# Министерство Образования Республики Беларусь

Учреждение образования

«Гомельский государственный технический университет

имени П.О.Сухого»

Кафедра: «Информатика»

по курсу: «Операционные системы и среды»

Лабораторная работа №6

«Планирование процессов»

Выполнил: студент группы ИП-31

Земчёнок Вадим Геннадьевич

Допуск к защите: Проверил: старший преподаватель

Дата защиты: Самовендюк Николай Владимирович

Гомель 2020 г.

**Цель:** Изучение типовых алгоритмов планирования процессов

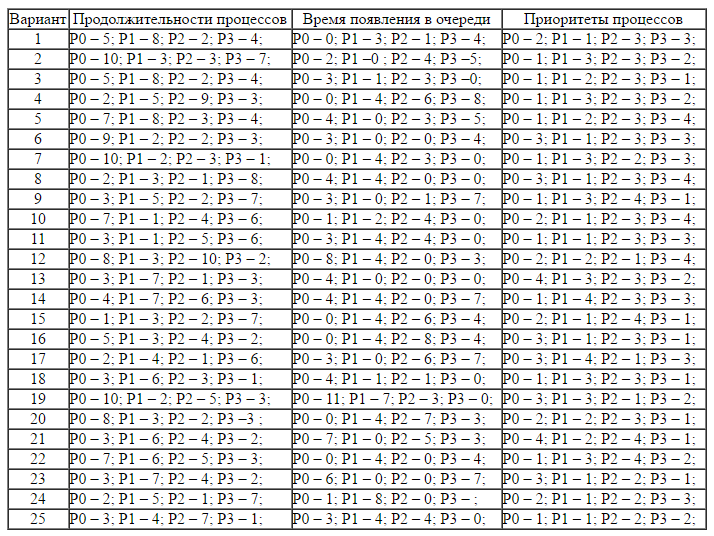
**Задание 1. Не вытесняющие алгоритмы планирования процессов**

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.

**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.

Таблица 2.1 Варианты заданий



**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.

**Листинг программы**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace app

{

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[] { 10, 2, 3, 1 }; //birt

static int[] p\_app = new int[] { 0, 4, 3, 0 }; //per

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

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

static void Main(string[] args)

{

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

processes.Add(new Proc(7, 4)); processes.Add(new Proc(8, 0)); processes.Add(new Proc(3, 3)); processes.Add(new Proc(4, 5));

Console.WriteLine("FCFS - прямой");

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

Out(processes);

Console.WriteLine();

Console.WriteLine("FCFS - обратный");

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

Out(processes);

Console.WriteLine();

Console.WriteLine("RR");

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

Out(processes);

Console.WriteLine();

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

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

Out(processes);

Console.WriteLine();

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

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

Out(processes);

Console.WriteLine();

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

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

Out(processes);

Console.WriteLine();

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

SJF\_Disloge\_Priority(p\_t, p\_app, 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\_app.Length; j++)

{

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

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

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

workedIndex = j;

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

{

proccessQueue.Enqueue(j);

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

}

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

workedIndex = proccessQueue.Dequeue();

}

PrintLine(i, FormArr(p, p\_app, 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\_app.Length - 1; j >= 0; j--)

{

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

{

proccessStack.Push(j);

}

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

workedIndex = j;

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

{

workedIndex = proccessStack.Pop();

proccessStack.Push(j);

}

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

workedIndex = proccessStack.Pop();

}

PrintLine(i, FormArr(p, p\_app, 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\_app.Length; j++)

{

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

proccessQueue.Enqueue(j);

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

{

workedIndex = j;

workedTime = 3; //ВЕЛИЧИНА КВАНТА ВРЕМЕНИ, ЗАДАННАЯ УСЛОВИЕМ!

}

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

{

proccessQueue.Enqueue(j);

workedIndex = proccessQueue.Dequeue();

workedTime = 3;

}

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

{

workedIndex = proccessQueue.Dequeue();

workedTime = 3;

}

}

PrintLine(i, FormArr(p, p\_app, 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\_app.Length; j++)

{

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

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

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

workedIndex = j;

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

{

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

}

PrintLine(i, FormArr(p, p\_app, 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\_app.Length; j++)

{

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

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

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

workedIndex = j;

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

{

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

}

PrintLine(i, FormArr(p, p\_app, 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\_app.Length; j++)

{

if (workedIndex >= 0 && p\_app[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\_app[j] == i && proccessList.Count == 0)

workedIndex = j;

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

{

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

}

PrintLine(i, FormArr(p, p\_app, 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\_app.Length; j++)

{

if (workedIndex >= 0 && p\_app[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\_app[j] == i && proccessList.Count == 0)

workedIndex = j;

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

{

workedIndex = proccessList.First();

proccessList.RemoveAt(0);

}

}

PrintLine(i, FormArr(p, p\_app, 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 no, int[] p, int procNum, List<Proc> procs)

{

Proc.Capacity++;

var line = no + "\t|";

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);

}

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($"Сумма И+Г всех процессов = {sumIG}");

Console.WriteLine($"И+Г каждого процесса: {countIG\_str}");

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

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

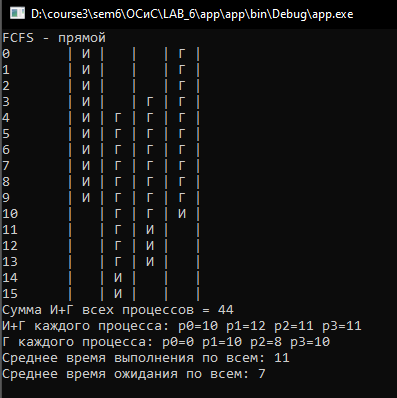
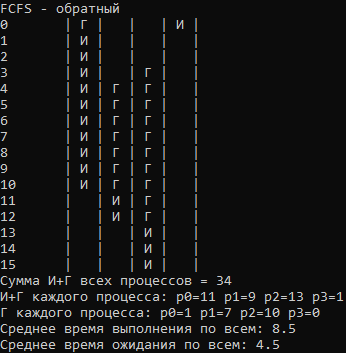
Console.WriteLine($"Среднее время ожидания по всем: { (double)proc.Sum(w => w.G) / proc.Count}");

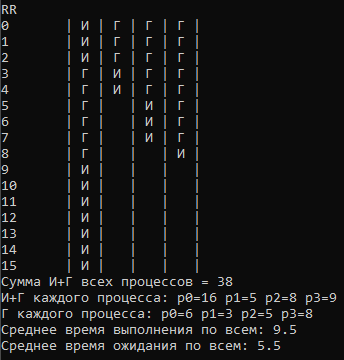
}

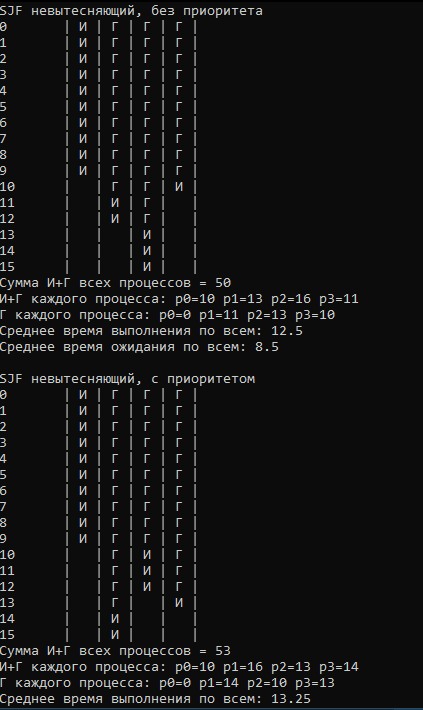
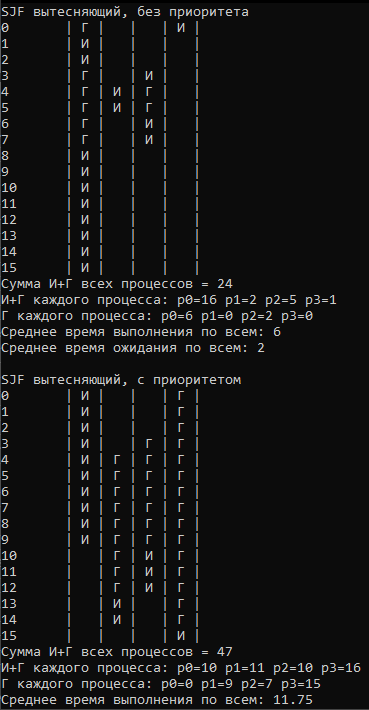
}

}

|  |  |  |  |
| --- | --- | --- | --- |
| Вар | Продолжительности процессов | Время появления в очереди | Приоритеты |
| 7 | P0 – 10, P1 – 2, P2 – 3, P3 – 1 | P0 – 0, P1 – 4, P2 – 3, P3 – 0 | P0 – 1, P1 – 3, P2 – 2, P3 – 3 |



** **

**Вывод:** я успешно изучил типовые алгоритмы планирования процессов.