گزارش آزمایش هشتم آزمایشگاه سیستم‌های عامل

اشکان شکیبا (۹۹۳۱۰۳۰)، علی هاشم‌پور (۹۹۳۱۰۸۲)

الگوریتم FCFS:

یک الگوریتم بر پایه سیاست FIFO که not 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;
    float average_wt, average_tt;
    int sum_wt = 0, sum_tt = 0;
    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     |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\t|%d\t\t\t    |%d\t  \t\t|\t%d", p[i].pid,
p[i].bt, p[i].wt, p[i].tt);
        count++;
    }

    average_wt = sum_wt / (float) count;
    average_tt = sum_tt / (float) count;
    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;
}
```

اگر تعداد پردازه‌ها را n در نظر بگیریم:

worst case time complexity: $O(n^2)$

average time complexity: $O(n^2)$

best case time complexity: $\theta(n)$

space complexity = $\theta(1)$

بعد از اجرای برنامه و تعیین burst time برنامه‌ها، می‌بینیم که زمان انتظار هر پردازه برابر با turnaround پردازه قبلی آن است. یعنی باید منتظر باشد تا پردازه قبلی کاملا به پایان رسیده و بعد اجرای خود را شروع کند.

خروجی برنامه:

```
ashkan@DESKTOP-B3PHD2K:/mnt/e/University/6/os/Lab/08$ gcc FCFS.c -o FCFS.out
ashkan@DESKTOP-B3PHD2K:/mnt/e/University/6/os/Lab/08$ ./FCFS.out
Number of process= 6

Process Burst Time
*P[1]:25
*P[2]:20
*P[3]:10
*P[4]:15
*P[5]:30
*P[6]:5

Process     |Burst Time     |Waiting Time       |Turnaround Time
_____________________________________________________________
#P[1]          |25                          |0                  |       25
#P[2]          |20                          |25                 |       45
#P[3]          |10                          |45                 |       55
#P[4]          |15                          |55                 |       70
#P[5]          |30                          |70                 |       100
#P[6]          |5                           |100                |       105

#Average Waiting Time= 49.17
#Average Turnaround Time= 66.67
##Total time= 0.015625
```

الگوریتم SJF:

در این الگوریتم بین پردازه‌های با اولویت یکسان، پردازه با زمان مورد انتظار کمتر انتخاب و اجرا می‌شود. این الگوریتم نیز not 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     |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\t|%d\t\t\t    |%d\t  \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);
}
```

اگر تعداد پردازه‌ها را n در نظر بگیریم:

worst case time complexity: $O(n^2)$

average time complexity: $O(n^2)$

best case time complexity: $\theta(n)$

space complexity = $\theta(1)$

در این زمان‌بند بعد از اجرای برنامه و تعیین burst time برنامه‌ها، می‌بینیم که پردازه دارای بیشترین burst time آخر از همه اجرا شده و زمان‌بند با مرتب‌سازی پردازه‌ها، آنهایی که زمان کمتری نیاز دارند را اول اجرا می‌کند. این الگوریتم می‌تواند منجر به مشکل starvation شود.

خروجی برنامه:

```
ashkan@DESKTOP-B3PHD2K:/mnt/e/University/6/os/Lab/08$ gcc sjf.c -o sjf.out
ashkan@DESKTOP-B3PHD2K:/mnt/e/University/6/os/Lab/08$ ./sjf.out
Number of process= 6

Process Burst Time:
*P[1]:25
*P[2]:20
*P[3]:10
*P[4]:15
*P[5]:30
*P[6]:5

Process     |Burst Time     |Waiting Time       |Turnaround Time
_____________________________________________________________
#P[6]          |5                          |0                  |       5
#P[3]          |10                         |5                  |       15
#P[4]          |15                         |15                 |       30
#P[2]          |20                         |30                 |       50
#P[1]          |25                         |50                 |       75
#P[5]          |30                         |75                 |       105

#Average Waiting Time= 29.17
#Average Turnaround Time= 46.67
```

الگوریتم sjf preemptive:

تفاوت آن با الگوریتم قبل در این است که هر زمان در هنگام اجرای پردازه‌ای، پردازه جدیدی برسد که burst time آن از باقیمانده burst time پردازه فعلی کمتر باشد، اجرای پردازه کنونی متوقف شده و پردازه جدید جایگزین آن می‌شود. واضح است که این الگوریتم preemptive است.

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

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

struct process p[20];
```

```c
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 = 20-1;
        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++;
            j = current_time + 1;
            sum_wt +=  j - p[index].ar - tmp[index];
            sum_tt +=  j - p[index].ar;
        }
        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;
}
```

خروجی برنامه:

```
ashkan@DESKTOP-B3PHD2K:/mnt/e/University/6/os/Lab/08$ gcc sjf_preemtive.c -o sjf_preemptive.out
ashkan@DESKTOP-B3PHD2K:/mnt/e/University/6/os/Lab/08$ ./sjf_preemptive.out
Number of process= 6
Arrival Time P[1]=      3
Arrival Time P[2]=      1
Arrival Time P[3]=      0
Arrival Time P[4]=      2
Arrival Time P[5]=      1
Arrival Time P[6]=      8
Process Burst Time P[1]=        25
Process Burst Time P[2]=        20
Process Burst Time P[3]=        10
Process Burst Time P[4]=        15
Process Burst Time P[5]=        30
Process Burst Time P[6]=        5


#Average Waiting Time= 27.50
#Average Turnaround Time= 45.00
```

الگوریتم round robin:

الگوریتمی که به هر پردازه تنها به اندازه کوانتوم زمانی فرصت داده می‌شود و پس از پایان آن به صف انتظار فرستاده می‌شود تا دوباره نوبتش شود. این الگوریتم preemptive است.

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

struct process {
    int pid;
    int bt;
    int wt;
    int tt;
    int ar;
};
struct process p[100];
int l, r, TQ, n, cur;

int main() {
```

```
    int sum_wt = 0, sum_tt = 0;
    int burstTime_tmp[20];
    float average_wt, average_tt;
    clock_t startTime = clock();
    printf("\nNumber of Processes:\t");
    scanf("%d", &n);
    printf("\n Time Quantum= ");
    scanf("%d", &TQ);

    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
    }
    for (int i = 0; i < n; i++) {
        printf("Burst Time P[%d]= ", p[i].pid);
        scanf("%d", &p[i].bt);
        burstTime_tmp[r++] = i;
        p[i].wt = -p[i].bt;
    }

    for (; l < r; l++) {
        int i = burstTime_tmp[l];
        if (p[i].bt > TQ) {
            cur += TQ;
            p[i].bt -= TQ;
            burstTime_tmp[r++] = i;
        } else {
            cur += p[i].bt;
            p[i].tt = cur;
            p[i].wt += p[i].tt;
        }
    }

    for (int i = 0; i < n; ++i) {
        printf("P[%d] => waiting time %d.\n", p[i].pid, p[i].wt);
        printf("P[%d] => turn around time %d.\n", p[i].pid, p[i].tt);
        sum_tt += p[i].tt;
        sum_wt += p[i].wt;
    }


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

خروجی برنامه:

```
ashkan@DESKTOP-B3PHD2K:/mnt/e/University/6/os/Lab/08$ ./roundrobin.out

Number of Processes:    6

 Time Quantum= 3
Burst Time P[1]= 25
Burst Time P[2]= 20
Burst Time P[3]= 10
Burst Time P[4]= 15
Burst Time P[5]= 30
Burst Time P[6]= 5
P[1] => waiting time 74.
P[1] => turn around time 99.
P[2] => waiting time 69.
P[2] => turn around time 89.
P[3] => waiting time 47.
P[3] => turn around time 57.
P[4] => waiting time 57.
P[4] => turn around time 72.
P[5] => waiting time 75.
P[5] => turn around time 105.
P[6] => waiting time 30.
P[6] => turn around time 35.


Average Waiting Time:   58.67
Avg Turnaround Time:    76.17
```

الگوریتم priority:

این الگوریتم پردازه‌ها را با توجه به اولویت آنها انتخاب و اجرا می‌کند.

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

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

struct process p[20];
```

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

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

    for (i = 0; i < n; i++) {
        printf("Burst Time P[%d]", p[i].pid);
        scanf("%d", &p[i].bt);
        printf("Priority P[%d]", p[i].pid);
        scanf("%d", &p[i].pr);
    }

    //sort based on the priority
    for (i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (p[i].pr < p[j].pr) {
                temp = p[i].pr;
                p[i].pr = p[j].pr;
                p[j].pr = temp;

                temp = p[i].bt;
                p[i].bt = p[j].bt;
                p[j].bt = temp;

                temp = p[i].pid;
                p[i].pid = p[j].pid;
                p[j].pid = temp;
            }
        }
    }

    p[0].wt = 0; //initialized into zero
    p[0].tt = p[0].bt;
    sum_tt = p[0].tt;

    for (i = 1; i < n; i++) {
        p[i].wt = p[i - 1].tt;
        sum_wt = sum_wt+ p[i].wt;
        p[i].tt = p[i].wt + p[i].bt;
        sum_tt = sum_tt+ p[i].tt;
    }

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

printf("_____________________________________________________________");
```

```
    for (i = 0; i < n; i++) {
        printf("\n#P[%d]\t\t|%d\t\t\t    |%d\t  \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 Average Wait Time : %0.2f \n", average_wt);
    printf("\n Average Turn Around Time : %0.2f \n", average_tt);
    clock_t endTime=clock();
    printf("\n##Total time= %f\n",(double)(endTime-
startTime)/CLOCKS_PER_SEC);
}
```

خروجی برنامه:

```
ashkan@DESKTOP-B3PHD2K:/mnt/e/University/6/os/Lab/08$ gcc priority.c -o priority.out
ashkan@DESKTOP-B3PHD2K:/mnt/e/University/6/os/Lab/08$ ./priority.out
Number of process= 6
Burst Time P[1]25
Priority P[1]4
Burst Time P[2]20
Priority P[2]3
Burst Time P[3]10
Priority P[3]1
Burst Time P[4]15
Priority P[4]6
Burst Time P[5]39
Priority P[5]2
Burst Time P[6]5
Priority P[6]5

Process     |Burst Time     |Waiting Time       |Turnaround Time
_____________________________________________________________
#P[4]           |15                         |0                  |       15
#P[6]           |5                          |15                 |       20
#P[1]           |25                         |20                 |       45
#P[2]           |20                         |45                 |       65
#P[5]           |39                         |65                 |       104
#P[3]           |10                         |104                        |       114
 Average Wait Time : 41.50

 Average Turn Around Time : 60.50
```