**Министерство Образования Республики Беларусь**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО»**

Кафедра «Информатика»

**Лабораторная работа № 6**

по дисциплине: «**Операционные системы и среды**»

на тему:«**Планирование процессов**»

Выполнил: ст. гр. ИП-32

Кардаш Е.В.

Принял: ст. преподаватель

Самовендюк Н.В.

Гомель 2021

**Цель работы:** изучить типовые алгоритмы планирования процессов.

**Задание 1. Не вытесняющие алгоритмы планирования процессов**

Выполнить различные алгоритмы планирований – First-Come, First-Served (FCFS) (прямой и обратный), Round Robin (RR), Shortest-Job-First (SJF) (не вытесняющий), Shortest-Job-First (SJF) (не вытесняющий приоритетный) для данных, приведенных в таблице 2.1 в соответствии со своим вариантом (номер по журналу). Вычислить полное время выполнения все процессов и каждого в отдельности, время ожидание для каждого процесса. Рассчитать среднее время выполнения процесса и среднее время ожидания. Результаты оформить в виде таблиц иллюстрирующих работу процессов.

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант  7 | Продолжительность процессов | Время появления в очереди | Приоритеты процессов |
| P0 | 10 | 0 | 1 |
| P1 | 2 | 4 | 3 |
| P2 | 3 | 3 | 2 |
| P3 | 1 | 0 | 3 |

**FCFS (First-Come, First Served)** (Прямой)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| P0 | И | И | И | И | И | И | И | И | И | И |  |  |  |  |  |  |  |
| P1 | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | И |  |  |  |  |  |
| P2 | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | И | И |  |  |
| P3 | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И |  |

Время выполнения процессов: P0 – 10, P1 – 12, P2 – 15, P3 – 16;

Время ожидания процессов: P0 – 0, P1 – 10, P2 – 12, P3 – 15

Среднее время выполнения: 13.25

Среднее время ожидания: 9.25

**FCFS (First-Come, First Served)** (Обратный)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| P0 | Г | Г | Г | Г | Г | Г | И | И | И | И | И | И | И | И | И | И |  |
| P1 | Г | Г | Г | Г | И | И |  |  |  |  |  |  |  |  |  |  |  |
| P2 | Г | И | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P3 | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Время выполнения процессов: P0 – 16, P1 – 6, P2 – 4, P3 – 1;

Время ожидания процессов: P0 – 6, P1 – 4, P2 – 1, P3 – 0

Среднее время выполнения: 6.75

Среднее время ожидания: 2.75

**Round Robin** (t = 3)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| P0 | И | И | И | Г | Г | Г | Г | Г | Г | И | И | И | И | И | И | И |  |
| P1 | Г | Г | Г | И | И |  |  |  |  |  |  |  |  |  |  |  |  |
| P2 | Г | Г | Г | Г | Г | И | И | И |  |  |  |  |  |  |  |  |  |
| P3 | Г | Г | Г | Г | Г | Г | Г | Г | И |  |  |  |  |  |  |  |  |

Время выполнения процессов: P0 – 16, P1 – 5, P2 – 8, P3 – 9;

Время ожидания процессов: P0 – 6, P1 – 3, P2 – 5, P3 – 8

Среднее время выполнения: 9.5

Среднее время ожидания: 5.5

**SJF** **(Shortest-Job-First)** (Не вытесняющий)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| P0 | Г | Г | Г | Г | Г | Г | И | И | И | И | И | И | И | И | И | И |  |
| P1 | Г | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P2 | Г | Г | Г | И | И | И |  |  |  |  |  |  |  |  |  |  |  |
| P3 | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Время выполнения процессов: P0 – 16, P1 – 3, P2 – 6, P3 – 1;

Время ожидания процессов: P0 – 6, P1 – 1, P2 – 3, P3 – 0

Среднее время выполнения: 6.5

Среднее время ожидания: 2.5

**SJF** **(Shortest-Job-First)** (Не вытесняющий, приоритетный)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | pr-ty | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| P0 | 1 | И | И | И | И | И | И | И | И | И | И |  |  |  |  |  |  |  |
| P1 | 3 | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | И |  |
| P2 | 2 | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | И | И |  |  |  |  |
| P3 | 3 | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И |  |  |  |

Время выполнения процессов: P0 – 10, P1 – 16, P2 – 13, P3 – 14;

Время ожидания процессов: P0 – 0, P1 – 14, P2 – 10, P3 – 13

Среднее время выполнения: 13.25

Среднее время ожидания: 9.25

**Задание 2. Вытесняющие алгоритмы планирования процессов**

Выполнить различные алгоритмы планирований – Shortest-Job-First (SJF) (вытесняющий) и Shortest-Job-First (SJF) (приоритетный) для данных, приведенных в таблице 2.1 в соответствии со своим вариантом. Вычислить полное время выполнения все процессов и каждого в отдельности, время ожидание для каждого процесса. Рассчитать среднее время выполнения процесса и среднее время ожидания. Результаты оформить в виде таблиц иллюстрирующих работу процессов.

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант  7 | Продолжительность процессов | Время появления в очереди | Приоритеты процессов |
| P0 | 10 | 0 | 1 |
| P1 | 2 | 4 | 3 |
| P2 | 3 | 3 | 2 |
| P3 | 1 | 0 | 3 |

**SJF** **(Shortest-Job-First)** (Вытесняющий)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | AT | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| P0 | 0 | Г | И | И | Г | Г | Г | Г | Г | И | И | И | И | И | И | И | И |  |
| P1 | 4 |  |  |  |  | Г | Г | И | И |  |  |  |  |  |  |  |  |  |
| P2 | 3 |  |  |  | И | И | И |  |  |  |  |  |  |  |  |  |  |  |
| P3 | 0 | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Время выполнения процессов: P0 – 16, P1 – 4, P2 – 3, P3 – 1;

Время ожидания процессов: P0 – 6, P1 – 2, P2 – 0, P3 – 0

Среднее время выполнения: 6

Среднее время ожидания: 2

**SJF** **(Shortest-Job-First)** (Вытесняющий приоритетный)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | pr-ty | AT | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| P0 | 1 | 0 | И | И | И | И | И | И | И | И | И | И |  |  |  |  |  |  |  |
| P1 | 3 | 4 |  |  |  |  | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | И |  |
| P2 | 2 | 3 |  |  |  | Г | Г | Г | Г | Г | Г | Г | И | И | И |  |  |  |  |
| P3 | 3 | 0 | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И |  |  |  |

Время выполнения процессов: P0 – 10, P1 – 12, P2 – 10, P3 – 14;

Время ожидания процессов: P0 – 0, P1 – 10, P2 – 7, P3 – 13

Среднее время выполнения: 11.5

Среднее время ожидания: 7.5

**Листинг программы:**

**Файл Program.cs:**

namespace Lab\_6

{

class Program

{

static void Main()

{

int[] burstTime = { 10, 2, 3, 1 };

int[] arrivalTime = { 0, 4, 3, 0 };

int[] priority = { 1, 3, 2, 3 };

Algorithms.FCFS(burstTime);

Algorithms.ReverseFCFS(burstTime);

Algorithms.RoundRobin(burstTime);

Algorithms.SJFNonPreemtive(burstTime);

Algorithms.SJFNonPreemtiveWithPriority(burstTime, priority);

Algorithms.SJFPreemtive(burstTime, arrivalTime);

Algorithms.SJFPreemtiveWithPriority(burstTime, arrivalTime, priority);

}

}

}

**Файл Print.cs:**

using System.Linq;

namespace Lab\_6

{

class Print

{

public static void Info(int[] TAT, int[] waitTime)

{

System.Console.WriteLine($"Время выполнения процессов: {TAT.Sum()}");

System.Console.WriteLine($"Среднее время выполнения: {System.Math.Round(TAT.Average(), 2)}");

System.Console.WriteLine($"Среднее время ожидания: {System.Math.Round(waitTime.Average(), 2)}");

System.Console.WriteLine($"Время выполнения: | {TAT.Select(x => x.ToString()).Aggregate((x, y) => $"{x,-2} | {y,-2}")}|");

System.Console.WriteLine($"Время ожидания: | {waitTime.Select(x => x.ToString()).Aggregate((x, y) => $"{x,-2} | {y,-2}")}|");

System.Console.WriteLine();

}

}

}

**Файл Algorithms.cs:**

using System.Collections.Generic;

using System.Linq;

using System;

namespace Lab\_6

{

public class Algorithms

{

private const int timeQuantum = 3;

public static void FCFS(int[] burstTime)

{

Console.WriteLine("FCFS (Прямой)");

int[] TAT = new int[burstTime.Length];

int[] waitTime = new int[burstTime.Length];

waitTime[0] = 0;

TAT[0] = burstTime[0];

for (int i = 1; i < burstTime.Length; i++)

{

waitTime[i] = TAT[i - 1];

TAT[i] = waitTime[i] + burstTime[i];

}

Print.Info(TAT, waitTime);

}

public static void ReverseFCFS(int[] burstTime)

{

Console.WriteLine("FCFS (обратный)");

int[] TAT = new int[burstTime.Length];

int[] waitTime = new int[burstTime.Length];

waitTime[burstTime.Length - 1] = 0;

TAT[burstTime.Length - 1] = burstTime.Last();

for (int i = burstTime.Length - 2; i >= 0; i--)

{

waitTime[i] = TAT[i + 1];

TAT[i] = waitTime[i] + burstTime[i];

}

Print.Info(TAT, waitTime);

}

public static void RoundRobin(int[] burstTime)

{

Console.WriteLine("Round robin");

int[] time = burstTime.ToArray();

int[] TAT = new int[burstTime.Length];

int[] waitTime = new int[burstTime.Length];

int current = 0;

bool stop = false;

while (!stop)

{

if (time[current] > 0)

{

for (int k = 0; k < timeQuantum; k++)

{

time[current]--;

TAT[current]++;

for (int i = 0; i < burstTime.Length; i++)

{

if (i != current && time[i] != 0)

{

waitTime[i]++;

TAT[i]++;

}

}

if (time[current] == 0)

{

break;

}

}

}

stop = true;

for (int i = 0; i < burstTime.Length; i++)

{

if (time[i] > 0)

{

stop = false;

break;

}

}

current = (current + 1) % burstTime.Length;

}

Print.Info(TAT, waitTime);

}

public static void SJFNonPreemtive(int[] burstTime)

{

Console.WriteLine("SJF (невытесняющий)");

int[] TAT = new int[burstTime.Length];

int[] waitTime = new int[burstTime.Length];

int i = 0;

int[] psExecOrderIndex = burstTime.Select(t => new

{

time = t,

psIndex = i++,

})

.OrderBy(x => x.time)

.Select(x => x.psIndex)

.ToArray();

int pastTime = 0;

for (i = 0; i < psExecOrderIndex.Length; i++)

{

int currentProcess = psExecOrderIndex[i];

waitTime[currentProcess] = pastTime;

TAT[currentProcess] = pastTime + burstTime[currentProcess];

pastTime += burstTime[currentProcess];

}

Print.Info(TAT, waitTime);

}

class SJFProcess

{

private int currentTime = 0;

public int QueupastTimeT { get; private set; }

public int ExecDurationTime { get; private set; }

public int Priority { get; set; }

public int Index { get; private set; }

public bool IsExecuted => ExecDurationTime == currentTime;

public bool IsNeverExecute => currentTime == 0;

public SJFProcess(int index, int execDurationTime, int queupastTimeT, int priority)

{

Index = index;

QueupastTimeT = queupastTimeT;

ExecDurationTime = execDurationTime;

Priority = priority;

}

public bool CanExecute(int currentTime)

{

if (IsExecuted)

{

return false;

}

if (currentTime >= QueupastTimeT)

{

return true;

}

else

{

return false;

}

}

public void Execute(int quant = 1)

{

currentTime += quant;

if (currentTime > ExecDurationTime)

{

currentTime = ExecDurationTime;

}

}

public int RemainingExecutionTime()

{

return ExecDurationTime - currentTime;

}

}

public static void SJFNonPreemtiveWithPriority(int[] burstTime, int[] priority)

{

Console.WriteLine("SJF (Невытесняющий приоритетный)");

int[] TAT = new int[burstTime.Length];

int[] waitTime = new int[burstTime.Length];

var processes = new List<SJFProcess>();

for (int i = 0; i < burstTime.Length; i++)

{

processes.Add(new SJFProcess(i, burstTime[i], 0, priority[i]));

}

for (int time = 0; processes.All(x => x.IsExecuted) == false;)

{

var psList = processes.Where(x => x.CanExecute(time));

if (psList.Any())

{

var minPriority = psList.Min(x => x.Priority);

var minPriorProcesses = psList.Where(x => x.Priority == minPriority);

var process = processes.First(x => x.ExecDurationTime == minPriorProcesses.Min(t => t.ExecDurationTime));

if (process.IsNeverExecute)

{

waitTime[process.Index] = time - process.QueupastTimeT;

}

process.Execute(process.ExecDurationTime);

TAT[process.Index] = (time + process.ExecDurationTime) - process.QueupastTimeT;

time += process.ExecDurationTime;

}

else

{

time++;

}

}

Print.Info(TAT, waitTime);

}

public static void SJFPreemtive(int[] burstTime, int[] arrivalTime)

{

Console.WriteLine("SJF (Вытесняющий)");

var allProc = new List<Process>();

var procList = new List<Process>();

var procTime = new SortedDictionary<int, int>();

var waitTime = new SortedDictionary<int, int>();

for (int i = 0; i < burstTime.Length; i++)

{

allProc.Add(new Process

{

waitTime = 0,

ProcTime = 0,

TimeLeft = burstTime[i],

Ready = arrivalTime[i],

Index = i

});

}

FindNextProc(allProc, procList);

int pointer = 0;

while (procList.Count != 0)

{

procList = procList.OrderBy(x => x.TimeLeft).ThenBy(x => x.Index).ToList();

procList.First().InProcess();

foreach (var i in procList.Skip(1))

{

i.Wait();

}

foreach (var i in allProc)

{

i.Wait();

}

if (procList.First().TimeLeft < 1)

{

procTime.Add(procList.First().Index, procList.First().ProcTime);

waitTime.Add(procList.First().Index, procList.First().waitTime);

procList.RemoveAt(0);

pointer++;

}

FindNextProc(allProc, procList);

}

int[] TATarray = procTime.Select(x => x.Value).ToArray();

for (int i = 0; i < burstTime.Length; i++)

{

TATarray[i] -= arrivalTime[i];

}

int[] waitTimeArray = waitTime.Select(x => x.Value).ToArray();

for (int i = 0; i < burstTime.Length; i++)

{

waitTimeArray[i] -= arrivalTime[i];

}

Print.Info(TATarray, waitTimeArray);

}

public static void SJFPreemtiveWithPriority(int[] burstTime, int[] arrivalTime, int[] priority)

{

Console.WriteLine("SJF (Вытесняющий приоритетный)");

var allProc = new List<Process>();

var procList = new List<Process>();

var procTime = new SortedDictionary<int, int>();

var waitTime = new SortedDictionary<int, int>();

for (int i = 0; i < burstTime.Length; i++)

{

allProc.Add(new Process

{

waitTime = 0,

Prior = priority[i],

ProcTime = 0,

TimeLeft = burstTime[i],

Ready = arrivalTime[i],

Index = i

});

}

FindNextProcPrior(allProc, procList);

int pointer = 0;

while (procList.Count != 0)

{

procList = procList.OrderBy(x => x.Prior).ThenBy(x => x.TimeLeft).ToList();

procList.First().InProcess();

foreach (var i in procList.Skip(1))

{

i.Wait();

}

foreach (var i in allProc)

{

i.Wait();

}

if (procList.First().TimeLeft < 1)

{

procTime.Add(procList.First().Index, procList.First().ProcTime);

waitTime.Add(procList.First().Index, procList.First().waitTime);

procList.RemoveAt(0);

pointer++;

}

FindNextProcPrior(allProc, procList);

}

int[] TATarray = procTime.Select(x => x.Value).ToArray();

for (int i = 0; i < burstTime.Length; i++)

{

TATarray[i] -= arrivalTime[i];

}

int[] waitTimeArray = waitTime.Select(x => x.Value).ToArray();

for (int i = 0; i < burstTime.Length; i++)

{

waitTimeArray[i] -= arrivalTime[i];

}

Print.Info(TATarray, waitTimeArray);

}

private static void FindNextProc(List<Process> allProc, List<Process> procList)

{

for (int i = 0; i < allProc.Count; i++)

{

if (allProc[i].Ready < 1)

{

if (procList.Count == 0)

{

procList.Add(allProc[i]);

allProc.RemoveAt(i);

i--;

}

else

{

int k = 0;

bool check = true;

foreach (var j in procList)

{

int test = j.TimeLeft - allProc[i].TimeLeft;

if (j.TimeLeft > allProc[i].TimeLeft)

{

procList.Insert(k, allProc[i]);

allProc.RemoveAt(i);

i--;

check = false;

break;

}

if (j.TimeLeft == allProc[i].TimeLeft)

{

check = false;

break;

}

k++;

}

if (check)

{

procList.Add(allProc[i]);

allProc.RemoveAt(i);

i--;

}

}

}

}

}

private static void FindNextProcPrior(List<Process> allProc, List<Process> procList)

{

for (int i = 0; i < allProc.Count; i++)

{

if (allProc[i].Ready < 1)

{

if (procList.Count == 0)

{

procList.Add(allProc[i]);

allProc.RemoveAt(i);

i--;

}

else

{

int k = 0;

bool check = true;

foreach (var j in procList)

{

if (j.Prior < allProc[i].Prior)

{

procList.Insert(k, allProc[i]);

allProc.RemoveAt(i);

i--;

check = false;

break;

}

k++;

}

if (check)

{

procList.Add(allProc[i]);

allProc.RemoveAt(i);

i--;

}

}

}

}

}

}

}

**Результат выполнения:**

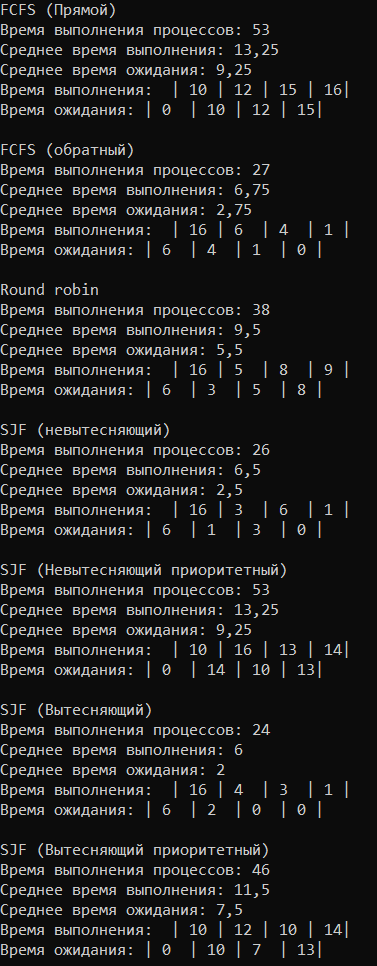


Рисунок 1 − Результат выполнения программы

**Вывод:** в ходе данной лабораторной работы были изучены типовые алгоритмы планирования процессов.