#include <stdio.h>

#define MAX\_QUEUE\_SIZE 100

typedef struct Processes {

int p\_no;

int arr\_time;

int burst\_time;

} Process;

Process \* queue[MAX\_QUEUE\_SIZE];

int front = 0, rear = -1, processed = 0, curr\_time = 0, tq = 6;

int last\_front = 0, last\_rear = -1;

void swap(Process \* a, Process \* b) {

Process temp = \* a;

\* a = \* b;

\* b = temp;

}

void sort(Process p[], int n) {

int i, j;

short swapped;

for (i = 0; i < n; ++i) {

swapped = 0;

for (j = 0; j < n-i-1; ++j)

{

if (p[j].arr\_time > p[j+1].arr\_time)

{

swap(&p[j], &p[j+1]);

swapped = 1;

}

}

if (swapped == 0)

break;

}

}

void enqueue(Process p[], int n) {

int i, j, can\_insert;

for (i = 0; i < n; ++i)

{

can\_insert = 1;

if (p[i].arr\_time <= curr\_time && p[i].burst\_time > 0)

{

if (front == 0) {

queue[++rear] = &p[i];

}

else

{

for (j = last\_front; j <= last\_rear; ++j) {

if (queue[j]->p\_no == p[i].p\_no)

can\_insert = 0;

}

if (can\_insert == 1)

queue[++rear] = &p[i];

}

}

}

for (i = last\_front; i <= last\_rear; ++i)

{

if (queue[i]->burst\_time > 0)

queue[++rear] = queue[i];

}

}

void execute() {

int i;

if (front-1 == rear) {

printf("CPU idle for 1 second.\n");

curr\_time += 1;

}

else {

last\_front = front;

last\_rear = rear;

for (i = front; i <= rear; ++i, ++front)

{

if (queue[i]->burst\_time > tq)

{

queue[i]->burst\_time -= tq;

curr\_time += tq;

printf("Process number %d excuted till %d seconds.\n", queue[i]->p\_no, curr\_time);

}

else if (queue[i]->burst\_time > 0)

{

curr\_time += queue[i]->burst\_time;

queue[i]->burst\_time = 0;

printf("Process number %d excuted till %d seconds.\n", queue[i]->p\_no, curr\_time);

++processed;

}

}

}

}

int main() {

int n, i;

short err\_flag = 0;

do {

if (err\_flag == 1)

fprintf(stderr, "\nNumber of processes should be greater than 1.\n");

printf("Enter the number of processes: ");

scanf("%d", &n);

err\_flag = 1;

} while (n < 1);

err\_flag = 0;

Process p[n];

for (i = 0; i < n; ++i) {

printf("\n");

printf("Enter arrival time of process %d: ", i+1);

scanf("%d", &p[i].arr\_time);

printf("Enter burst time of process %d: ", i+1);

scanf("%d", &p[i].burst\_time);

p[i].p\_no = i+1;

}

sort(&p[0], n); // Sort the processes according to the arrival time of each process.

while (1) {

enqueue(p, n);

printf("\nIn queue: ");

for (i = 0; i <= rear; ++i) {

printf("%d ", queue[i]->p\_no);

}

printf("\nFront = %d, Rear = %d.\n\n", front, rear);

execute();

// If all the processes have been processed, break from the loop.

if (processed == n)

break;

}

return 0;

}