گزارش آزمایش ۸
شبیه‌سازی الگوریتم‌های زمانبندی

نام‌و نام خانوادگی:روژینا کاشفی و هلیا سادات هاشمی پور نام استاد:سرکار خانم علیزاده تاریخ: ۱۴۰۰/۹/۲۷

بخش اول

برای پیاده‌سازی الگوریتم first come first serve میدانیم آن مبتنی بر FIFO است و یک الگوریتم غیر preemptive و pre_preemptive است.

```c
#include<stdio.h>
#include "time.h"

struct process {
    int pid;
    int bt;
    int wt;
    int tt;
};

struct process p[20];

int main() {
    int n;
    int i, j;
    int sum_wt=0,sum_tt=0;
    float average_wt, average_tt;
    clock_t startTime = clock();
    printf("Number of process= ");
    scanf("%d", &n);

    printf("\nProcess Burst Time\n");

    for (i = 0; i < n; i++) {
        printf("*P[%d]= ", i + 1);
        scanf("%d", &p[i].bt);
    }

    for (i = 0; i < n; i++) {
        p[i].pid = i + 1;
    }

    p[0].wt = 0; //initial to zero

    //calculate waiting time
    for (i = 1; i < n; i++) {
        p[i].wt = 0;
        for (j = 0; j < i; j++)
            p[i].wt = p[i].wt + p[j].bt;
    }

    printf("\nProcess\t|Burst Time\t    |Waiting Time\t|Turnaround Time\n");

printf("_____________________________________________________");

    int count = 0;
    //calculate turnaround time
    for (i = 0; i < n; i++) {
        p[i].tt = p[i].bt + p[i].wt;
        sum_wt = sum_wt + p[i].wt;
        sum_tt = sum_tt + p[i].tt;
        printf("\n#P[%d]\t|%d\t|\t    |%d\t  \t\t|%d", p[i].pid, p[i].bt, p[i].wt, p[i].tt);
        count++;
    }

    average_wt =(float) sum_wt / count;
    average_tt = (float) sum_tt / count;
    printf("\n\n#Average Waiting Time= %0.2f", average_wt);
    printf("\n#Average Turnaround Time= %0.2f\n", average_tt);
    clock_t endTime = clock();
    printf("\n##Total time= %f\n", (double) (endTime - startTime) / CLOCKS_PER_SEC);
    return 0;
}
```

مطابق دستور کار برای هر فرایند یک شماره، یک burst time، یک waiting time و یک turnaround time گرفتیم.

به تعداد دلخواه کاربر فرایند با burst time دلخواه کاربر میسازد.

به هر فرایند یک id میدهد.

بدترین زمان اجرای این الگوریتم $O(n^2)$ است به دلیل دو تا حلقه تو درتو در محاسبه waitingtime ها است.

Worst case time complexity : $O(n^2)$
Average case time complexity : $\theta(n^2)$
Best case time complexity : $\theta(n)$
Space complexity : $\theta(1)$

N برابر است با تعداد فرایندها.

زمان انتظار سایر فرآیندها که برابر است با زمان اجراي فرآيند قبل محاسبه میکند.زمان اجراي یک فرآیند برابر است با مجموع زمان انتظار و زمان سرویسدهی

مطابق دستور کار زمان انتظار برای فرایند اول صفر است.

محاسبه میانگین زمان turnaround time و waiting time

خروجی به صورت زیر است ابتدا تعداد فرایندهایی که داریم را به عنوان ورودی و سپس bursttime مورد نظر میدهیم که مشاهده میکنیم به ترتیب برابر است با 20،15،26،10، 14 است.
سپس مشاهده میکنیم زمان انتظار فرایندهای بعدی برابر است با زمان turnaround قبلی.
فرایند اول هیچ زمانی برای انتظار ندارد و ۲۰ ثانیه زمان نیاز دارد بنابراین turnaround = 20+0 میشود.
فرایند دوم اندازه turnaround قبلی که برابر ۲۰ است waiting time دارد و خود نیز نیاز به ۱۵ ثانیه دارد به همین علت turnaround تایمش برابر میشود با 35=20+15 و مشاهده میکنیم زمان انتظار برای فرایند سوم برابر است با turnaround فرایند دوم و به همین منوال ادامه میدهیم.
در نهایت میانگین زمان‌های انتظار و turnaround را بدست می‌آوریم و زمان اجرای برنامه را نیز محاسبه کردیم و مشاهده میکنیم عملکرد پایینی دارد، زیرا میانگین زمان انتظار بالا است.

```
helia@helia-virtual-machine:~/Desktop/OSLab/8$ gcc FCFS.c -o FCFS
helia@helia-virtual-machine:~/Desktop/OSLab/8$ ./FCFS
Number of process= 5

Process Burst Time
*P[1]= 20
*P[2]= 15
*P[3]= 26
*P[4]= 10
*P[5]= 14

Process |Burst Time         |Waiting Time       |Turnaround Time
_____________________________________________________________
#P[1]   |20                 |0                  |20
#P[2]   |15                 |20                 |35
#P[3]   |26                 |35                 |61
#P[4]   |10                 |61                 |71
#P[5]   |14                 |71                 |85

#Average Waiting Time= 37.40
#Average Turnaround Time= 54.40

##Total time= 0.000321
```

بخش دوم

در الگوریتم انتخاب کوتاه‌ترین فرایند، فرآیندی برای پردازش انتخاب می‌شود که به کوتاه‌ترین زمان پردازش نیاز داشته باشد.وقتی که چند کار با اهمیت یکسان برای اجرا شدن در صف ورودی قرار می‌گیرند، زمان‌بند باید **ابتدا کوتاه‌ترین کار** Shortest Jib First یا به اختصار SJF را انتخاب کند و یک الگوریتم زمان‌بندی غیر preemptive و pre-emptive است.

به این الگوریتم گاهی **الگوریتمSPN** مخفف (Shortest Process Next) نیز گفته می‌شود. الگوریتم زیر غیر preemptive است.

```c
#include<stdio.h>
#include "time.h"

struct process {
    int pid;
    int bt;
    int wt;
    int tt;
};

struct process p[20];


int main() {
    int i, j;
    int n;
    int sum_wt = 0, sum_tt = 0;
    int temp;
    float average_wt, average_tt;
    clock_t startTime=clock();
    printf("Number of process= ");
    scanf("%d", &n);

    printf("\nProcess Burst Time:\n");

    for (i = 0; i < n; i++) {
        printf("*P[%d]= ", i + 1);
        scanf("%d", &p[i].bt);
    }

    for (i = 0; i < n; i++) {
        p[i].pid = i+1;
    }

    //sort cpu burst time in ascending order
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].bt > p[j + 1].bt) {
                temp = p[j].bt;
                p[j].bt = p[j + 1].bt;
                p[j + 1].bt = temp;

                temp = p[j].pid;
                p[j].pid = p[j + 1].pid;
                p[j + 1].pid = temp;
            }
        }
    }

    p[0].wt = 0; //initial to zero

    //calculate waiting time
    for (i = 1; i < n; i++) {
        p[i].wt = 0;
        for (j = 0; j < i; j++) {
            p[i].wt = p[i].wt + p[j].bt;
        }
        sum_wt = sum_wt + p[i].wt;
    }


 printf("\nProcess\t|Burst Time\t    |Waiting Time\t|Turnaround
Time\n");

printf("_____________________________________________________________________');

    for (i = 0; i < n; i++) {
        p[i].tt = p[i].bt + p[i].wt;
        sum_tt = sum_tt + p[i].tt;
         printf("\n#P[%d]\t|%d\t\t    |%d\t  \t\t|%d", p[i].pid,
p[i].bt, p[i].wt, p[i].tt);
    }

    average_wt = (float) sum_wt / n;
    average_tt = (float) sum_tt / n;

    printf("\n\n#Average Waiting Time= %0.2f", average_wt);
    printf("\n#Average Turnaround Time= %0.2f\n", average_tt);
    clock_t endTime=clock();
    printf("\n##Total time= %f\n",(double)(endTime-startTime)/
CLOCKS_PER_SEC);
}
```

بدترین زمان اجرای این الگوریتم $O(n^2)$ است به دلیل دو تا حلقه تو درتو در مرتب‌سازی و محاسبه waitingtime ها است.

Worst case time complexity : $O(n^2)$
Average case time complexity : $\theta(n^2)$
Best case time complexity : $\theta(n)$
Space complexity : $\theta(1)$

N برابر است با تعداد فرایندها.

مطابق قسمت قبل برای هر فرایند یک شماره یک burst timeو یک waitingtimeو یک turnaroundtime درنظر گرفتیم.
به تعداد دلخواه فرایند کاربر با زمان bursttime دلخواه میسازد و به هرکدام یک id میدهد.
تفاوتش ان است که از سیاست fifo استفاده نمیکند و cpu ابتدا به کسی اختصاص داده میشود که زمان burst کمتری دارد.
بنابراین ابتدا با الگوریتم selection sort شروع به مرتب سازی صعودی میکند.
مطابق دستورکار زمان انتظار فرایند اول برابر صفر است . شروع له محاسبه waitingtime و turnaroundtime فرایند ها با استفاده از فرایند قبلی میکند و درنهایت میانگین waitingtime و turnaroundtime را بدست میاورد و زمان اجرای برنامه را نمایش میدهد.

مشاهده میکنیم که کاربر ۴ فرایند داده است با bursttime متفاوت با ترتیب های مختلف ۲،۶،۳،۲۱.
اولین فرایند داده شده طولانی‌ترین bursttime را دارد اما اخر از همه اجرا میشود به دلیل اینکه فرایندهایی که burstiime کمتری دارند سریعتر اجرا میشود و پردازنده فرایندها را به صورت صعودی مرتب میکند.
Waitingtime فرایند اول صفر است و busrttime ان ۲ به همین دلیل .turnaround=2+0میشود.
Waitingtime فرایند بعدی برابر است با turnaround فرایند قبلی و turnaroundجدید برابر با waitingtime+ bursttime=3+2=5 است و به همین منوال ادامه میدهیم.مشاهده میکنیم که میانگین waitingtime کاهش یافته است امادر سیستم‌های دسته‌ای که زمان مورد نیاز CPU از پیش مشخص نیست به سادگی پیاده‌سازی می‌شود وپردازنده باید از قبل بداند که پردازش چه مدت طول خواهد کشید.

```
helia@helia-virtual-machine:~/Desktop/OSLab/8$ gcc SJF.c -o SJF
helia@helia-virtual-machine:~/Desktop/OSLab/8$ ./SJF
Number of process= 4

Process Burst Time:
*P[1]= 21
*P[2]= 3
*P[3]= 6
*P[4]= 2

Process |Burst Time         |Waiting Time       |Turnaround Time
_____________________________________________________________
#P[4]   |2                  |0                  |2
#P[2]   |3                  |2                  |5
#P[3]   |6                  |5                  |11
#P[1]   |21                 |11                 |32

#Average Waiting Time= 4.50
#Average Turnaround Time= 12.50

##Total time= 0.000273
```

الگوریتم preemptive آن به صورت زیر است.
در اصل فرق که در دو قسمت وجود دارد این است که ما به arrival timeنیاز دازیم.

```c
#include<stdio.h>
#include "time.h"

struct process {
    int pid;
    int bt;
    int wt;
    int tt;
    int ar;
};

struct process p[20];

int main() {
    int tmp[20];
    int i;
    int cnt = 0;
    int n;
    float sum_wt = 0, sum_tt = 0;
    float average_wt, average_tt;
    clock_t startTime = clock();
    printf("Number of process= ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        printf("Arrival Time P[%d]= \t", i + 1);
        scanf("%d", &p[i].ar);
    }
    for (i = 0; i < n; i++) {
        printf("Process Burst Time P[%d]= \t", i + 1);
        scanf("%d", &p[i].bt);
        tmp[i] = p[i].bt;
    }

    for (i = 0; i < n; i++) {
        p[i].pid = i + 1;
    }

    int j;
    int index;
    p[19].bt = 10000000;
    int current_time = 0;

    while (cnt != n) {
        index = -1;
        for (i = 0; i < n; i++) {
            if (p[i].ar <= current_time && p[i].bt < p[index].bt) {
                if (p[i].bt > 0) {
                    index = i;
                }
            }
        }
        p[index].bt--;

        if (p[index].bt == 0) {
            cnt++;
            sum_tt +=  current_time + 1 - p[index].ar;
            sum_wt +=  sum_tt- tmp[index];
        }
        current_time++;
    }

    average_wt = (float) sum_wt / n;
    average_tt = (float) sum_tt / n;
    printf("\n\n#Average Waiting Time= %0.2f", average_wt);
    printf("\n#Average Turnaround Time= %0.2f", average_tt);
    clock_t endTime = clock();
    printf("\n##Total time= %f\n", (double) (endTime - startTime) /
CLOCKS_PER_SEC);
    return 0;
}
```

در این قسمت برای اینکه الگوریتم ما به صورت قبضه ای باشد یک whileتعریف کرده ایم که در آن بررسی های لازم انجام می شود در اصل در لوپ نخست شرط چک می شود و با توجه به آن مقدار index آپدیت می شود. سپس بعد از اتمام لوپ اول شرط بعدی چک می شود و اگر برقرار بود sum_wtو sum_tt را محاسبه می کنیم در اصل اینجا ما باید arrival وexit time را هم در نظر بگیریم. همانند قسمت timeقبل میانگین را محاسبه می کنیم.
در اصل برای محاسبه ی sum_wtو sum_ttاز روابط زیر کمک گرفته ایم.(در اصل tmp[index]ما درالگوریتم نوشته شده همان burst timeما می باشد که در ابتدا مقداردهی شده است)

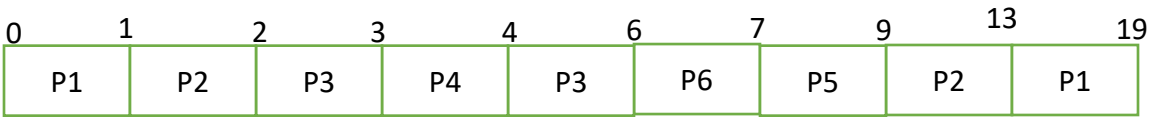
Turn Around time = Exit time – Arrival time

Waiting time = Turn Around time – Burst time

آنچه در خروجی نشان داده می شود به صورت زیر است:

```
helia@helia-virtual-machine:~/Desktop/OSLab/8$ ./SJFP
Number of process= 6

Process Arrival Time P[1]= 0

Process Arrival Time P[2]= 1

Process Arrival Time P[3]= 2

Process Arrival Time P[4]= 3

Process Arrival Time P[5]= 4

Process Arrival Time P[6]= 5

Process Burst Time P[1]= 7

Process Burst Time P[2]= 5

Process Burst Time P[3]= 3

Process Burst Time P[4]= 1

Process Burst Time P[5]= 2

Process Burst Time P[6]= 1


#Average Waiting Time= 4.00
#Average Turnaround Time= 7.17

##Total time= 0.000466
```

نمودار گانت:



Process Id	Exit time	Turn Around time	Waiting time
P1	19	19 – 0 = 19	19 – 7 = 12
P2	13	13 – 1 = 12	12 – 5 = 7
P3	6	6 – 2 = 4	4 – 3 = 1
P4	4	4 – 3 = 1	1 – 1 = 0
P5	9	9 – 4 = 5	5 – 2 = 3
P6	7	7 – 5 = 2	2 – 1 = 1

Average Turn Around time = (19 + 12 + 4 + 1 + 5 + 2) / 6 = 43 / 6 = 7.17

Average waiting time = (12 + 7 + 1 + 0 + 3 + 1) / 6 = 24 / 6 = 4

پیچیدگی زمانی آن $O(n^2)$ است.