Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики

**Лабораторная работа №1**

**Дисциплина «Операционные системы»**

**Выполнил:**

Группа P3319

**Преподаватель:**

Лаздин Артур Вячеславович

Санкт-Петербург

2019

**Задание**

В соответствии с вариантом задания реализовать программную имитацию реализации алгоритма планирования. Программно подготовить исходные данные, представляющие собой пары чисел, где первое число – время поступления процесса, а второе – время его выполнения.

Для полученных данных определить среднее время оборота процессов и среднее время ожидания процессов в соответствии с реализованными алгоритмами планирования.  
Сделать вывод о предпочтительном алгоритме планирования, считая критерием эффективности **минимальное среднее время ожидания**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варианта | Алгоритм 1 | Алгоритм 2 | Диапазон длительностей выполнения | Среднее время между поступающими процессами |
| 16 | **FCFS** | **RR(4)** | 6÷12 | 7±3 |

Генерация исходных данных:

private void CreateProcesses()

{

Processes = new Process[PROCESS\_COUNT];

int time = 0;

for (int i = 0; i < PROCESS\_COUNT; i++)

{

int execTime = random.Next(MinExecTime, MaxExecTime + 1);

Processes[i] = new Process(time, execTime);

time += random.Next(MinGenerationTime, MaxGenerationTime + 1);

}

}

Алгоритм FCFS:

private SimulationResult RunFCFS()

{

double totalWaitingTime = 0;

int maxQueueLength = 0;

var queue = new Queue<Process>();

int remainingProcesses = PROCESS\_COUNT;

int nextProcessIndex = 0;

int time = 0;

while (remainingProcesses > 0)

{

//Появление новых процессов

while (nextProcessIndex < PROCESS\_COUNT

&& Processes[nextProcessIndex].ArrivalTime <= time)

{

Process nextProcess = Processes[nextProcessIndex];

queue.Enqueue(nextProcess);

nextProcessIndex++;

}

//Обработка процесса в очереди

if (queue.Count > 0)

{

maxQueueLength = Math.Max(maxQueueLength, queue.Count);

var p = queue.Dequeue();

totalWaitingTime += time - p.ArrivalTime;

time += p.RemainingTime;

remainingProcesses--;

}

else

{

Process nextProcess = Processes[nextProcessIndex];

time = nextProcess.ArrivalTime;

}

}

double avgWaitingTime = totalWaitingTime / PROCESS\_COUNT;

return new SimulationResult(avgWaitingTime, maxQueueLength);

}

Алгоритм Round Robin:

private SimulationResult RunRoundRobin()

{

double totalWaitingTime = 0;

int maxQueueLength = 0;

var queue = new Queue<Process>();

int remainingProcesses = PROCESS\_COUNT;

int iterationCount = 0;

int nextProcessIndex = 0;

int time = 0;

while (remainingProcesses > 0)

{

//Появление новых процессов

while (nextProcessIndex < PROCESS\_COUNT

&& Processes[nextProcessIndex].ArrivalTime <= time)

{

Process nextProcess = Processes[nextProcessIndex];

queue.Enqueue(nextProcess);

nextProcessIndex++;

}

//Обработка процесса в очереди

if (queue.Count > 0)

{

maxQueueLength = Math.Max(maxQueueLength, queue.Count);

var p = queue.Dequeue();

int deltaT = Math.Min(RoundRobinTime, p.RemainingTime);

iterationCount++;

totalWaitingTime += time - p.ArrivalTime;

time += deltaT;

p.Advance(time, deltaT);

if (p.IsFinished) remainingProcesses--;

else queue.Enqueue(p);

}

else

{

Process nextProcess = Processes[nextProcessIndex];

time = nextProcess.ArrivalTime;

}

}

Debug.Assert(iterationCount >= PROCESS\_COUNT);

double avgWaitingTime = totalWaitingTime / iterationCount;

return new SimulationResult(avgWaitingTime, maxQueueLength);

}

Результат работы программы:

Среднее время ожидания для FCFS: 53,72

Макс. длина очереди для FCFS: 11

Среднее время ожидания для RR: 30,7045454545455

Макс. длина очереди для RR: 18

Результат: RR более эффективен.

**Вывод:** при использовании алгоритма Round Robin процесс меньше ожидает своей очереди, поскольку процессор уделяет каждому процессу перед ним не более одного кванта времени и они выполняются как бы параллельно.

При этом длина очереди может быть больше, чем при использовании алгоритма FCFS, так как процессы могут снова возвращаться в очередь.