**РУСЕНСКИ УНИВЕРСИТЕТ „АНГЕЛ КЪНЧЕВ“**

**КУРСОВА ЗАДАЧА**

ПО

ОПЕРАЦИОННИ СИСТЕМИ

Студент:

Фак. Номер:

Група:

Специалност:

Дата: Проверил:

**Съдържание**

[1. Задание 3](#_Toc437994880)

[2. Описание на използваните структури и алгоритми 3](#_Toc437994881)

[3. Код на програмата (листинг) 4](#_Toc437994882)

[4. Демо 9](#_Toc437994883)

# Задание

Да се разработи програмна система, симулираща работата на модула за **управление на процесора (стратегия Shortest Remaining Time)** при следните изходни данни:

* ***Основни необходими структури:***
  + Ready таблица на готовите процеси
* ***Основни обработки:***
  + Load стартиране на нов процес;
  + Dispatch извикване на диспечер по стратегия SRT;
  + Finish завършване на изпълняващия се процес
  + Show показване на състоянието на основните структури

Разработеният проект да има подходящ потребителски интерфейс за обръщение към основните обработки. Обяснителната записка да съдържа заданието, описание на използваните структури данни и алгоритми и програмен листинг.

# Описание на използваните структури и алгоритми

Най-краткото оставащо време (Shortest Remaining Time, SRT) е версия на дисциплината Shortest Process Next (SPN или SJF – Shortest Job First). Диспечерът избира най-краткото очаквано оставащо време за обработка. Когато един нов процес се присъедини към опашката на готовите процеси, възможно е той да има по-кратко оставащо време за обработка, отколкото текущо изпълняваният процес. Следователно, диспечерът може да превключи всеки път, когато нов процес стане готов. Както при SPN, SRT трябва да разполага с изчисленото време за обработка, за да се изпълни функцията за избор. При тази стратегия има риск за увисване на по-дългите процеси.

* int main() – главна функция
* void calculateProcessSequence() – изпълнява основните изчисления нужни при диспечиране по стратегията SRT. В това число влизат: изчисляване времето на изчакване на всеки процес, изчисляване на средното време на изчакване Tw и оборотното време Tr, също така и данните, необходими за „изчертаването“ на диаграмата на Гант.
  + Tw, Tr и диаграмата на Гант не играят роля при реализирането на алгоритъма SRT, те се изчисляват и се визуализират просто информативно, но Tw и Tr могат да бъдат използвани за сравняване на стратегии. При разработването на приложението беше използван видео материал ( <https://www.youtube.com/watch?v=h-e7QtjfmkI> ), в който се показва как работи стратегията SRT и се изчисляват тези данни и затова бяха включени в приложението.
* int findAptProcessNumber(int currentTime) – служи за намиране на процеса с най-краткото очаквано оставащо време за обработка.
* void drawGanttChart() – служи за отпечатване на диаграмата на Гант. Тази диаграма илюстрира продължителността на изпълняване на процесите и направените превключвания.

# Код на програмата (листинг)

#include <stdlib.h>

#include <stdio.h>

#include <locale.h>

int numberOfProcesses,

totalCPUBurstTime,

\*arrivalTime,

\*CPUBurstTime,

\*CPUBurstTimeCopy,

\*processNumber,

minimumArrivalTime,

\*processSequenceForEachSecond,

\*processFinishSequence,

\*waitingTime;

float averageTurnAroundTime,

averageWaitingTime = 0;

/\* помощни масиви за изчертавато на Гант диаграмата \*/

int \*processNumberGantt,

\*CPUBurstTimeGantt,

ganttSize;

void drawGanttChart();

void calculateProcessSequence();

int findAptProcessNumber(int);

int main()

{

/\* кирилица \*/

setlocale(LC\_ALL, "bgr");

int i, j, temp;

printf("Въведете броя на процесите : ");

scanf("%d", &numberOfProcesses);

arrivalTime = (int\*)malloc(sizeof(int) \* numberOfProcesses);

CPUBurstTime = (int\*)malloc(sizeof(int) \* numberOfProcesses);

CPUBurstTimeCopy = (int\*)malloc(sizeof(int) \* numberOfProcesses);

processNumber = (int\*)malloc(sizeof(int) \* numberOfProcesses);

waitingTime = (int\*)malloc(sizeof(int) \* numberOfProcesses);

processFinishSequence = (int\*)malloc(sizeof(int) \* numberOfProcesses);

minimumArrivalTime = 2147483647;

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

{

processFinishSequence[i] = -1;

waitingTime[i] = 0;

processNumber[i] = i;

printf("\nВъведете данните за процес номер %d\n", i + 1);

printf("\n");

printf("Време на пристигане : ");

scanf("%d", &arrivalTime[i]);

printf("Необходимо време за изпълнение : ");

scanf("%d", &CPUBurstTime[i]);

CPUBurstTimeCopy[i] = CPUBurstTime[i];

totalCPUBurstTime += CPUBurstTime[i];

if (minimumArrivalTime > arrivalTime[i])

minimumArrivalTime = arrivalTime[i];

}

processSequenceForEachSecond = (int\*)malloc(sizeof(int)\*totalCPUBurstTime);

calculateProcessSequence();

printf("\nПроцес\t|\tTw\t|\tTr\n\n");

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

{

printf("\nP[%d]\t|\t%d\t|\t%d",

processFinishSequence[i] + 1,

waitingTime[processFinishSequence[i]],

waitingTime[processFinishSequence[i]] + CPUBurstTimeCopy[processFinishSequence[i]]);

}

printf("\n\nСредно време, изразходено за изчакване на процесора = %.2f", averageWaitingTime);

printf("\nСредно оборотно време = %.2f\n\n", averageTurnAroundTime);

drawGanttChart();

/\* освобождаване на заделената динамична памет \*/

free(arrivalTime);

free(CPUBurstTime);

free(CPUBurstTimeCopy);

free(processNumber);

free(processSequenceForEachSecond);

free(processFinishSequence);

free(waitingTime);

free(processNumberGantt);

free(CPUBurstTimeGantt);

return 0;

}

void calculateProcessSequence()

{

int i, j, pNumber, prevProcess, tempCPUBurstTime, counter, prevProcesss;

counter = 0;

for (i = minimumArrivalTime; i < totalCPUBurstTime + minimumArrivalTime; i++)

{

pNumber = findAptProcessNumber(i);

processSequenceForEachSecond[i - minimumArrivalTime] = pNumber;

CPUBurstTime[pNumber]--;

if (CPUBurstTime[pNumber] == 0)

{

processFinishSequence[counter++] = pNumber;

}

/\* изчисляване време на изчакване на всеки процес \*/

for (j = 0; j < numberOfProcesses; j++)

if (CPUBurstTime[j] != 0 && arrivalTime[j] <= i && j != pNumber)

waitingTime[j]++;

}

/\* изчисляване размера на масивите за Гант диаграмата \*/

ganttSize = 1;

prevProcess = processSequenceForEachSecond[0];

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

{

if (prevProcess != processSequenceForEachSecond[i])

ganttSize++;

prevProcess = processSequenceForEachSecond[i];

}

/\* заделяне на памет за масивите за Гант диаграмата \*/

processNumberGantt = (int\*)malloc(sizeof(int)\*ganttSize);

CPUBurstTimeGantt = (int\*)malloc(sizeof(int)\*ganttSize);

/\* вкарване на данните в масивите за Гант диаграмата \*/

prevProcess = processSequenceForEachSecond[0];

tempCPUBurstTime = 0;

counter = 0;

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

{

if (prevProcess != processSequenceForEachSecond[i])

{

processNumberGantt[counter] = prevProcess;

CPUBurstTimeGantt[counter] = tempCPUBurstTime;

counter++;

tempCPUBurstTime = 0;

}

tempCPUBurstTime++;

prevProcess = processSequenceForEachSecond[i];

}

CPUBurstTimeGantt[ganttSize - 1] = tempCPUBurstTime;

processNumberGantt[ganttSize - 1] = prevProcess;

/\* изчисляване на средното време на изчакване Tw и оборотното време Tr \*/

averageWaitingTime = 0;

averageTurnAroundTime = 0;

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

{

averageWaitingTime += waitingTime[i];

averageTurnAroundTime += waitingTime[i] + CPUBurstTimeCopy[i];

}

averageWaitingTime = averageWaitingTime / (float)numberOfProcesses;

averageTurnAroundTime = averageTurnAroundTime / (float)numberOfProcesses;

}

int findAptProcessNumber(int currentTime)

{

int i, min = 2147483647, pNumber;

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

if (arrivalTime[i] <= currentTime && min > CPUBurstTime[i] && CPUBurstTime[i] != 0)

{

min = CPUBurstTime[i];

pNumber = i;

}

return pNumber;

}

void drawGanttChart()

{

const int maxWidth = 100;

int scalingFactor, i, counter, tempi, currentTime;

printf("Гант диаграма : \n\n");

scalingFactor = maxWidth / totalCPUBurstTime;

for (i = 0; i < scalingFactor \* totalCPUBurstTime + 2 + ganttSize; i++)

printf("-");

printf("\n|");

counter = 0, tempi = 0;

for (i = 0; i < scalingFactor \* totalCPUBurstTime; i++)

if (i == CPUBurstTimeGantt[counter] \* scalingFactor + tempi)

{

counter++;

tempi = i;

printf("|");

}

else if (i == (CPUBurstTimeGantt[counter] \* scalingFactor) / 2 + tempi)

printf("P%d", processNumberGantt[counter] + 1);

else

printf(" ");

printf("|");

printf("\n");

for (i = 0; i < scalingFactor \* totalCPUBurstTime + 2 + ganttSize; i++)

printf("-");

printf("\n");

/\* печатане на времевите маркери \*/

counter = 0;

tempi = 0;

currentTime = minimumArrivalTime;

printf("%d", currentTime);

for (i = 0; i < scalingFactor \* totalCPUBurstTime; i++)

if (i == CPUBurstTimeGantt[counter] \* scalingFactor + tempi)

{

tempi = i;

currentTime += CPUBurstTimeGantt[counter];

counter++;

printf("%2d", currentTime);

}

else

{

printf(" ");

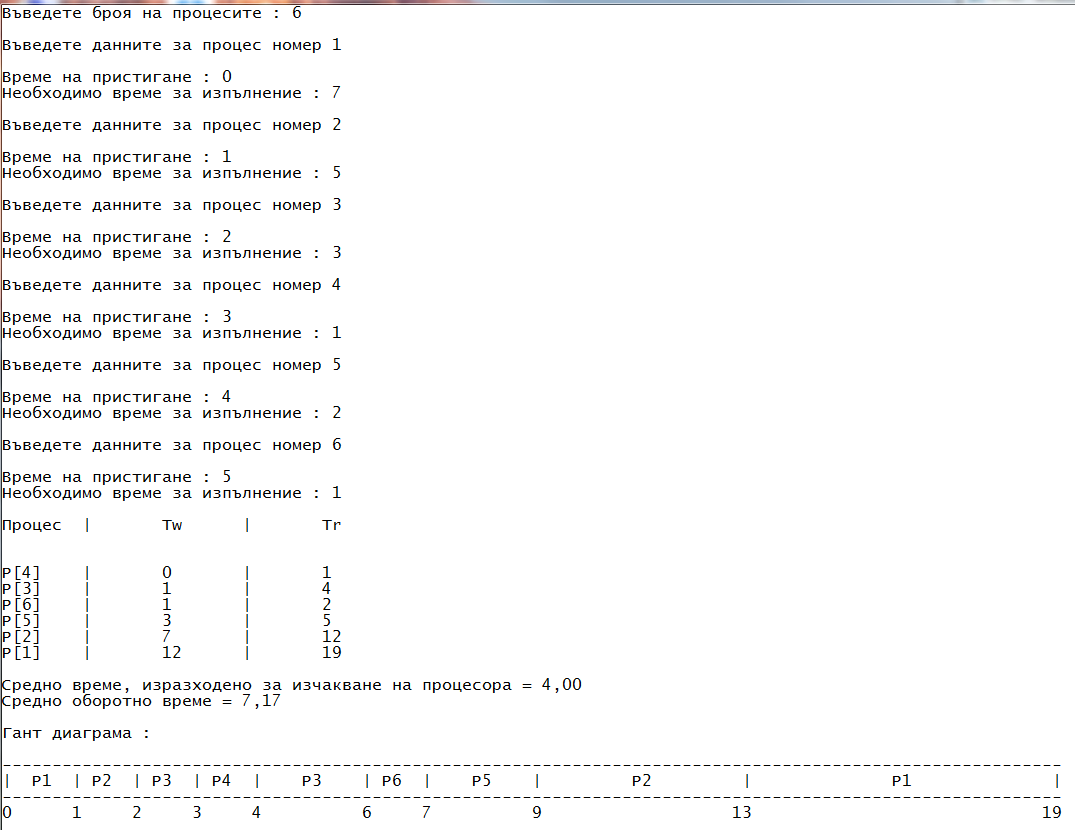
}

currentTime += CPUBurstTimeGantt[counter];

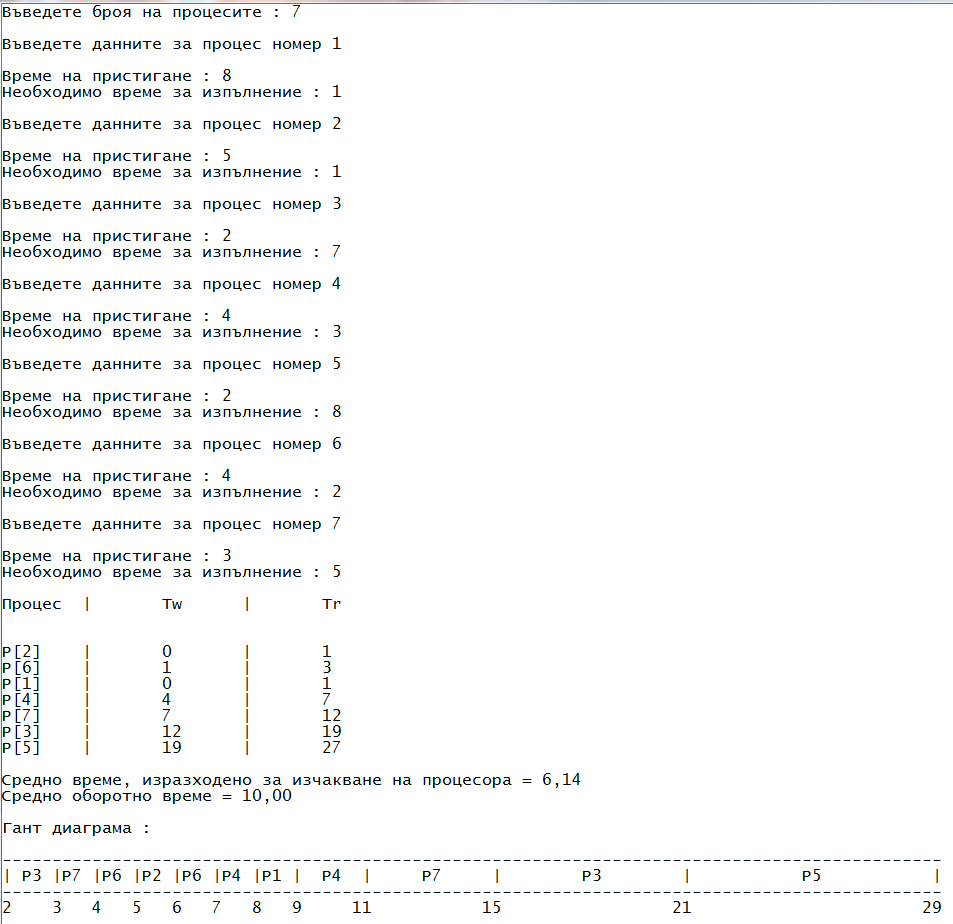
printf("%2d\n\n", currentTime);

}

# Демо



Фигура 1 Тестов пример 1



Фигура 2 Тестов пример 2