دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش کار آزمایشگاه سیستم عامل – شماره ۸ بخش دوم

رفع عیب الگوریتم‌های زمانبندی

حسنا اویارحسینی – ۹۸۲۳۰۱۰

استاد درس: جناب آقای مهندس کیخا

نیمسال دوم سال تحصیلی ۰۱-۱۴۰۰

<u>بخش ۱</u>) با در نظر گرفتن نقاط قوت و ضعف چهار الگوریتم مورد بررسی در آزمایش هشتم، هریک از الگوریتم های معرفی شده را رفع عیب نموده و گزارش دهید ایراد الگوریتم را به چه صورت استفاده کرده اید.

۱- <u>بررسی الگوریتم priority</u>:

مشکل الگوریتم priority قحطی می باشد به این معنی که ممکن است یک پردازه با اولویت پایین پشت پردازه ها با اولویت بالا برای مدت خیلی طولانی گیر کند. برای حل این مشکل از الگوریتم multilevel queue استفاده میکنیم به این معنی که اولویت پردازه ها را تغییر میدهیم در واقع به نوعی aging صورت میگرد و اگر یک پردازه با اولویت بالا زمان زیادی میگیرد اولویت آن به مرور زمان کم میشود در نتیجه جلوی پردازه ها با اولویت کم را نمیگیرد:

<u>کد:</u>

در این کد فرض کردیم که سه لول صف داریم که به ترتیب الگوریتم های RR با کوانتوم ۸، RR با کوانتوم ۱۶ و FCFS را اجرا میکنند. سه آرایه برای این سه صف داریم در واقع هر صف برای پردازه های مربوط به یک اولویت می باشد (صف اول برای پردازه هایی با اولویت ۰، صف دوم برای پردازه هایی با اولویت ۱ و ...). در اینجا برای سادگی فرض کرده ایم اولویت و زمان ورود همه پردازه ها یکسان است (در حالت واقعی پردازه ها با اولویت های متفاوت میتواند در وسط کار اضافه شده و باید در صف مربوط به اولویت خود اضافه شوند). هر پردازه با یک متغیر از نوع struct process می باشد که شامل فیلدهای زیر است:

متغیر	توضیح
AT	Arrival time
BT	Burst time
WT	Waiting time
TAT	Turnaround time
RT	Remaining time

سپس از کاربر تعداد و burst time هر پردازه را میگیریم.

ابتدا همه پردازه ها وارد صف Q۱ میشوند و الگوریتم RR با کوانتوم ۸ که در آزمایش ۸-بخش اول توضیح داده شده بود روی آنها اجرا میشود. اگر کار پردازه تمام شد که حذف میشود اما در غیر این صورت زمان باقی مانده این پردازه آپدیت و به صف دوم منتقل میشود این کار تا زمانی ادامه میابد که تمام پردازه ها یکبار در صف اول اجرا شوند. سپس نوبت به صف دوم میرسد و پردازه های موجود در آن به ترتیب با الگوریتم RR با کوانتوم ۱۶ اجرا میشوند و پردازه هایی که همچنان تمام نمیشوند وارد صف سوم میشوند. پس از اتمام صف دو پردازه های موجود در صف ۳ به ترتیب اجرا میشوند تا تمام شوند. در هر صف پس از اتمام صف وضعیت هر پردازه و زمان باقی مانده آن چاپ میشود و در نهایت زمان اجرا و زمان انتظار نیز اعلام میشود:

```
struct process
{
    int name;
    int AT,BT,WT,TAT,RT,CT;
};
/*Three queues*/
struct process * Q١;
struct process * Q٢;
struct process * Q٣;

int n;

int main()
{
     int total_WT=٠, total_TAT=٠;
     int i,j,k=٠,r=٠,time=٠,tq١=٨,tq٢=١٦,flag=٠;
     char c;
     printf("Enter no of processes:");
     scanf("%d",&n);
     Q١ = (struct process*) malloc(n*sizeof(struct process));
     Q٢ = (struct process*) malloc(n*sizeof(struct process));
     Q٣ = (struct process*) malloc(n*sizeof(struct process));

     for(i=٠;i<n;i++)
     {
         Q١[i].name= i + ١;
         printf("\nEnter the arrival time and burst time of process %d:
",Q١[i].name);
         scanf("%d%d",&Q١[i].AT,&Q١[i].BT);
         Q١[i].RT=Q١[i].BT;/*save burst time in remaining time for each process*/

    }
time=Q١[٠].AT;
printf("Process in first queue following RR with qt=٨");
printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
for(i=٠;i<n;i++)
{

  if(Q١[i].RT<=tq١)
  {

       time+=Q١[i].RT;/*from arrival time of first process to completion of this
process*/
       Q١[i].RT=٠;
```

اجرای الگوریتم صف اول

RR, q = ٨

```
        Q١[i].WT=time-Q١[i].AT-Q١[i].BT;/*amount of time process has been waiting
in the first queue*/
        Q١[i].TAT=time-Q١[i].AT;/*amount of time to execute the process*/
        printf("\n%d\t\t%d\t\t%d\t\t%d",Q١[i].name,Q١[i].RT,Q١[i].WT,Q١[i].TAT);
        total_TAT += Q١[i].TAT;
        total_WT += Q١[i].WT;

   }
   else/*process moves to queue ٢ with qt=١٦*/
   {
       Q٢[k].WT=time;
       time+=tq١;
       Q١[i].RT-=tq١;
       Q٢[k].BT=Q١[i].RT;
       Q٢[k].RT=Q٢[k].BT;
       Q٢[k].name=Q١[i].name;
       flag=١;
       printf("\n%d\t\t%d\t\t%d\t\t%d",Q٢[k].name,Q٢[k].BT,Q٢[k].WT,Q٢[k].TAT);
       k=k+١;


    }
```

ادامه اجرای الگوریتم صف اول

RR, q = ٨

```
}
if(flag==١)
 {printf("\nProcess in second queue following RR with qt=١٦");
  printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
}for(i=٠;i<k;i++)
   {
    if(Q٢[i].RT<=tq٢)
     {
       time+=Q٢[i].RT;/*from arrival time of first process +BT of this process*/
       Q٢[i].RT=٠;
       Q٢[i].WT=time-tq١-Q٢[i].BT;/*amount of time process has been waiting in
the ready queue*/
       Q٢[i].TAT=time-Q٢[i].AT;/*amount of time to execute the process*/
       printf("\n%d\t\t%d\t\t%d\t\t%d",Q٢[i].name,Q٢[i].RT,Q٢[i].WT,Q٢[i].TAT);
       total_TAT += Q٢[i].TAT;
       total_WT += Q٢[i].WT;

    }
    else/*process moves to queue ٣ with FCFS*/
    {
      Q٣[r].AT=time;
      time+=tq٢;
```

ااجرای الگوریتم صف دوم

RR, q = ١٦

```
      Q۲[i].RT-=tq۲;
      Q۳[r].BT=Q۲[i].RT;
      Q۳[r].RT=Q۳[r].BT;
      Q۳[r].name=Q۲ [i].name;
      flag=۲;
       printf("\n%d\t\t%d\t\t%d\t\t%d",Q۳[r].name,Q۳[r].BT,Q۳[r].WT,Q۳[r].TAT);
      r=r+۱;


    }

  }
```

ادامه اجرای الگوریتم صف دوم

RR, q = ۱۶

```
{if(flag==۲)
printf("\nProcess in third queue following FCFS ");
}
for(i=۰;i<r;i++)
{
    if(i==۰)
            Q۳[i].CT=Q۳[i].BT+time;
        else
            Q۳[i].CT=Q۳[i-۱].CT+Q۳[i].BT;

}

for(i=۰;i<r;i++)
    {
        Q۳[i].RT=۰;
        Q۳[i].TAT=Q۳[i].CT;
        Q۳[i].WT=Q۳[i].TAT-Q۳[i].BT-tq۱-tq۲;
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t",Q۳[i].name,Q۳[i].RT,Q۳[i].WT,Q۳[i].TA
T);
        total_TAT += Q۳[i].TAT;
        total_WT += Q۳[i].WT;

    }
printf("\n");
printf("Average waiting time = %f", (float)total_WT / (float)n);
printf("\nAverage turn around time = %f\n", (float)total_TAT / (float)n);

}
```

اجرای الگوریتم صف سوم

FCFS

خروجی کد:

```
lab@ubuntu:~/Desktop/HW8 2$ gcc MLFQ.c -o MLFQ
lab@ubuntu:~/Desktop/HW8 2$ ./MLFQ
Enter no of processes:6

Enter the arrival time and burst time of process 1: 0 3

Enter the arrival time and burst time of process 2: 0 8

Enter the arrival time and burst time of process 3: 0 12

Enter the arrival time and burst time of process 4: 0 20

Enter the arrival time and burst time of process 5: 0 25

Enter the arrival time and burst time of process 6: 0 35
Process in first queue following RR with qt=8
Process         RT              WT              TAT
1               0               0               3
2               0               3               11
3               4               11              0
4               12              19              0
5               17              27              0
6               27              35              0
Process in second queue following RR with qt=16
Process         RT              WT              TAT
3               0               35              47
4               0               39              59
5               1               0               0
6               11              0               0
Process in third queue following FCFS
5               0               67              92
6               0               68              103
Average waiting time = 35.333332
Average turn around time = 52.500000
lab@ubuntu:~/Desktop/HW8 2$
```

۲- بررسی الگوریتم RR:

مشکل الگوریتم RR این است که اولویت پردازه ها را در نظر نمیگیرد و مثلا نمیتواند از Non makeable interrupt ها پشتیبانی کند برای حل نسبی (نه کامل) این مشکل دو را وجود دارد:

۱- به هر پردازه با توجه به اولویت آن time quantum اختصاص دهیم یعنی پردازه با اولویت بیشتر کوانتوم زمانی بزرگتری داشته باشد تا زودتر تمام شود.

۲- مرتب سازی پردازه ها در صف اصلی بر اساس اولویت باشد، و نه بر اساس FCFS که باعث میشود اولویت ها نیز تا حدی لحاظ شوند.

در ادامه کد مربوط به راه حل دوم را بررسی میکنیم:

کد:

این کد همانند کد RR بخش یک آزمایش ۸ است با این تفاوت که با ورود هر پردازه جدید یکبار پردازه ها را براساس اولویت مرتب میکنیم (در اینجا برای سادگی فرض کرده ایم اولویت و زمان ورود همه پردازه ها یکسان است بخاطر همین فقط یکبار در ابتدا مرتب سازی کرده ایم).

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

struct Process{

  int pid;
  int bt;
```

```
  int wt;
  int tt;
  int at; //arrival time
  int nt; //needed time
  int priority;
};

struct Process * p;

int main()
{

  int n = ٠; //total number of process
  p = (struct Process*) malloc(n*sizeof(struct Process));
  int i =٠;
  int flag = ٠;
  int time = ٠;
  int remaining_processes = ٠;
  int quantum = ٠;
  int total_wt = ٠;
  int total_tat = ٠;


  printf("Enter number of Processes:");
  scanf("%d",&n);
  p = (struct Process*) malloc(n*sizeof(struct Process));

  remaining_processes = n;

  for(int i  = ١; i <= n; i++){
    p[i-١].pid = i;
    printf("P%d:\n", i);
    printf("Burst Time: ");
    scanf("%d",&p[i-١].bt);
    printf("Priority: ");
    scanf("%d",&p[i-١].priority);
    p[i-١].wt = ٠;
    p[i-١].tt = ٠;
    p[i-١].nt = p[i-١].bt;
    p[i-١].at = ٠;

  }

  printf("Enter the Time Quantum(q):");
  scanf("%d",&quantum);
```

```c
printf("\nProcess \t Turnaround Time \t Waiting Time\n\n");

time = ۰;
i = ۰;

 // sort by priority
  for (int k = ۰; k < n; ++k)
      for (int j = k; j < n; ++j){
          if (p[k].priority > p[j].priority)
              swap(k,j);
          else if (p[k].priority == p[j].priority && p[k].bt < p[j].bt)
              swap(k,j);
      }
```

مرتب سازی بر اساس اولویت

```c
while(remaining_processes > ۰)
{

  if(p[i].nt <= quantum && p[i].nt > ۰)
  {
    time += p[i].nt;
    p[i].nt = ۰;
    flag = ۱;
  }
  else if(p[i].nt > ۰)
  {
    p[i].nt -= quantum;
    time += quantum;
    printf("P%d\n", i + ۱);
  }
  if(p[i].nt == ۰ && flag == ۱)
  {
    remaining_processes--;
    p[i].tt = time - p[i].at;
    p[i].wt = time - p[i].at - p[i].bt;
    total_wt += p[i].wt;
    total_tat += p[i].tt;

    printf("P%d\t\t\t%d\t\t%d\n", p[i].pid, p[i].tt, p[i].wt);
    flag=۰;
  }

  if(i == n - ۱)
    i = ۰;

  else if(p[i+۱].at <= time)
```

```
      i ++;

    else
      i = ۰;
  }

    printf("Average waiting time = %f", (float)total_wt / (float)n);
  printf("\nAverage turn around time = %f\n", (float)total_tat / (float)n);

  return ۰;
}
```

خروجی کد:

این مثال همان مثالی است که در بخش اول آزمایش هشت نیز امتحان شد که مشاهده میشود اینجا اولویت هم لحاظ شده است.

```
lab@ubuntu:~/Desktop/HW8 2$ gcc RRP.c -o RRP
lab@ubuntu:~/Desktop/HW8 2$ ./RRP
Enter number of Processes:5
P1:
Burst Time: 10
Priority: 3
P2:
Burst Time: 1
Priority: 1
P3:
Burst Time: 2
Priority: 4
P4:
Burst Time: 1
Priority: 5
P5:
Burst Time: 2
Priority: 5
Enter the Time Quantum(q):2

Process          Turnaround Time         Waiting Time

P2                     1                0
P1
P3                     5                3
P5                     7                5
P4                     8                7
P1
P1
P1
P1                     16               6
Average waiting time = 4.200000
Average turn around time = 7.400000
lab@ubuntu:~/Desktop/HW8 2$
```

۱- بررسی الگوریتم SJF , FCFS:

- مشکل الگوریتم FCFS پدیده کاروان است به این معنی که ممکن است یک پردازه های با CPU burst بالا ابتدا وارد شود و در نتیجه پردازهایی با CPU burst پایین تر بعد از آن بیایند و پشت آن گیر کنند که این موضوع باعث میشود waiting time به طور متوسط بالا رود.
- اشکال الگوریتم SJF آن است که اگر تعداد پردازه ها با CPU burst پایین خیلی زیاد شود نوبت به پردازه هایی با CPU burst بالا نمیرسد و هیچ وقت CPU به آنها تخصیص نمی یابد.

برای حل مشکلات این الگوریتم میتوان از الگوریتم های دیگر مثل دو الگوریتم بالا استفاده کرد زیرا اگر بخواهیم این الگوریتم ها را برای رفع ایرادشان تغییر بدهیم الگوریتم ماهیتش را از دست میدهد.