**ЛАБОРАТОРНА РОБОТА #3**

З дисципліни «Системне програмне забезпечення»

На тему «Дослідження заявок обслуговування при обмежених ресурсах»

|  |  |
| --- | --- |
| **Виконав**  Студент групи КВ-21  Комарницький О.Б. | **Перевірив**  Проф. Симоненко В.П. |

|  |  |  |
| --- | --- | --- |
| **Оцінка** | **Дата** | **Підпис** |
|  |  |  |

**Варіант #9**

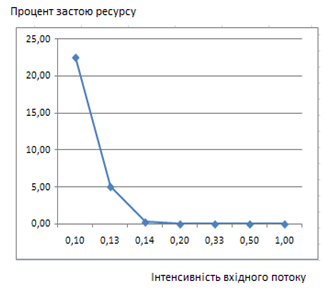
**Завдання:**

1. Згідно варіанту виконати по тактову візуалізацію системи обслуговування заявок з розподіленим ресурсом, параметрами системи є діапазон ваг заявок, діапазон інтервалів виникнення наступної заявки (інтенсивність вхідного потоку) , діапазон пріоритетів (для пріоритетних дисциплін обслуговування). В систему може надходити будь-яка кількість заявок.
2. Якщо система обслуговування з чергами, то кількість черг не більша 32. Якщо система обслуговування з пріоритетами, то число пріоритетів не більш ніж 32.
3. *Побудувати графіки залежності середнього часу очікування від інтенсивності вхідного потоку заявок та залежність проценту простою ресурсу від інтенсивності вхідного потоку заявок.*
4. *Для пріоритетних систем побудувати графік залежності середнього часу очікування від пріоритету при фіксованій інтенсивності вхідного потоку заявок*. Для систем без пріоритетів побудувати графік залежності кількості заявок від часу очікування при фіксованій інтенсивності вхідного потоку заявок.
5. *Пояснити форму графіків.*

|  |  |
| --- | --- |
| **Варіант #** | **Алгоритм** |
| 9 | Алгоритм RR (*Round Robin*) з пріоритетами, без витіснення |

**Графіки залежності**

Середній час очікування = f(інтенсивність вхідного потоку) Процент застою = f(інтенсивність вхідного потоку)



**Висновок:**

1. Графік залежності середнього часу очікування від інтенсивності вхідного потоку.

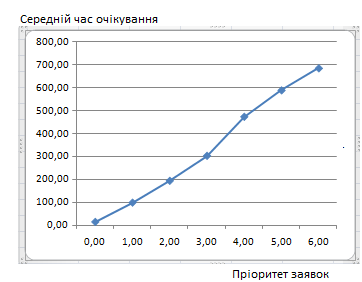
При збільшенні інтенсивності вхідного потоку збільшується середній час очікування заявок, що пояснюється більшим завантаженням процесора. На ділянці області визначення, де процент застою процесорного ресурсу різко падає до нуля, спостерігається параболічна залежність росту середнього часу очікування від інтенсивності вхідного потоку, що пояснюється значним навантаженням системи чергою вхідних заявок.

Коли ж процент застою ресурсу впав до нуля, бачимо лінійну залежність, тобто кожне наступне значення середнього часу очікування буде більше на величину кількості заявок.

1. Графік залежності проценту застою ресурсу процесора від інтенсивності вхідного потоку.

При збільшенні інтенсивності вхідного потоку зменшується процент застою процесорного ресурсу, адже чим більше прийшло заявок, тим менше процесор буде застоюватись.

Крім того, помітно межу, де процесор повністю перенавантажується від потоку вхідних заявок.

****Середній час очікування = f(пріоритет заявок)

**Висновок:**

На графіку зображено 7 пріоритетів (від 0 – найбільшого до 6 – найменшого). Як помітно із графіка – чим більший пріоритет, тим менший час очікування.

Графік лінійний, адже середня кількість і складність заявок одного пріоритету приблизно однакова, тому збільшення середнього часу очікування буде лінійним, тобто на час опрацювання заявок даного пріоритету.

**Лістинг мовою С++**

***Файл Interface.h***

struct Proc

{

unsigned int ID;

unsigned int complexity;

unsigned int time\_waited;

unsigned int time\_joined;

unsigned int time\_done;

unsigned int priority;

int time\_left;

};

struct Settings

{

unsigned int tasks\_count = 100; // 40

const unsigned int INTENSITY\_PERIOD = 1;

const unsigned int MAX\_COMPLEX = 15; //30

const unsigned int MAX\_PRIO = 3; // 7

const unsigned int QUANTUM = 1;

const unsigned int DELAY = 1;

};

void run();

void outfile();

***Файл Interface.cpp***

#include "Interface.h"

#include <vector>

#include <Windows.h>

#include <fstream>

using namespace std;

vector<Proc \*> get\_best\_prio();

unsigned int index\_by\_id(unsigned int id);

void addProc(unsigned int & tact, unsigned int & id);

void output();

vector<Proc> Queue;

vector<Proc> ListCompleted;

Settings settings;

unsigned int UNUSED = 0; // Processor's time is not used

void run()

{

unsigned int tact = 0;

static unsigned int ID = 0;

vector<Proc \*> proc\_best\_prio;

unsigned int last\_priority = settings.MAX\_PRIO + 1, tmp\_index;

while (settings.tasks\_count || Queue.size())

{

// Add process with possibility

addProc(tact, ID);

// Find best priority process list

proc\_best\_prio = get\_best\_prio();

// Find next process index. Priority changed:

if (proc\_best\_prio.size() && proc\_best\_prio[0]->priority != last\_priority)

tmp\_index = 0;

else // Priority not changed

tmp\_index = (tmp\_index < 0 || tmp\_index >= proc\_best\_prio.size() - 1) ? 0 : tmp\_index + 1;

if (proc\_best\_prio.size())

{

Proc \* proc = proc\_best\_prio[tmp\_index];

proc->time\_left -= settings.QUANTUM;

last\_priority = proc->priority;

// Rest are waiting

for (auto & p : Queue)

if (p.ID != proc->ID)

p.time\_waited += settings.QUANTUM;

// Check if task completed

if (proc->time\_left <= 0)

{

proc->time\_done = tact + settings.QUANTUM + proc->time\_left;

proc->time\_left = 0;

ListCompleted.push\_back(\*proc);

Queue.erase(Queue.begin() + index\_by\_id(proc->ID));

proc\_best\_prio.erase(proc\_best\_prio.begin() + tmp\_index);

}

// Next processes arrive during the quantum

for (unsigned int t0 = tact + 1; t0 < tact + settings.QUANTUM; ++t0)

addProc(t0, ID);

tact += settings.QUANTUM;

}

else // No process in queue. Go to next tact

{

tact++;

UNUSED++;

}

output();

Sleep(settings.DELAY);

}

}

vector<Proc \*> get\_best\_prio()

{

vector<Proc \*> proc\_max\_priority;

// Find max priority;

unsigned int min = settings.MAX\_PRIO + 1;

for (auto & p : Queue)

if (p.priority < min)

min = p.priority;

for (auto & p : Queue)

if (p.priority == min)

proc\_max\_priority.push\_back(&p);

return proc\_max\_priority;

}

unsigned int index\_by\_id(unsigned int id)

{

int pos = 0;

for (auto & v : Queue)

if (v.ID == id)

return pos;

else

pos++;

return -1;

}

void addProc(unsigned int & tact, unsigned int & id)

{

if (settings.tasks\_count && tact % settings.INTENSITY\_PERIOD == 0)

{

Proc p;

p.time\_left = p.complexity = rand() % settings.MAX\_COMPLEX + 1;

p.priority = rand() % settings.MAX\_PRIO;

p.time\_joined = tact;

p.ID = id++;

p.time\_waited = 0;

Queue.push\_back(p);

settings.tasks\_count--;

}

}

***Файл Main.cpp***

#include "Interface.h"

#include <iostream>

using namespace std;

int main()

{

run();

outfile();

cout << endl << "Calculation finished. Data written" << endl;

getchar();

return 0;

}