به نام خدا

محمد جواد زندیه 9831032

گزارش آزمایش شماره 8 سیستم عامل

در این آزمایش باید الگوریتم های اختصاص cpu به پردازه ها را پیاده سازی کنیم که 4 الگوریتم زیر خواسته شده است:

1. FCFS
2. SJF
3. PriorityQueue
4. RoundRobin

در این کد ابتدا داده های اولیه مورد نیاز یعنی تعداد پردازه ها و اینکه هر یک از آنها چه cpu burst ای دارند از کاربر گرفته می شود و با توجه به اینکه چه الگوریتم ای را بخواهیم استفاده کنیم اطلاعات بعدی گرفته خواهد شد. در FCFS نیاز است که زمان ورود هر یک از پردازه ها به منظور تخصیص cpu به آنها را از کاربر بگیریم و در PriorityQueue هم نیاز است تا اولویت هر یک از پردازه ها را بدانیم که از کاربر گرفته می شود.

شمای کلی این کد به این صورت هستند:

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct process{…
}process;

int n; // number of processes
int i, j; // used for iterations
int total_burst; // total cpu burst of processes

/* compare two processes  by arrival time */
int compare0(const void* process1, const void* process2){…

/* compare two processes  by priority */
int compare1(const void* process1, const void* process2){…

/* compare two processes  by remaining time */
int compare2(const void* process1, const void* process2){…
```

```
 39  /* print all processes info */
 40 > void print_processes_info(process PCB[]){ ⋯
 47
 48  /* first come first serve policy */
 49 > void FCFS(process PCB[]){ ⋯
 70
 71  /* shortest job first policy */
 72 > void SJF(process PCB[]){ ⋯
 87
 88  /* priority queue policy */
 89 > void PriorityQueue(process PCB[]){ ⋯
110
111  /* round robin policy */
112 > void RoundRobin(process PCB[]){ ⋯
151
152
153 > int main(int argc, char const *argv[]){ ⋯
```

cpu_policy.c

cpu_policy.c > RoundRobin(process [])

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct process{
    int pid;
    int priority;
    int at; // arrival time
    int bt; // burst time
    int rmt; // remaining time
    int wt; // waiting time
    int tat; // turnaround time
}process;

int n; // number of processes
int i, j; // used for iterations
int total_burst; // total cpu burst of processes
```

```c
/* compare two processes  by arrival time */
int compare0(const void* process1, const void* process2){
    process* p1 = (process*) process1;
    process* p2 = (process*) process2;
    return p2->at < p1->at;
}

/* compare two processes  by priority */
int compare1(const void* process1, const void* process2){
    process* p1 = (process*) process1;
    process* p2 = (process*) process2;
    return p1->priority > p2->priority;
}

/* compare two processes  by remaining time */
int compare2(const void* process1, const void* process2){
    process* p1 = (process*) process1;
    process* p2 = (process*) process2;
    return p1->rmt > p2->rmt;
}
```

```c
/* print all processes info */
void print_processes_info(process PCB[]){
    printf("\n%s%9s%10s%7s%6s%12s", "pid", "arrival", "priority", "burst", "wait", "turnarround");
    for(i = 0; i < n; i++)
        printf("\n%d%8d%10d%8d%7d%8d\n",
        PCB[i].pid, PCB[i].at, PCB[i].priority, PCB[i].bt, PCB[i].wt, PCB[i].tat);
    printf("\n");
}
```

```c
/* first come first serve policy */
void FCFS(process PCB[]){
    printf("\narrival time of processes: ");
    for(i = 0; i < n; i++){
        int arrival;
        scanf("%d", &arrival);
        PCB[i].at = arrival;
    }

    int time = 0;

    qsort(PCB, n, sizeof(struct process), compare0);

    for(i = 0; i < n; i++){
        PCB[i].wt = time;
        PCB[i].tat = PCB[i].wt + PCB[i].bt;
        PCB[i].rmt = 0;
        time += PCB[i].bt;
    }
    printf("\nFCFS:\n");
    print_processes_info(PCB);
}
```

```c
/* shortest job first policy */
void SJF(process PCB[]){
    int time = 0;

    qsort(PCB, n, sizeof(struct process), compare2);

    for(i = 0; i < n; i++){
        PCB[i].at = 0;
        PCB[i].wt = time;
        PCB[i].tat = PCB[i].wt + PCB[i].bt;
        PCB[i].rmt = 0;
        time += PCB[i].bt;
    }
    printf("\nSJF:\n");
    print_processes_info(PCB);
}
```

```c
/* priority queue policy */
void PriorityQueue(process PCB[]){
    printf("\npriority of processes: ");
    for(i = 0; i < n; i++){
        int priority;
        scanf("%d", &priority);
        PCB[i].priority = priority;
    }

    int time = 0;

    qsort(PCB, n, sizeof(struct process), compare1);

    for(i = 0; i < n; i++){
        PCB[i].wt = time;
        PCB[i].tat = PCB[i].wt + PCB[i].bt;
        PCB[i].rmt = 0;
        time += PCB[i].bt;
    }
    printf("\nPriorityQueue:\n");
    print_processes_info(PCB);
}
```

```c
/* round robin policy */
void RoundRobin(process PCB[]){
    int q = 1;
    printf("\ntime quantum: ");
    scanf("%d", &q);

    int time = 0;
    i = 0;

    while(total_burst != 0){
        if(PCB[i].rmt >= q){
            j = (i+1)%n;
            while(j != i){
                if(PCB[j].rmt != 0)
                    PCB[j].wt += q;
                j = (j+1)%n;
            }
            PCB[i].rmt -= q;
            total_burst -= q;
            i = (i+1)%n;
```

```
        }else if(PCB[i].rmt > 0){
            j = (i+1)%n;
            while(j != i){
                if(PCB[j].rmt != 0)
                    PCB[j].wt += PCB[i].rmt;
                j = (j+1)%n;
            }
            total_burst -= PCB[i].rmt;
            PCB[i].rmt = 0;
            i = (i+1)%n;
        }else if(PCB[i].rmt == 0){
            i = (i+1)%n;
        }
    }

    for(i = 0; i < n; i++)
        PCB[i].tat = PCB[i].wt + PCB[i].bt;

    print_processes_info(PCB);
}
```

```
int main(int argc, char const *argv[]){
    printf("number of processes: ");
    scanf("%d", &n);

    process PCB[n];
    total_burst = 0;

    printf("\nburst time of processes: ");
    for(i = 0; i < n; i++){
        int burst;
        scanf("%d", &burst);
        total_burst += burst;
        PCB[i].pid = i;
        PCB[i].wt = 0;
        PCB[i].tat = 0;
        PCB[i].at = 0;
        PCB[i].priority = 0;
        PCB[i].bt = burst;
        PCB[i].rmt = burst;
    }

    FCFS(PCB);
    SJF(PCB);
    PriorityQueue(PCB);
    RoundRobin(PCB);

    return 0;
```

خروجی به ازای هر یک از الگوریتم ها:

```
javad@javad-HP-350-G1:~/Desktop/OSLab/project8$ gcc cpu_policy.c -o cpu_policy
javad@javad-HP-350-G1:~/Desktop/OSLab/project8$ ./cpu_policy
number of processes: 5

burst time of processes: 1 4 3 5 3

arrival time of processes: 1 0 4 3 4

FCFS:

pid  arrival  priority  burst  wait turnarround
1       0         0       4      0       4

0       1         0       1      4       5

3       3         0       5      5      10

2       4         0       3     10      13

4       4         0       3     13      16

javad@javad-HP-350-G1:~/Desktop/OSLab/project8$
```

```
javad@javad-HP-350-G1:~/Desktop/OSLab/project8$ gcc cpu_policy.c -o cpu_policy
javad@javad-HP-350-G1:~/Desktop/OSLab/project8$ ./cpu_policy
number of processes: 5

burst time of processes: 1 4 3 6 3

SJF:

pid  arrival  priority  burst  wait turnarround
0       0         0       1      0       1

2       0         0       3      1       4

4       0         0       3      4       7

1       0         0       4      7      11

3       0         0       6     11      17

javad@javad-HP-350-G1:~/Desktop/OSLab/project8$
```

```
javad@javad-HP-350-G1:~/Desktop/OSLab/project8$ ./cpu_policy
number of processes: 6

burst time of processes: 1 3 12 5 7 3

priority of processes: 1 2 6 3 2 4

PriorityQueue:

pid  arrival  priority  burst  wait turnarround
0       0         1       1      0       1

1       0         2       3      1       4

4       0         2       7      4      11

3       0         3       5     11      16

5       0         4       3     16      19

2       0         6      12     19      31

javad@javad-HP-350-G1:~/Desktop/OSLab/project8$
```

```
javad@javad-HP-350-G1:~/Desktop/OSLab/project8$ gcc cpu_policy.c -o cpu_policy
javad@javad-HP-350-G1:~/Desktop/OSLab/project8$ ./cpu_policy
number of processes: 7

burst time of processes: 1 4 12 5 17 3 4

time quantum: 3

RoundRobin:

pid  arrival  priority  burst  wait turnarround
0       0         0       1      0       1

1       0         0       4     16      20

2       0         0      12     26      38

3       0         0       5     20      25

4       0         0      17     29      46

5       0         0       3     13      16

6       0         0       4     25      29

javad@javad-HP-350-G1:~/Desktop/OSLab/project8$
```

بخش پنجم: برای تعداد فرآیندهای به اندازه کافی بزرگ، روش‌های پیاده‌سازی شده در قبل را در قالب جدول بر اساس مشخصه‌های الگوریتم‌های زمانبند، مقایسه کنید و برای هریک دلیل بیاورید که در چه کاربردی مناسب و در چه کاربردی نامناسب است.

Algorithm	Time complexity	Usage
FCFS	$O(n\ lgn)$	در مواقعی که cpu burst ها کوتاه هستند مفید است اما اگر طولانی باشد می تواند باعث ایجاد Starvation بشود و همچنین ممکن است اثر کاروان داشته باشد Convoy effect
SJF	$O(n\ lgn)$	در کل میانگین waiting time پردازه ها را پایین می آورد اما Response time برای پردازه ها بالا می آورد و می تواند باعث این شود که سیستم responsive نباشد
Priority Queue	$O(n\ lgn)$	استفاده از این سیاست در صف انتظار می تواند این خوبی را داشته باشد که پردازه های با اولویت تر زود تر بتوانند cpu را بدست آوردند اما ممکن است باعث ایجاد قحطی شده و برخی پردازه ها با اولویت پایین هرگز نوبت اجرا پیدا نکنند
Round Robin	$O(n^2)$	این روش برای زمانی که بخواهیم response time برای پردازه ها پایین بیاید خوب است و سیستم ما خیلی responsive تر خواهد بود اما هزینه context switch اگر time quantum به درستی تعیین نشده باشد کاملا سرباری برای زمان کل می باشد.