**ЛАБОРАТОРНА РОБОТА #1**

З дисципліни «Операційні системи»

На тему «Планування процесів»

|  |  |
| --- | --- |
| **Виконав**  Студент групи КВ-21  Комарницький О.Б. | **Перевірила**  Ст. викл. кафедри СПіСКС  Дробязко І.П. |

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

**Варіант #8**

**Завдання:**

## 8. Багаторівневі черги (2 рівні)

Усі процеси можна обробляти однаково. Але це не доцільно. З точки зору роботи в пакетному режимі хороші стратегії SJF і SRTF, а для інтерактивної роботи – [RR](file:///E:\Project\ОС\4\lecture.html#rr#rr). Якщо ми відрізняємо в системі пакетні і інтерактивні процеси, тоді можна для кожної з цих груп використовувати свою стратегію планування, що відповідає саме цій групі.

Користувач може визначити *фонові (пакетні)* процеси і процеси *переднього плану* *(інтерактивні процеси)*. Фонові процеси обслуговуються згідно стратегії [SJF](file:///E:\Project\ОС\4\sjf), тому що під час роботи в пакетному режимі ми піклуємося про мінімізацію середнього часу очікування. Процеси переднього плану обслуговуються згідно [RR](file:///E:\Project\ОС\4\lecture.html#rr#rr) стратегії, оскільки у разі інтерактивної роботи ми повинні мінімізувати час відгуку.

Залишається знайти взаємовідношення між обома групами процесів. Звичайно, процесам переднього плану потрібно дати привілеї (тоді ми мінімізуємо час відгуку). Проте, до якого ступеня? Можна припустити, що ми завжди виконуємо процеси переднього плану перед пакетними в надії, що завжди існують періоди неактивності користувача. Якщо, проте, ми побоюємося голодування фонових процесів, можна розподілити час між чергою фонових процесів і процесів переднього плану. Наприклад, для інтерактивних процесів – 80 % і для фонових – 20 %.

**Для моделювання обрати 2** черги, одна зяких обслуговується за алгоритмім **LIFO**, а друга – за алгоритмім [**RR**](file:///E:\Project\ОС\4\lecture.html#rr#rr) (дивись пояснення до завдання 4 і 7).

**Код програми мовою C++**

***Main.cpp***

#include "Task.h"

#include <list>

#include <stdlib.h> // rand, system("cls")

#include <Windows.h> // sleep

#include <fstream>

using namespace std;

void addTasks(list<Task> & queue, unsigned int tact);

Settings settings;

int main()

{

list<Task> queueBackground, queueInteractive;

list<Task> logsBackround, logsInteractive;

unsigned int tact = 0, bufN = 0;

while (settings.tasksNumber || !queueBackground.empty() || !queueInteractive.empty())

{

// Adding tasks to queues

addTasks(queueBackground, tact);

addTasks(queueInteractive, tact);

// Processor works differently depends on tact

// 80% - interactive, 20% - background -> only 4th, 9th, 14th, 19th, ... are background

if ( ((tact + 1) % 5 == 0 || queueInteractive.empty()) && !queueBackground.empty() )

{

queueBackground.front().execute(1, tact);

// Delay all other processes

for (auto & t : queueBackground)

if ( !(t == queueBackground.front()))

t.delay();

for (auto & t: queueInteractive)

t.delay();

if (queueBackground.front().isPerformed())

{

logsBackround.push\_back(queueBackground.front());

queueBackground.pop\_front();

}

}

else if (!queueInteractive.empty())

{

if (bufN >= queueInteractive.size()) bufN = 0;

// get task according to RR algourithm

list<Task>::iterator it = queueInteractive.begin();

advance(it, bufN++);

(\*it).execute(1, tact);

// Delay all other processes

for (auto & t : queueInteractive)

if (!(t == \*it))

t.delay();

for (auto & t : queueBackground)

t.delay();

if ((\*it).isPerformed())

{

logsInteractive.push\_back(\*it);

queueInteractive.remove(\*it);

}

}

// Clear screen

system("cls");

// Output background tasks info

output(queueBackground, logsBackround, "Background tasks manager");

// Output interactive tasks info

output(queueInteractive, logsInteractive, "Interactive tasks manager");

printf("\nCurrent tact: %4d\nTasks will come: %2d", tact, settings.tasksNumber);

Sleep(settings.tactTimeInMS);

tact++;

}

outfile(logsBackround, "Background tasks");

outfile(logsInteractive, "Interactive tasks");

return 0;

}

void addTasks(list<Task> & queue, unsigned int tact)

{

if (!settings.tasksNumber) return;

// Decide how many tasks should we add per tact

unsigned int N = rand() % (settings.tasksMaxPerTact + 1);

// If not many task are we allowed to add

N = settings.tasksNumber < N ? settings.tasksNumber : N;

while (N)

{

queue.push\_back(Task(tact, rand() % settings.taskMaxTime + 1));

N--;

settings.tasksNumber--;

}

}

***Клас ‘процес’ (задача)***

class Task

{

public:

Task(int, int);

bool isPerformed();

unsigned int getID();

unsigned int getTime(int ); // simple getter

void execute(int, int); // execute task for exact time

void delay(); // delay, while task is not working

friend bool operator==(const Task & left, const Task & right);

private:

unsigned int id;

static unsigned int idCommon;

/\* time1 - joined, time2 - time to work on, time3 - done, time4 - delay \*/

unsigned int time1, time2, time3 = 0, time4 = 0, timeToDo = 0;

};

***Опис методів класу ‘процес’(задача)***

#include "Task.h"

unsigned int Task::idCommon = 0;

Task::Task(int \_time1, int \_time2) : time1(\_time1), time2(\_time2), id(idCommon++), timeToDo(\_time2) {}

unsigned int Task::getID()

{

return id;

}

void Task::execute(int t, int tact)

{

timeToDo -= t;

// if task is Done

if (timeToDo <= 0) time3 = t + tact;

}

bool Task::isPerformed()

{

return timeToDo <= 0;

}

unsigned int Task::getTime(int i) // Simple getter

{

switch (i)

{

case 1:

return time1; // Joined

case 2:

return time2; // Task time

case 3:

return time3; // Done

case 4:

return timeToDo; // Left

case 5:

return time4; // Delay

default:

break;

}

}

bool operator== (const Task & task1, const Task & task2)

{

return task1.id == task2.id;

}

void Task::delay()

{

time4++;

}

**Аналіз результатів**

Оскільки даний планувальник працює по-різному залежно від типу задач: якщо задачі інтерактивні, то він мусить приділити кванти свого часу всім процесам; якщо ж задачі фонові, то необхідно виконувати лише найстаріші, тому будемо аналізувати кожну чергу окремо, але не забуваючи про характеристики іншої.

* Щоб переконатися, що планувальник виконує задані алгоритми (LIFO, RR) проведемо тестування і зафіксуємо основні характеристики кожного процесу (час надходження, час виконання, час завершення, час затримки)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Фонові задачі | | | | |
| ID | Час приєднання | Складність виконання | Час за скільки виконав | Затримка |
| 0 | 0 | 2 | 5 | 3 |
| 2 | 4 | 6 | 35 | 25 |
| 4 | 5 | 2 | 42 | 35 |
| 6 | 6 | 5 | 47 | 36 |
| 12 | 11 | 5 | 52 | 36 |
| 14 | 12 | 4 | 56 | 40 |
| Інтерактивні задачі | | | | |
| ID | Час приєднання | Складність виконання | Час за скільки виконав | Затримка |
| 1 | 1 | 3 | 4 | 0 |
| 5 | 5 | 1 | 6 | 0 |
| 13 | 11 | 1 | 14 | 2 |
| 10 | 9 | 2 | 21 | 10 |
| 7 | 6 | 3 | 23 | 14 |
| 11 | 10 | 3 | 31 | 18 |

\*Результати подані в одиниці такту.

Фоновий режим.

Із результатів помітно – чим новіші задачі, тим довші у них затримка і час виконання, адже спочатку виконується (і-1) задача, а вже потім – і задача.

Інтерактивний режим.

Тут складніша ситуація, адже на час виконання процесу впливають 2 фактори:

* складність виконання;
* кількість інших процесів, що знаходяться в черзі, тобто чим менший час приєднання, тим менше інших процесів у черзі, отже менший час виконання.

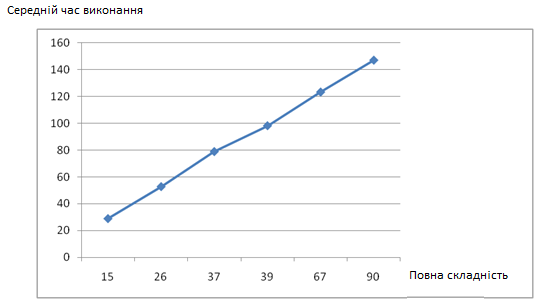
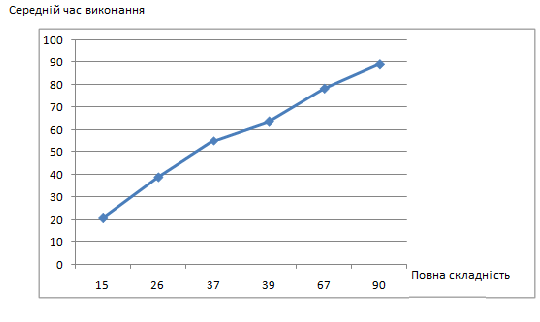
Висновок:

Оскільки результати з таблиці відповідають особливостям роботи двох режимів, то планувальник працює правильно.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Фонові процеси | | | | |
| Тест № | К-сть процесів | Середня складність | Середній час виконання | Середній час затримки | Повна складність |
| 1 | 4 | 3.75 | 29 | 21.5 | 15 |
| 2 | 7 | 3.71429 | 52.8571 | 41.5714 | 26 |
| 3 | 12 | 3.08333 | 79 | 61.4167 | 37 |
| 4 | 16 | 3.0625 | 98.3125 | 75.25 | 49 |
| 5 | 21 | 3.19048 | 123.429 | 94.2381 | 67 |
| 6 | 27 | 3.33333 | 147.185 | 111.593 | 90 |
|  | Інтерактивні процеси | | | | |
| Тест № | Всього процесів | Середня складність | Середній час виконання | Середній час затримки | Повна складність |
| 1 | 6 | 4.33333 | 20.6667 | 11.1667 | 26 |
| 2 | 13 | 3.84615 | 38.7692 | 26.0769 | 50 |
| 3 | 18 | 3.88889 | 54.9444 | 38.7222 | 70 |
| 4 | 24 | 3.54167 | 63.625 | 41.9167 | 85 |
| 5 | 29 | 3.62069 | 78.3448 | 51.8621 | 105 |
| 6 | 33 | 3.63636 | 89.0606 | 58.9697 | 120 |

* Проведемо тестування роботи планувальника в різних умовах: при різній кількості процесів, імовірності появи процесу в кожний такт, складності нових процесів.

\*Результати подані в одиниці такту.

 **Фоновий** **режим** **Інтерактивний** **режим**

Висновок

Отже, результати цілком прогнозовані – із збільшенням кількості задач збільшується час очікування (затримки) задач, відповідно збільшується час роботи системи.

На графіку можна помітити лінійний характер збільшення часу роботи процесора, що пояснюється постійною зайнятістю процесора та його рівномірним навантаженням упродовж усього циклу роботи.

Крім того, варто відзначити відмінність середньої затримки процесів у фоновому та інтерактивному режимах. У фоновому – значно більша затримка, що пояснюється його меншим пріоритетом при розподілі ресурсів системи, адже він отримує лише 20% робочого часу процесора, в той же час інтерактивний отримує 80%.