Exp. No. 5a FCFS Scheduling

Date:

Aim

To schedule snapshot of processes queued according to FCFS scheduling.

Process Scheduling

- CPU scheduling is used in multiprogrammed operating systems.
- By switching CPU among processes, efficiency of the system can be improved.
- Some scheduling algorithms are FCFS, SJF, Priority, Round-Robin, etc.
- Gantt chart provides a way of visualizing CPU scheduling and enables to understand better.

First Come First Serve (FCFS)

- Process that comes first is processed first
- FCFS scheduling is non-preemptive
- Not efficient as it results in long average waiting time.
- Can result in starvation, if processes at beginning of the queue have long bursts.

Algorithm

1. Define an array of structure *process* with members *pid*, *btime*, *wtime* & *ttime*.
2. Get length of the ready queue, i.e., number of process (say *n*)
3. Obtain *btime* for each process.
4. The *wtime* for first process is 0.
5. Compute *wtime* and *ttime* for each process as:
   a. $wtime_{i+1} = wtime_i + btime_i$
   b. $ttime_i = wtime_i + btime_i$
6. Compute average waiting time *awat* and average turnaround time *atur*
7. Display the *btime*, *ttime* and *wtime* for each process.
8. Display GANTT chart for the above scheduling
9. Display *awat* time and *atur*
10. Stop

Program

```
/* FCFS Scheduling  - fcfs.c */

#include <stdio.h>

struct process
{
   int pid;
   int btime;
   int wtime;
   int ttime;
} p[10];

main()
{
   int i,j,k,n,ttur,twat;
   float awat,atur;

   printf("Enter no. of process : ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
   {
      printf("Burst time for process P%d (in ms) : ",(i+1));
      scanf("%d", &p[i].btime);
      p[i].pid = i+1;
   }

   p[0].wtime = 0;
   for(i=0; i<n; i++)
   {
      p[i+1].wtime = p[i].wtime + p[i].btime;
      p[i].ttime = p[i].wtime + p[i].btime;
   }
   ttur = twat = 0;
   for(i=0; i<n; i++)
   {
      ttur += p[i].ttime;
      twat += p[i].wtime;
   }
   awat = (float)twat / n;
   atur = (float)ttur / n;

   printf("\n       FCFS Scheduling\n\n");
   for(i=0; i<28; i++)
      printf("-");
   printf("\nProcess B-Time T-Time W-Time\n");
   for(i=0; i<28; i++)
      printf("-");
```

```
    for(i=0; i<n; i++)
       printf("\n  P%d\t%4d\t%3d\t%2d",
                p[i].pid,p[i].btime,p[i].ttime,p[i].wtime);
    printf("\n");
    for(i=0; i<28; i++)
       printf("-");

    printf("\n\nAverage waiting time     : %5.2fms", awat);
    printf("\nAverage turn around time : %5.2fms\n", atur);

    printf("\n\nGANTT Chart\n");
    printf("-");
    for(i=0; i<(p[n-1].ttime + 2*n); i++)
       printf("-");
    printf("\n");
    printf("|");
    for(i=0; i<n; i++)
    {
       k = p[i].btime/2;
       for(j=0; j<k; j++)
          printf(" ");
       printf("P%d",p[i].pid);
       for(j=k+1; j<p[i].btime; j++)
          printf(" ");
       printf("|");
    }
    printf("\n");
    printf("-");
    for(i=0; i<(p[n-1].ttime + 2*n); i++)
       printf("-");
    printf("\n");
    printf("0");
    for(i=0; i<n; i++)
    {
       for(j=0; j<p[i].btime; j++)
          printf(" ");
       printf("%2d",p[i].ttime);
    }
}
```

Output

```
Enter no. of process : 4
Burst time for process P1 (in ms) : 10
Burst time for process P2 (in ms) : 4
Burst time for process P3 (in ms) : 11
Burst time for process P4 (in ms) : 6

       FCFS Scheduling

----------------------------
Process B-Time T-Time W-Time
----------------------------
  P1      10     10      0
  P2       4     14     10
  P3      11     25     14
  P4       6     31     25
----------------------------

Average waiting time     : 12.25ms
Average turn around time : 20.00ms


GANTT Chart
----------------------------------------
|     P1    |  P2 |     P3     |   P4  |
----------------------------------------
0           10    14           25      31
```

Result

Thus waiting time & turnaround time for processes based on FCFS scheduling was computed and the average waiting time was determined.

Exp. No. 5b **SJF Scheduling**

Date:

Aim

To schedule snapshot of processes queued according to SJF scheduling.

Shortest Job First (SJF)

- Process that requires smallest burst time is processed first.
- SJF can be preemptive or non–preemptive
- When two processes require same amount of CPU utilization, FCFS is used to break the tie.
- Generally efficient as it results in minimal average waiting time.
- Can result in starvation, since long critical processes may not be processed.

Algorithm

1. Define an array of structure *process* with members *pid*, *btime*, *wtime* & *ttime*.
2. Get length of the ready queue, i.e., number of process (say *n*)
3. Obtain *btime* for each process.
4. *Sort* the processes according to their *btime* in ascending order.
   a. If two process have same *btime*, then FCFS is used to resolve the tie.
5. The *wtime* for first process is 0.
6. Compute *wtime* and *ttime* for each process as:
   a. $wtime_{i+1} = wtime_i + btime_i$
   b. $ttime_i = wtime_i + btime_i$
7. Compute average waiting time *awat* and average turn around time *atur*.
8. Display *btime*, *ttime* and *wtime* for each process.
9. Display GANTT chart for the above scheduling
10. Display *awat* and *atur*
11. Stop

Program

```
/* SJF Scheduling - sjf.c */

#include <stdio.h>

struct process
{
   int pid;
   int btime;
   int wtime;
   int ttime;
} p[10], temp;

main()
{
   int i,j,k,n,ttur,twat;
   float awat,atur;

   printf("Enter no. of process : ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
   {
      printf("Burst time for process P%d (in ms) : ",(i+1));
      scanf("%d", &p[i].btime);
      p[i].pid = i+1;
   }

   for(i=0; i<n-1; i++)
   {
      for(j=i+1; j<n; j++)
      {
         if((p[i].btime > p[j].btime) ||
            (p[i].btime == p[j].btime && p[i].pid > p[j].pid))
         {
            temp = p[i];
            p[i] = p[j];
            p[j] = temp;
         }

      }
   }
   p[0].wtime = 0;
   for(i=0; i<n; i++)
   {
      p[i+1].wtime = p[i].wtime + p[i].btime;
      p[i].ttime = p[i].wtime + p[i].btime;
   }
   ttur = twat = 0;
```

```
   for(i=0; i<n; i++)
   {
      ttur += p[i].ttime;
      twat += p[i].wtime;
   }
   awat = (float)twat / n;
   atur = (float)ttur / n;

   printf("\n       SJF Scheduling\n\n");
   for(i=0; i<28; i++)
      printf("-");
   printf("\nProcess B-Time T-Time W-Time\n");
   for(i=0; i<28; i++)
      printf("-");
   for(i=0; i<n; i++)
      printf("\n  P%-4d\t%4d\t%3d\t%2d",
               p[i].pid,p[i].btime,p[i].ttime,p[i].wtime);
   printf("\n");
   for(i=0; i<28; i++)
      printf("-");
   printf("\n\nAverage waiting time     : %5.2fms", awat);
   printf("\nAverage turn around time : %5.2fms\n", atur);

   printf("\n\nGANTT Chart\n");
   printf("-");
   for(i=0; i<(p[n-1].ttime + 2*n); i++)
      printf("-");
   printf("\n|");
   for(i=0; i<n; i++)
   {
      k = p[i].btime/2;
      for(j=0; j<k; j++)
         printf(" ");
      printf("P%d",p[i].pid);
      for(j=k+1; j<p[i].btime; j++)
         printf(" ");
      printf("|");
   }
   printf("\n-");
   for(i=0; i<(p[n-1].ttime + 2*n); i++)
      printf("-");
   printf("\n0");
   for(i=0; i<n; i++)
   {
      for(j=0; j<p[i].btime; j++)
         printf(" ");
      printf("%2d",p[i].ttime);
   }
}
```

Output

```
Enter no. of process : 5
Burst time for process P1 (in ms) : 10
Burst time for process P2 (in ms) : 6
Burst time for process P3 (in ms) : 5
Burst time for process P4 (in ms) : 6
Burst time for process P5 (in ms) : 9

       SJF Scheduling

----------------------------
Process B-Time T-Time W-Time
----------------------------
  P3       5      5      0
  P2       6     11      5
  P4       6     17     11
  P5       9     26     17
  P1      10     36     26
----------------------------

Average waiting time     : 11.80ms
Average turn around time : 19.00ms


GANTT Chart
-----------------------------------------------
|  P3  |   P2  |   P4  |    P5    |     P1    |
-----------------------------------------------
0      5      11      17         26          36
```

Result

Thus waiting time & turnaround time for processes based on SJF scheduling was computed and the average waiting time was determined.

Exp. No. 5c **Priority Scheduling**

Date:

Aim

To schedule snapshot of processes queued according to Priority scheduling.

Priority

- Process that has higher priority is processed first.
- Prioirty can be preemptive or non–preemptive
- When two processes have same priority, FCFS is used to break the tie.
- Can result in starvation, since low priority processes may not be processed.

Algorithm

1. Define an array of structure *process* with members *pid*, *btime*, *pri*, *wtime* & *ttime*.
2. Get length of the ready queue, i.e., number of process (say *n*)
3. Obtain *btime* and *pri* for each process.
4. *Sort* the processes according to their *pri* in ascending order.
   a. If two process have same *pri*, then FCFS is used to resolve the tie.
5. The *wtime* for first process is 0.
6. Compute *wtime* and *ttime* for each process as:
   a. $wtime_{i+1} = wtime_i + btime_i$
   b. $ttime_i = wtime_i + btime_i$
7. Compute average waiting time *awat* and average turn around time *atur*
8. Display the *btime*, *pri*, *ttime* and *wtime* for each process.
9. Display GANTT chart for the above scheduling
10. Display *awat* and *atur*
11. Stop

Program

```
/* Priority Scheduling  - pri.c */

#include <stdio.h>

struct process
{
   int pid;
   int btime;
   int pri;
   int wtime;
   int ttime;
} p[10], temp;

main()
{
   int i,j,k,n,ttur,twat;
   float awat,atur;

   printf("Enter no. of process : ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
   {
      printf("Burst time for process P%d (in ms) : ", (i+1));
      scanf("%d", &p[i].btime);
      printf("Priority for process P%d : ", (i+1));
      scanf("%d", &p[i].pri);
      p[i].pid = i+1;
   }

   for(i=0; i<n-1; i++)
   {
      for(j=i+1; j<n; j++)
      {
         if((p[i].pri > p[j].pri) ||
            (p[i].pri == p[j].pri && p[i].pid > p[j].pid) )
         {
            temp = p[i];
            p[i] = p[j];
            p[j] = temp;
         }

      }
   }
   p[0].wtime = 0;
   for(i=0; i<n; i++)
   {
      p[i+1].wtime = p[i].wtime + p[i].btime;
      p[i].ttime = p[i].wtime + p[i].btime;
   }
```

```
    ttur = twat = 0;
    for(i=0; i<n; i++)
    {
       ttur += p[i].ttime;
       twat += p[i].wtime;
    }
    awat = (float)twat / n;
    atur = (float)ttur / n;

    printf("\n\t Priority Scheduling\n\n");
    for(i=0; i<38; i++)
       printf("-");
    printf("\nProcess B-Time Priority T-Time  W-Time\n");
    for(i=0; i<38; i++)
       printf("-");
    for (i=0; i<n; i++)
       printf("\n  P%-4d\t%4d\t%3d\t%4d\t%4d",
          p[i].pid,p[i].btime,p[i].pri,p[i].ttime,p[i].wtime);
    printf("\n");
    for(i=0; i<38; i++)
       printf("-");

    printf("\n\nAverage waiting time     : %5.2fms", awat);
    printf("\nAverage turn around time : %5.2fms\n", atur);

    printf("\n\nGANTT Chart\n");
    printf("-");
    for(i=0; i<(p[n-1].ttime + 2*n); i++)
       printf("-");
    printf("\n|");
    for(i=0; i<n; i++)
    {
       k = p[i].btime/2;
       for(j=0; j<k; j++)
          printf(" ");
       printf("P%d",p[i].pid);
       for(j=k+1; j<p[i].btime; j++)
          printf(" ");
       printf("|");
    }
    printf("\n-");
    for(i=0; i<(p[n-1].ttime + 2*n); i++)
       printf("-");
    printf("\n0");
    for(i=0; i<n; i++)
    {
       for(j=0; j<p[i].btime; j++)
          printf(" ");
       printf("%2d",p[i].ttime);
    }
}
```

Output

```
Enter no. of process : 5
Burst time for process P1 (in ms) : 10
Priority for process P1 : 3
Burst time for process P2 (in ms) : 7
Priority for process P2 : 1
Burst time for process P3 (in ms) : 6
Priority for process P3 : 3
Burst time for process P4 (in ms) : 13
Priority for process P4 : 4
Burst time for process P5 (in ms) : 5
Priority for process P5 : 2

         Priority Scheduling

-------------------------------------
Process B-Time Priority T-Time  W-Time
-------------------------------------
  P2       7      1        7       0
  P5       5      2       12       7
  P1      10      3       22      12
  P3       6      3       28      22
  P4      13      4       41      28
-------------------------------------

Average waiting time     : 13.80ms
Average turn around time : 22.00ms

GANTT Chart
-------------------------------------------------
|   P2   |  P5  |     P1    |   P3  |      P4      |
-------------------------------------------------
0        7     12          22      28             41
```

Result

Thus waiting time & turnaround time for processes based on Priority scheduling was computed and the average waiting time was determined.

Exp. No. 5d **Round Robin Scheduling**

Date:

Aim

To schedule snapshot of processes queued according to Round robin scheduling.

Round Robin

- All processes are processed one by one as they have arrived, but in rounds.
- Each process cannot take more than the time slice per round.
- Round robin is a fair preemptive scheduling algorithm.
- A process that is yet to complete in a round is preempted after the time slice and put at the end of the queue.
- When a process is completely processed, it is removed from the queue.

Algorithm

1. Get length of the ready queue, i.e., number of process (say n)
2. Obtain *Burst* time B_i for each processes P_i.
3. Get the *time slice* per round, say *TS*
4. Determine the number of rounds for each process.
5. The wait time for first process is 0.
6. If $B_i > TS$ then process takes more than one round. Therefore turnaround and waiting time should include the time spent for other remaining processes in the same round.
7. Calculate *average* waiting time and turn around time
8. Display the GANTT chart that includes
   a. order in which the processes were processed in progression of rounds
   b. Turnaround time T_i for each process in progression of rounds.
9. Display the *burst* time, *turnaround* time and *wait* time for each process (in order of rounds they were processed).
10. Display *average* wait time and turnaround time
11. Stop

Program

```
/* Round robin scheduling  - rr.c */

#include <stdio.h>

main()
{
   int i,x=-1,k[10],m=0,n,t,s=0;
   int a[50],temp,b[50],p[10],bur[10],bur1[10];
   int wat[10],tur[10],ttur=0,twat=0,j=0;
   float awat,atur;

   printf("Enter no. of process : ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
   {
      printf("Burst time for process P%d : ", (i+1));
      scanf("%d", &bur[i]);
      bur1[i] = bur[i];
   }
   printf("Enter the time slice (in ms) : ");
   scanf("%d", &t);

   for(i=0; i<n; i++)
   {
      b[i] = bur[i] / t;
      if((bur[i]%t) != 0)
         b[i] += 1;
      m += b[i];
   }

   printf("\n\t\tRound Robin Scheduling\n");

   printf("\nGANTT Chart\n");
   for(i=0; i<m; i++)
      printf("--------");
   printf("\n");

   a[0] = 0;
   while(j < m)
   {
      if(x == n-1)
         x = 0;
      else
         x++;
      if(bur[x] >= t)
      {
         bur[x] -= t;
         a[j+1] = a[j] + t;
```

```
      if(b[x] == 1)
      {
         p[s] = x;
         k[s] = a[j+1];
         s++;
      }
      j++;
      b[x] -= 1;
      printf("  P%d    |", x+1);
   }
   else if(bur[x] != 0)
   {
      a[j+1] = a[j] + bur[x];
      bur[x] = 0;
      if(b[x] == 1)
      {
         p[s] = x;
         k[s] = a[j+1];
         s++;
      }
      j++;
      b[x] -= 1;
      printf("  P%d  |",x+1);
   }
}

printf("\n");
for(i=0;i<m;i++)
   printf("--------");
printf("\n");

for(j=0; j<=m; j++)
   printf("%d\t", a[j]);

for(i=0; i<n; i++)
{
   for(j=i+1; j<n; j++)
   {
      if(p[i] > p[j])
      {
         temp = p[i];
         p[i] = p[j];
         p[j] = temp;

         temp = k[i];
         k[i] = k[j];
         k[j] = temp;
      }
   }
}
```

```
   for(i=0; i<n; i++)
   {
      wat[i] = k[i] - bur1[i];
      tur[i] = k[i];
   }
   for(i=0; i<n; i++)
   {
      ttur += tur[i];
      twat += wat[i];
   }

   printf("\n\n");
   for(i=0; i<30; i++)
      printf("-");
   printf("\nProcess\tBurst\tTrnd\tWait\n");
   for(i=0; i<30; i++)
      printf("-");
   for (i=0; i<n; i++)
      printf("\nP%-4d\t%4d\t%4d\t%4d", p[i]+1, bur1[i],
                     tur[i],wat[i]);
   printf("\n");
   for(i=0; i<30; i++)
      printf("-");

   awat = (float)twat / n;
   atur = (float)ttur / n;
   printf("\n\nAverage waiting time     : %.2f ms", awat);
   printf("\nAverage turn around time : %.2f ms\n", atur);
}
```

Output

```
Enter no. of process : 5
Burst time for process P1 : 10
Burst time for process P2 : 29
Burst time for process P3 : 3
Burst time for process P4 : 7
Burst time for process P5 : 12
Enter the time slice (in ms) : 10

                Round Robin Scheduling

GANTT Chart
-------------------------------------------------------------------
P1    |  P2   |  P3  |  P4  |  P5  |  P2   |  P5   |   P2  |
-------------------------------------------------------------------
0    10      20     23      30      40      50      52      61

------------------------------
Process Burst   Trnd    Wait
------------------------------
P1        10      10       0
P2        29      61      32
P3         3      23      20
P4         7      30      23
P5        12      52      40
------------------------------

Average waiting time     : 23.00 ms
Average turn around time : 35.20 ms
```

Result

Thus waiting time and turnaround time for processes based on Round robin scheduling was computed and the average waiting time was determined.