sum/length
```

Robin Route with 2 time quantum:

באלגוריתם זה נשתמש במשתנה עזר (counter) אשר יכיל את מספר התהליכים שהסתיימו.
עבור כל תהליך שזמן הריצה שלו (burst) הוא 0 (נבדוק זאת גם מראש – ראה שלב 3), יגדל ערך המשתנה.

נעבור על מערך התהליכים הממוין עד אשר נוודא שכל התהליכים הסתיימו
ובכל איטרקציה נבדוק מספר דברים:

1- האם עברנו על גבי כל המערך ויש להתחיל מההתחלה? (ראה שלב 4.1)
2- האם התהליך לא הגיע לפני הזמן הנוכחי (time < processes[i].arrival)?
ואם כן, האם כל התהליכים עד לתהליך זה הסתיימו (יש לקדם את הזמן הנוכחי בהתאם)?
או שיש לרוץ מההתחלה על התהליכים הנ"ל?
(ראה שלב 4.2).
3- האם התהליך הגיע לפני הזמן הנוכחי (time >= processes[i].arrival) ואינו הסתיים (processes[i].burst ≠ 0)?
אם כן, יש לבדוק האם התהליך יסיים את פעילותו בסוף ה-time quantum הנוכחי (ראה שלב 4.3.1.1). אם כן, נבצע לו הרצה, נסיים אותו ונגדיל את מספר התהליכים שהסתיימו (counter).
אחרת, נבצע לו הרצה למשך ה-time quantum הנתון (נקטין את זמן הריצה שלו).
(ראה שלב 4.3).

שימו לב: בעת <u>סיום הרצה לתהליך</u>, אנו נקדם את הזמן הנוכחי בזמן הריצה שלו (burst) ונחשב את זמן ההרצה שלו באמצעות חיסור בין זמן סיום ההרצה (הזמן הנוכחי המעודכן) לזמן ההגעה (arrival).

RR(processes, length)

```
1          time ← 0, sum ← 0, turnaround ← 0
2          i ← 0, counter ← 0
3          for i 0 → length
3.1            if processes[i].burst = 0
3.1.1                  counter ← counter + 1
4          while counter < length
4.1            if i = length
4.1.1                  i ← 0
4.2            else if processes[i].arrival > time
4.2.1                  if i <= length
4.2.1.1                        time ← processes[i].arrival
4.2.2                  else
4.2.2.1                        i ← 0
4.3            else
4.3.1                  if processes[i].burst ≠ 0
4.3.1.1                        if processes[i].burst <= TIME QUANTUM
4.3.1.1.1                              time ← time + processes[i].burst
4.3.1.1.2                              turnaround ← time – processes[i].arrival
4.3.1.1.3                              sum ← sum + turnaround
4.3.1.1.4                              processes[i].burst ← 0
4.3.1.1.5                              counter ← counter + 1
4.3.1.2                        else
4.3.1.2.1                              time ← time + TIME QUANTUM
4.3.1.2.2                              processes[i].burst ← processes[i].burst – TIME QUANTUM
4.3.1.3                        i ← i +1
5          return sum/length
```

SJF preemptive:

באלגוריתם זה השתמשנו במבנה נתונים תור עדיפויות (priority queue) מבוסס פונקציית השוואה בין זמן הריצה של תהליכים (burst) אשר באמצעותו נוכל להוציא ולהשתמש בתהליך הקצר ביותר שנכניס אליו (בעל ה-burst הקטן ביותר).

נעבור על מערך התהליכים הממוין ובכל איטרקציה נמצא את כל התהליכים אשר הגיעו עד הזמן הנוכחי (time >= processes[i].arrival) ונכניס אותם לתור העדיפויות.

במידה ולא נמצאו תהליכים כנ"ל (התור ריקה) נקדם את הזמן לזמן הקרוב ביותר בו יגיע תהליך.

אחרת, נוציא תהליך מהתור, ונבדוק האם התהליך יסיים את פעילותו בסוף מחזור השעון הנוכחי (ראה שלב 4.3.2). אם כן, נבצע לו הרצה ונסיים אותו (ראה שלבים 4.3.2.1-4.3.2.3).

אחרת, נבצע לו הרצה למשך מחזור שעון אחד (נקטין את זמן הריצה שלו) ונחזירו לתור (ראה שלבים 4.3.3.1-4.3.3.3).

שימו לב: בעת <u>סיום הרצה לתהליך</u>, אנו נקדם את הזמן הנוכחי בזמן הריצה שלו (burst) ונחשב את זמן ההרצה שלו באמצעות חיסור בין זמן סיום ההרצה (הזמן הנוכחי המעודכן) לזמן ההגעה (arrival).

SJF preemptive(processes, length)

```
1        time ← 0, sum ← 0, turnaround ← 0
2        i ← 0
3        queue is a priority queue
                 sorted upwords by <process>.burst
                 (shortest burst first)
4        while i < length or queue not empty
4.1          while i < length and processes[i].arrival <= time
4.1.1                queue.enqueue( processes[i] )
4.1.2                i ← i + 1
4.2          if stack is empty
4.2.1                time ← processes[i].arrival
4.3          else
4.3.1                p ← queue.dequeue()
4.3.2                if p.burst <= 1
4.3.2.1                      time ← time + p.burst
4.3.2.2                      turnaround ← time - p.arrival
4.3.2.3                      sum ← sum + turnaround
4.3.3                else
4.3.3.1                      time ← time + 1
4.3.3.2                      p.burst ← p.burst - 1
4.3.3.3                      queue.enqueue( p )
5        return sum/length
```

צילומי מסך

```
lintibi1@com3:~$ cat input1.txt
 4
 3,5
 5,8
 1,10
 6,9
lintibi1@com3:~$ ./main.exe input1.txt

 input1.txt
 FCFS: mean turnaround = 17.25
 LCFS (NP): mean turnaround = 19.25
 LCFS (P): mean turnaround = 20
 RR: mean turnaround = 23.75
 SJF: mean turnaround = 16.5
```

```
lintibi1@com3:~$ cat input2.txt
 8
 3,5
 5,8
 1,10
 6,9
 1,8
 1,2
 4,7
 9,4
lintibi1@com3:~$ ./main.exe input2.txt
 FCFS: mean turnaround = 28.125
 LCFS (NP): mean turnaround = 30.125
 LCFS (P): mean turnaround = 30.75
 RR: mean turnaround = 37.375
 SJF: mean turnaround = 21.625
```

```
lintibi1@com3:~$ cat input3.txt
 2
 3,5
 50,18
lintibi1@com3:~$ ./main.exe input3.txt
 FCFS: mean turnaround = 11.5
 LCFS (NP): mean turnaround = 11.5
 LCFS (P): mean turnaround = 11.5
 RR: mean turnaround = 11.5
 SJF: mean turnaround = 11.5
```

```
lintibi1@com3:~$ cat input4.txt
4
2,5
3,6
3,7
4,8
lintibi1@com3:~$ ./main.exe input4.txt
FCFS: mean turnaround = 14
LCFS (NP): mean turnaround = 14.75
LCFS (P): mean turnaround = 17.25
RR: mean turnaround = 20
SJF: mean turnaround = 14
```

```
lintibi1@com3:~$ cat input5.txt
4
3,0
5,8
1,0
6,9
lintibi1@com3:~$ ./main.exe input5.txt
FCFS: mean turnaround = 6
LCFS (NP): mean turnaround = 6
LCFS (P): mean turnaround = 6.5
RR: mean turnaround = 7.5
SJF: mean turnaround = 6
```