# 실습 (배치 시스템에서의 스케줄링)

다음 3가지 스케줄링을 시뮬레이션하는 프로그램을 각각 실행해보시오.

FCFS.c
SJF.c
SRTN.cpp

**컴파일 방법**

```
gcc   -o FCFS   FCFS.c
gcc   -o SJF   SJF.c
g++   -o SRTN   SRTN.cpp
```

**입력 데이터 파일 예 (sample.dat):**

```
0   8       (도착시각 0, burst time은 8)
1   6
3   3
12   4
```

(입력 데이터 파일의 내용은 각 줄마다 arrival time과 burst time이 있음)

**실행 1 (입력: sample.dat)**

```
./FCFS   < sample.dat
./SJF < sample.dat
./SRTN < sample.dat
```

**실행 2 (랜덤 데이터 파일 사용)**

랜덤하게 arrival time과 burst time을 발생시켜 입력 데이터 파일 생성

1) randATBT.c 컴파일

```
gcc   -o randATBT   randATBT.c
```

2) randATBT 실행하여 랜덤 데이터 파일 생성 (데이터 개수 10000)

```
./randATBT 10000   > load1.dat
./randATBT 10000   > load2.dat
./randATBT 10000   > load3.dat
```

3) 생성한 3개의 랜덤 데이터 파일을 이용하여 각 3개의 스케줄링 결과 비교
load1.dat 데이터 파일을 사용하는 경우의 예:

```
./FCFS   < load1.dat   | tail
./SJF   < load1.dat | tail
./SRTN   < load1.dat | tail
```

<<질의사항>>

1. (실행 1)에서 몇 개의 프로세스를 스케줄링하는가?
2. (실행 1)에서 waiting time과 turnaround time이 의미는 무엇인가?
3. (실행 1)에서 3개의 스케줄링을 waiting time과 turnaround time 측면에서 비교하시오.
4. (실행 2)에서 3개의 스케줄링을 waiting time과 turnaround time 측면에서 비교하시오. (각 3개의 입력 데이터 파일에 대해)
5. SJF가 FCFS보다 좋은 점과 그렇지 않은 점은 무엇이라고 생각하는가?
6. SRTN의 장점과 단점은 무엇이라고 생각하는가?
7. 이 실험에서 throughput은 3개의 스케줄링 간에 차이가 있다고 생각하는가?

# [ FCFS.c ]

```
// C program for implementation of FCFS
#include <stdio.h>

#define NUM_PROC       10000

// Function to find the waiting time for all
// processes
void findWaitingTime(int n, int bt[], int wt[], int at[])
{
      int service_time[n];
      service_time[0] = at[0];
      wt[0] = 0;

      // calculating waiting time
      for (int i = 1; i < n ; i++ )
      {
            // Add burst time of previous processes
            service_time[i] = service_time[i-1] +  bt[i-1];

            // Find waiting time for current process =
            // sum - at[i]
            wt[i] = service_time[i] - at[i];

            // If waiting time for a process is in negative
            // that means it is already in the ready queue
            // before CPU becomes idle so its waiting time is 0
            if (wt[i] < 0)
                  wt[i] = 0;
      }
}

// Function to calculate turn around time
void findTurnAroundTime(int n, int bt[], int wt[], int tat[])
{
      // Calculating turnaround time by adding bt[i] +  wt[i]
      for (int i = 0; i < n ; i++ )
            tat[i] = bt[i] +  wt[i];
}

// Function to calculate average waiting and turn-around
// times.
void findavgTime(int n, int bt[], int at[])
{
      int wt[n], tat[n];

      // Function to find waiting time of all processes
      findWaitingTime(n, bt, wt, at);
```

```
    // Function to find turn around time for all processes
    findTurnAroundTime(n, bt, wt, tat);

    // Display processes along with all details
    printf("ProcessID  Arrival Time  Burst Time  Waiting Time  Turnaround Time
Completion Time₩n");
    int total_wt = 0, total_tat = 0;
    for (int i = 0 ; i < n ; i++)
    {
        total_wt = total_wt +  wt[i];
        total_tat = total_tat +  tat[i];
        int compl_time = tat[i] +  at[i];

printf("     %5d         %7d       %7d         %7d             %7d             %7d₩n",
                i, at[i], bt[i], wt[i], tat[i], compl_time);
    }
    printf("Avg. Waiting Time      = %9.2f₩n", (float)total_wt / (float)n);
    printf("Avg. Turnaround Time = %9.2f₩n", (float)total_tat / (float)n);
}

int inputData(int arrival_time[], int burst_time[])
{
    int       i = 0;
    int       num;
    do {
        num = scanf("%d %d", &arrival_time[i], &burst_time[i]);
        if (num <= 0)          // End-of-file or zero data
            break;
        i++;
    } while (1);
    return i;
}
// Driver code
int main()
{
    // # of Process
    int       n;

    // Burst time of all processes
    int burst_time[NUM_PROC];

    // Arrival time of all processes
    int arrival_time[NUM_PROC];

    n = inputData(arrival_time, burst_time);
    findavgTime(n, burst_time, arrival_time);
    return 0;
}
```

## [ SJF.c ]

```
// C program to implement Shortest Job first with Arrival Time
#include <stdio.h>
#define NUM_PROC       10000

int mat[NUM_PROC][6];
/*   mat[i][0]: process ID
     mat[i][1]: arrival time
     mat[i][2]: burst time
     mat[i][3]:
     mat[i][4]: waiting time
     mat[i][5]: turnaround time
     mat[i][6]:
*/

void swap(int *a, int *b)
{
     int temp = *a;
     *a = *b;
     *b = temp;
}

void arrangeArrival(int num, int mat[][6])
{
     for(int i=0; i<num; i++) {
          for(int j=0; j<num-i-1; j++) {
               if(mat[j][1] > mat[j+1][1]) {
                    for(int k=0; k<5; k++) {
                         swap(&mat[j][k], &mat[j+1][k]);
                    }
               }
          }
     }
}

void completionTime(int num, int mat[][6])
{
     int temp, val;
     mat[0][3] = mat[0][1] +  mat[0][2];
     mat[0][5] = mat[0][3] - mat[0][1];
     mat[0][4] = mat[0][5] - mat[0][2];

     for(int i=1; i<num; i++) {
          temp = (mat[i-1][3]>mat[i][1]) ? mat[i-1][3] : mat[i][1];       // earliest start
time of the next job
          int low = mat[i][2];
          for(int j=i; j<num; j++) {
               if(temp >= mat[j][1] && low >= mat[j][2]) {
                    low = mat[j][2];
                    val = j;
```

```
            }
        }
        mat[val][3] = temp + mat[val][2];
        mat[val][5] = mat[val][3] - mat[val][1];
        mat[val][4] = mat[val][5] - mat[val][2];
        for(int k=0; k<6; k++) {
            swap(&mat[val][k], &mat[i][k]);
        }
    }
}

int main()
{
    int       num = 0;
    int       i = 0;
    int       readnum;

    // Read data: arrival time, burst time
    do {
        readnum = scanf("%d %d", &mat[i][1], &mat[i][2]);    // arrival time, burst
time
        if (readnum <= 0)          // End-of-file or zero data
            break;
        mat[i][0] = i;             // Process ID
        i++;
    } while (1);
    num = i;

    arrangeArrival(num, mat);
    completionTime(num, mat);

    // Display processes along with all details
    printf("ProcessID  Arrival Time  Burst Time  Waiting Time  Turnaround Time
Completion Time₩n");
    int total_wt = 0, total_tat = 0;
    for (int i = 0 ; i < num ; i++)
    {
        total_wt = total_wt + mat[i][4];
        total_tat = total_tat + mat[i][5];
        int compl_time = mat[i][5] + mat[i][1];

printf("      %5d          %7d       %7d          %7d             %7d             %7d₩n",
                mat[i][0], mat[i][1], mat[i][2], mat[i][4], mat[i][5], compl_time);
    }
    printf("Avg. Waiting Time      = %9.2f₩n", (float)total_wt / (float)num);
    printf("Avg. Turnaround Time = %9.2f₩n", (float)total_tat / (float)num);
}
```

## [ SRTN.cpp ]

```
#include <bits/stdc++.h>
#include <cstdio>
#define NUM_PROC      10000

using namespace std;

struct Process {
    int pid; // Process ID
    int at; // Arrival Time
    int bt; // Burst Time
};

// Function to find the waiting time for all
// processes
void findWaitingTime(Process proc[], int n, int wt[])
{
    int rt[n];

    // Copy the burst time into rt[]
    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;

    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    bool check = false;

    // Process until all processes gets
    // completed
    while (complete != n) {

        // Find process with minimum
        // remaining time among the
        // processes that arrives till the
        // current time`
        for (int j = 0; j < n; j++) {
            if ((proc[j].at <= t) &&
            (rt[j] < minm) && rt[j] > 0) {
                minm = rt[j];
                shortest = j;
                check = true;
            }
        }

        if (check == false) {
            t++;
            continue;
        }
```

```
            // Reduce remaining time by one
            rt[shortest]--;

            // Update minimum
            minm = rt[shortest];
            if (minm == 0)
                minm = INT_MAX;

            // If a process gets completely
            // executed
            if (rt[shortest] == 0) {

                // Increment complete
                complete++;
                check = false;

                // Find finish time of current
                // process
                finish_time = t + 1;

                // Calculate waiting time
                wt[shortest] = finish_time -
                               proc[shortest].bt -
                               proc[shortest].at;

                if (wt[shortest] < 0)
                    wt[shortest] = 0;
            }
            // Increment time
            t++;
        }
    }

    // Function to calculate turn around time
    void findTurnAroundTime(Process proc[], int n, int wt[], int tat[])
    {
        // calculating turnaround time by adding
        // bt[i] + wt[i]
        for (int i = 0; i < n; i++)
            tat[i] = proc[i].bt + wt[i];
    }

    // Function to calculate average time
    void findavgTime(Process proc[], int n)
    {
        int wt[n], tat[n];
        int total_wt = 0, total_tat = 0;

        // Function to find waiting time of all
        // processes
```

```
    findWaitingTime(proc, n, wt);

    // Function to find turn around time for
    // all processes
    findTurnAroundTime(proc, n, wt, tat);

    // Display processes along with all
    // details
    printf("ProcessID  Arrival Time  Burst Time  Waiting Time  Turnaround Time
Completion Time₩n");
//     int total_wt = 0, total_tat = 0;
    for (int i = 0 ; i < n ; i++) {
        total_wt = total_wt +  wt[i];
        total_tat = total_tat +  tat[i];
        int compl_time = tat[i] +  proc[i].at;

printf("     %5d         %7d       %7d         %7d             %7d             %7d₩n",
                proc[i].pid, proc[i].at, proc[i].bt, wt[i], tat[i], compl_time);
    }
    printf("Avg. Waiting Time      = %9.2f₩n", (float)total_wt / (float)n);
    printf("Avg. Turnaround Time = %9.2f₩n", (float)total_tat / (float)n);

}

int inputData(Process proc[])
{
    int       i = 0;
    int       num;
    do {
        num = scanf("%d %d", &proc[i].at, &proc[i].bt);
        if (num <= 0)          // End-of-file or zero data
            break;
        proc[i].pid = i;
        i++;
    } while (1);
    return i;
}

// Driver code
int main()
{
//     Process proc[] = { { 1, 1, 6 }, { 2, 1, 8 },
//                        { 3, 2, 7 }, { 4, 3, 3 } };
    Process proc[NUM_PROC];

    int n = inputData(proc);

    findavgTime(proc, n);
    return 0;
}
```

## [ randATBT.c ]

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define TRUE      1
#define FALSE     0
int main(int argc, char *argv[])
{
        if (argc < 2) {
                fprintf(stderr, "Please provide the number of pairs <arrival time, burst time>₩n");
                fprintf(stderr, "Ex: gen_rand 100₩n");
                exit(1);
        }

        int       n = atoi(argv[1]);
        int       ATZERO = FALSE;
        if (argc == 3 && atoi(argv[2]) == 0)
                ATZERO = TRUE;              // arrival time is always 0
        int       at, bt;             // arrival time, burst time
        int       cur_at;
        srand(time(NULL));
        for (int i = 0; i < n; i++) {
                at = rand() % 11;                   // 0 ~ 10
                bt = rand() % 10 +  1;     // 1 ~ 10
                cur_at = cur_at +  at;
                if (ATZERO)
                        printf("%d %d₩n", 0, bt);
                else
                        printf("%d %d₩n", cur_at, bt);
        }
}
```