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»**

**Планирование процессов**

Выполнил студент

группы ИП-31

*Васюков И.А.*

Проверил старший преподаватель

*Самовендюк Н.В.*

Гомель 2019 г.

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

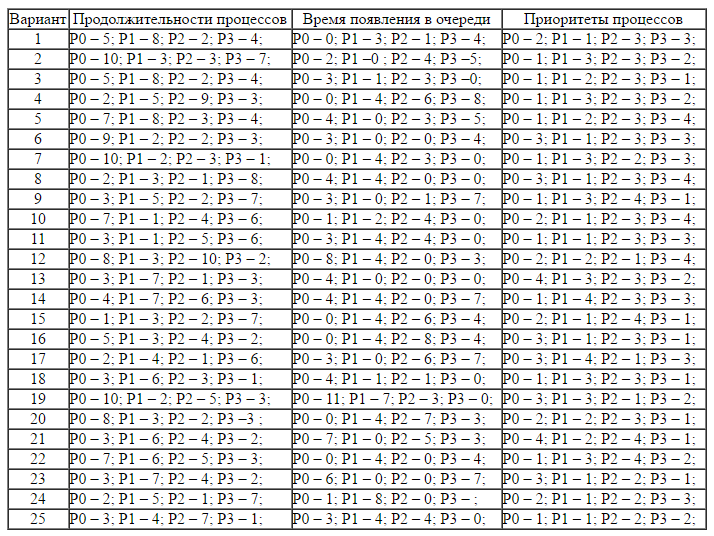
**Практическая часть:**

**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.

**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.



**Примечания:**

Для алгоритма Round Robin (RR) величина кванта времени 3 для всех вариантов.

Для приоритетных алгоритмов меньшее значение соответствует более высокому приоритету.

**Задание 3.**

Разработать программную реализацию алгоритмов задания 1 и 2.

Сравнить полученные результаты и сделать выводы.

**Код программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace OC\_LAB\_6

{

class Proc: IComparable<Proc>

{

public int WaitTime { get; set; }

public int BurstTime { get; set; }

public static int Capacity { get; set; }

public int G { get; set; }

public int I { get; set; }

public int AppearTime { get; set; }

public Proc(int burst,int appear)

{

BurstTime = burst;

AppearTime = appear;

}

public int CompareTo(Proc other)

{

return AppearTime.CompareTo(other.AppearTime);

}

}

class Program

{

static int[] p\_t = new int[] { 2, 5, 9, 3 };//birt

static int[] p\_ap = new int[] { 0, 4, 6, 8 };//per

static int[] p\_pr = new int[] { 1, 3, 3, 2 };

static void Main(string[] args)

{

List<Proc> processes = new List<Proc>();

processes.Add(new Proc(2,0)); processes.Add(new Proc(5,4)); processes.Add(new Proc(9,6)); processes.Add(new Proc(3,8));

Console.WriteLine("FCFS - прямой\n-----------------");

FCFS\_Direct(p\_t, p\_ap,processes);

Out(processes);

Console.WriteLine();

Console.WriteLine("FCFS - обратный\n-----------------");

FCFS\_Reverse(p\_t, p\_ap, processes);

Console.WriteLine();

Out(processes);

Console.WriteLine("RR\n-----------------");

RR(p\_t, p\_ap, processes);

Out(processes);

Console.WriteLine();

Console.WriteLine("SJF невытесняющий, без приоритета\n-----------------");

SJF\_NoDisloge\_NoPriority(p\_t, p\_ap,processes);

Out(processes);

Console.WriteLine();

Console.WriteLine("SJF невытесняющий, с приоритетом\n-----------------");

SJF\_NoDisloge\_Priority(p\_t, p\_ap, p\_pr, processes);

Out(processes);

Console.WriteLine();

Console.WriteLine("SJF вытесняющий, без приоритета\n-----------------");

SJF\_Disloge\_NoPriority(p\_t, p\_ap,processes);

Out(processes);

Console.WriteLine();

Console.WriteLine("SJF вытесняющий, с приоритетом\n-----------------");

SJF\_Disloge\_Priority(p\_t, p\_ap, p\_pr, processes);

Out(processes);

Console.WriteLine();

Console.ReadLine();

}

static void FCFS\_Direct(int[] p\_t, int[] p\_app,List<Proc> processes)

{

foreach (var pr in processes)

{

pr.G = 0;

pr.I = 0;

}

var p = p\_t.ToArray(); //copy array

var workedIndex = -1;

var proccessQueue = new Queue<int>();

for (var i = 0; p.Sum() != 0; i++)

{

for (var j = 0; j < p\_ap.Length; j++)

{

if (workedIndex >= 0 && p\_ap[j] == i)

proccessQueue.Enqueue(j); //добавить в конец

else if (workedIndex < 0 && p\_ap[j] == i && proccessQueue.Count == 0)

workedIndex = j;

else if (workedIndex < 0 && p\_ap[j] == i && proccessQueue.Count != 0)

{

proccessQueue.Enqueue(j);

workedIndex = proccessQueue.Dequeue(); //достать из начала

}

else if (workedIndex < 0 && p\_ap[j] != i && proccessQueue.Count != 0)

workedIndex = proccessQueue.Dequeue();

}

PrintLine(FormArr(p, p\_ap, i), workedIndex,processes);

if (workedIndex >= 0)

{

p[workedIndex]--;

if (p[workedIndex] == 0)

workedIndex = -1;

}

}

}

static void FCFS\_Reverse(int[] p\_t, int[] p\_app,List<Proc> procs)

{

foreach(var pr in procs)

{

pr.G = 0;

pr.I = 0;

}

var p = p\_t.ToArray();

var workedIndex = -1;

var proccessStack = new Stack<int>();

for (var i = 0; p.Sum() != 0; i++)

{

for (var j = p\_ap.Length - 1; j >= 0; j--)

{

if (workedIndex >= 0 && p\_ap[j] == i)

{

proccessStack.Push(j);

}

else if (workedIndex < 0 && p\_ap[j] == i && proccessStack.Count == 0)

workedIndex = j;

else if (workedIndex < 0 && p\_ap[j] == i && proccessStack.Count != 0)

{

workedIndex = proccessStack.Pop();

proccessStack.Push(j);

}

else if (workedIndex < 0 && p\_ap[j] != i && proccessStack.Count != 0)

workedIndex = proccessStack.Pop();

}

PrintLine(FormArr(p, p\_ap, i), workedIndex,procs);

if (workedIndex >= 0)

{

p[workedIndex]--;

if (p[workedIndex] == 0)

workedIndex = -1;

}

}

int a = 0;

}

static void RR(int[] p\_t, int[] p\_app,List<Proc> procs)

{

foreach(var pr in procs)

{

pr.I = 0;

pr.G = 0;

}

var p = p\_t.ToArray();

var workedIndex = -1;

var workedTime = 0;

var proccessQueue = new Queue<int>();

for (var i = 0; p.Sum() != 0; i++)

{

for (var j = 0; j < p\_ap.Length; j++)

{

if (workedIndex >= 0 && p\_ap[j] == i)

proccessQueue.Enqueue(j);

else if (workedIndex < 0 && p\_ap[j] == i && proccessQueue.Count == 0)

{

workedIndex = j;

workedTime = 3;

}

else if (workedIndex < 0 && p\_ap[j] == i && proccessQueue.Count != 0)

{

proccessQueue.Enqueue(j);

workedIndex = proccessQueue.Dequeue();

workedTime = 3;

}

else if (workedIndex < 0 && p\_ap[j] != i && proccessQueue.Count != 0)

{

workedIndex = proccessQueue.Dequeue();

workedTime = 3;

}

}

PrintLine(FormArr(p, p\_ap, i), workedIndex, procs);

if (workedIndex >= 0)

{

workedTime--;

p[workedIndex]--;

if (p[workedIndex] == 0)

workedIndex = -1;

else if (workedTime == 0)

{

proccessQueue.Enqueue(workedIndex);

workedIndex = -1;

}

}

}

}

static void SJF\_NoDisloge\_NoPriority(int[] p\_t, int[] p\_app,List<Proc> procs)

{

foreach(var item in procs)

{

item.G = 0;

item.I = 0;

}

var p = p\_t.ToArray();

var workedIndex = -1;

var proccessList = new List<int>();

for (var i = 0; p.Sum() != 0; i++)

{

for (var j = 0; j < p\_ap.Length; j++)

{

if (workedIndex >= 0 && p\_ap[j] == i)

proccessList.Insert(FindPosition\_NonPriority(p[j], proccessList, p), j);

else if (workedIndex < 0 && p\_ap[j] == i && proccessList.Count == 0)

workedIndex = j;

else if (workedIndex < 0 && p\_ap[j] == i && proccessList.Count != 0)

{

proccessList.Insert(FindPosition\_NonPriority(p[j], proccessList, p), j);

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

else if (workedIndex < 0 && p\_ap[j] != i && proccessList.Count != 0)

{

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

}

PrintLine(FormArr(p, p\_ap, i), workedIndex,procs);

if (workedIndex >= 0)

{

p[workedIndex]--;

if (p[workedIndex] == 0)

workedIndex = -1;

}

}

}

static void SJF\_NoDisloge\_Priority(int[] p\_t, int[] p\_app, int[] p\_pr,List<Proc> procs)

{

foreach (var item in procs)

{

item.G = 0;

item.I = 0;

}

var p = p\_t.ToArray();

var workedIndex = -1;

var proccessList = new List<int>();

for (var i = 0; p.Sum() != 0; i++)

{

for (var j = 0; j < p\_ap.Length; j++)

{

if (workedIndex >= 0 && p\_ap[j] == i)

proccessList.Insert(FindPosition\_Priority(p\_pr[j], proccessList, p\_pr), j);

else if (workedIndex < 0 && p\_ap[j] == i && proccessList.Count == 0)

workedIndex = j;

else if (workedIndex < 0 && p\_ap[j] == i && proccessList.Count != 0)

{

proccessList.Insert(FindPosition\_Priority(p\_pr[j], proccessList, p\_pr), j);

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

else if (workedIndex < 0 && p\_ap[j] != i && proccessList.Count != 0)

{

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

}

PrintLine(FormArr(p, p\_ap, i), workedIndex,procs);

if (workedIndex >= 0)

{

p[workedIndex]--;

if (p[workedIndex] == 0)

workedIndex = -1;

}

}

}

static void SJF\_Disloge\_NoPriority(int[] p\_t, int[] p\_app,List<Proc> procs)

{

foreach (var item in procs)

{

item.G = 0;

item.I = 0;

}

var p = p\_t.ToArray();

var workedIndex = -1;

var proccessList = new List<int>();

for (var i = 0; p.Sum() != 0; i++)

{

for (var j = 0; j < p\_ap.Length; j++)

{

if (workedIndex >= 0 && p\_ap[j] == i)

{

proccessList.Insert(FindPosition\_NonPriority(p[j], proccessList, p), j);

proccessList.Insert(FindPosition\_NonPriority(p[workedIndex], proccessList, p), workedIndex);

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

else if (workedIndex < 0 && p\_ap[j] == i && proccessList.Count == 0)

workedIndex = j;

else if (workedIndex < 0 && p\_ap[j] == i && proccessList.Count != 0)

{

proccessList.Insert(FindPosition\_NonPriority(p[j], proccessList, p), j);

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

else if (workedIndex < 0 && p\_ap[j] != i && proccessList.Count != 0)

{

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

}

//PrintLine(FormArr(p, p\_ap, i), workedIndex,procs);

if (workedIndex >= 0)

{

p[workedIndex]--;

if (p[workedIndex] == 0)

workedIndex = -1;

}

}

}

static void SJF\_Disloge\_Priority(int[] p\_t, int[] p\_app, int[] p\_pr,List<Proc> procs)

{

foreach (var item in procs)

{

item.G = 0;

item.I = 0;

}

var p = p\_t.ToArray();

var workedIndex = -1;

var proccessList = new List<int>();

for (var i = 0; p.Sum() != 0; i++)

{

for (var j = 0; j < p\_ap.Length; j++)

{

if (workedIndex >= 0 && p\_ap[j] == i)

{

proccessList.Insert(FindPosition\_Priority(p\_pr[j], proccessList, p\_pr), j);

proccessList.Insert(FindPosition\_Priority(p\_pr[workedIndex], proccessList, p\_pr), workedIndex);

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

else if (workedIndex < 0 && p\_ap[j] == i && proccessList.Count == 0)

workedIndex = j;

else if (workedIndex < 0 && p\_ap[j] == i && proccessList.Count != 0)

{

proccessList.Insert(FindPosition\_Priority(p\_pr[j], proccessList, p\_pr), j);

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

else if (workedIndex < 0 && p\_ap[j] != i && proccessList.Count != 0)

{

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

}

PrintLine(FormArr(p, p\_ap, i), workedIndex,procs);

if (workedIndex >= 0)

{

p[workedIndex]--;

if (p[workedIndex] == 0)

workedIndex = -1;

}

}

}

static int FindPosition\_NonPriority(int time, List<int> list, int[] p\_t)

{

for (var i = 0; i < list.Count; i++)

{

if (time < p\_t[list[i]])

return i;

}

return list.Count;

}

static int FindPosition\_Priority(int prior, List<int> list, int[] p\_p)

{

for (var i = 0; i < list.Count; i++)

{

if (prior <= p\_p[list[i]])

return i;

}

return list.Count;

}

static int[] FormArr(int[] p, int[] p\_ap, int time)

{

var result = new List<int>();

for (var i = 0; i < p.Length; i++)

{

if (p[i] == 0 || time < p\_ap[i])

result.Add(0);

else

result.Add(1);

}

return result.ToArray();

}

static void PrintLine(int[] p, int procNum,List<Proc> procs)

{

Proc.Capacity++;

var line = "|";

for (var i = 0; i < p.Length; i++)

{

if (p[i] == 0)

line += " |";

else if (procNum == i)

{

procs[i].I++;

line += " И |";

}

else

{

line += " Г |";

procs[i].G++;

}

}

Console.WriteLine(line);

Console.WriteLine("-----------------");

}

public static void Out(List<Proc> proc)

{

int sumBurst = 0;

int sumIG = 0;

StringBuilder countIG\_str = new StringBuilder();

StringBuilder countG\_str = new StringBuilder();

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

{

sumBurst += proc[i].BurstTime;

countIG\_str.Append($"p{i}={proc[i].I + proc[i].G} ");

countG\_str.Append($"p{i}={proc[i].G} ");

sumIG += proc[i].I + proc[i].G;

}

Console.WriteLine($"Полное время выполнения всех процессов: {sumIG}");

Console.WriteLine($"Время выполнения каждого процесса: {countIG\_str}");

Console.WriteLine($"Время ожидания каждого процесса: {countG\_str}");

Console.WriteLine($"Среднее время выполнения процесса: {(double)sumIG / proc.Count}");

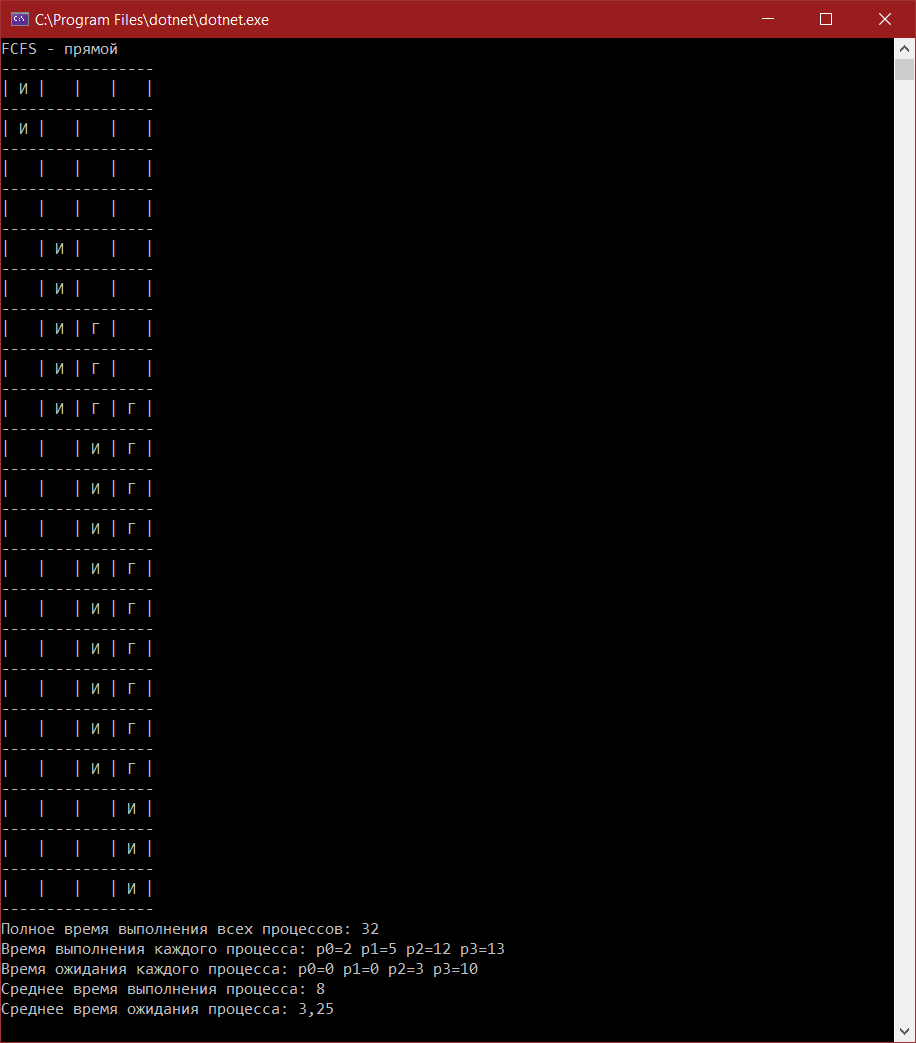
Console.WriteLine($"Среднее время ожидания процесса: { (double)proc.Sum(w => w.G) / proc.Count}");

}

}

}

Результат выполнения



**Вывод:** В данной работе были изучены типовые алгоритмы планирования процессов.