Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет прикладної математики

Кафедра спеціалізованих систем та системного програмування

**Комп'ютерна криптографія**

Лаборатонрна робота №1

Виконав Жук А.М. :

Група КВ-31м

Перевірив:  
Тесленко О.К

Київ 2013

Завдання

## **5. 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 [SJF](file:///E:\Project\ОС\4\sjf), а друга – за алгоритмім [RR](file:///E:\Project\ОС\4\lecture.html#rr#rr) (дивись пояснення до завдання 2 і 4).

Текст програми

#include <iostream>

#include <string>

#include <fstream>

#include <list>

#include <iomanip>

#include <windows.h>

using namespace std;

int const quantum = 2;

int time = 0;

int q1\_time = 0;

int q2\_time = 0;

struct process {

string id;

int starttime;

int runtime;

int timeleft;

};

struct executed\_process {

string id;

int starttime;

int runtime;

int endtime;

int delaytime;

};

list<process> queue1, queue2;

list<process> q1, q2;

list<executed\_process> executed;

void load\_from\_file(list<process> &q, string path);

void task\_manager();

void show\_results();

void change\_time();

void fon();

void init();

int main() {

load\_from\_file(queue1, "tasks1.txt");

load\_from\_file(queue2, "tasks2.txt");

task\_manager();

show\_results();

return 0;

}

void load\_from\_file(list<process> &q, string path) {

fstream in(path.c\_str());

process tmp;

string s;

char strt[100];

in >> s;

in >> s;

in >> s;

cout << path << endl;

cout << " id starttime runtime" << endl;

while (!in.eof()) {

q.push\_back(tmp);

in >> tmp.id;

in >> tmp.starttime;

in >> tmp.runtime;

cout << setw(4) << tmp.id << " ";

cout << setw(9) << tmp.starttime << " ";

cout << setw(7) << tmp.runtime << " ";

cout << endl;

tmp.timeleft = tmp.runtime;

}

cout << endl;

in.close();

}

void fon() {

list<process>::iterator it, min;

it = min = q2.begin();

while (it != q2.end()) {

if (it->runtime < min->runtime) {

min = it;

}

it++;

}

process p = \*min;

executed\_process current;

current.id = p.id;

current.starttime = p.starttime;

current.runtime = p.runtime;

current.delaytime = time - p.starttime;

current.endtime = time + p.runtime;

while (current.endtime != time) {

cout << setw(4) << time << " ";

cout << setw(6) << current.id << endl;

change\_time();

q2\_time++;

}

executed.push\_back(current);

q2.erase(min);

}

void task\_manager() {

init();

list<executed\_process> current\_list;

executed\_process tmp\_executed;

list<process>::iterator c\_it = q1.begin(), tmp;

cout << "time" << " task" << endl;

int i;

while (1) {

if (q1.empty() || q1\_time > time \* 0.8) {

if (q2.empty()) {

if (time == 50) {

break;

} else {

cout << setw(4) << time << " ";

cout << setw(6) << "free" << endl;

change\_time();

if (!q1.empty()) {

c\_it = q1.begin();

}

continue;

}

} else {

fon();

continue;

}

}

i = 0;

while (i < quantum) {

cout << setw(4) << time << " ";

cout << setw(6) << c\_it->id << endl;

if (c\_it->timeleft <= 1) {

tmp\_executed.id = c\_it->id;

tmp\_executed.starttime = c\_it->starttime;

tmp\_executed.runtime = c\_it->runtime;

tmp\_executed.delaytime = time - c\_it->starttime - c\_it->runtime + 1;

tmp\_executed.endtime = time;

executed.push\_back(tmp\_executed);

tmp = c\_it;

c\_it++;

q1.erase(tmp);

break;

}

c\_it->timeleft--;

change\_time();

q1\_time++;

i++;

}

if (i == quantum) {

c\_it++;

}

if (c\_it == q1.end()) {

c\_it = q1.begin();

}

}

}

void show\_results() {

cout << endl;

if (executed.empty()) {

cout << "There are no executed processes" << endl;

return;

}

cout << " id starttime runtime endtime delaytime" << endl;

double delay = 0;

int size = executed.size();

while (!executed.empty()) {

cout << setw(4) << executed.front().id << " ";

cout << setw(9) << executed.front().starttime << " ";

cout << setw(7) << executed.front().runtime << " ";

cout << setw(7) << executed.front().endtime << " ";

cout << setw(9) << executed.front().delaytime << " ";

cout << endl;

delay += executed.front().delaytime;

executed.pop\_front();

}

delay /= size;

cout << "delay time = " << delay << endl;

}

void init() {

list<process>::iterator it;

for (it = queue1.begin(); it != queue1.end(); it++) {

if (it->starttime == 0) {

q1.push\_back(\*it);

}

}

for (it = queue2.begin(); it != queue2.end(); it++) {

if (it->starttime == 0) {

q2.push\_back(\*it);

}

}

}

void change\_time() {

//Sleep(100);

time++;

list<process>::iterator it;

for (it = queue1.begin(); it != queue1.end(); it++) {

if (it->starttime == time) {

q1.push\_back(\*it);

}

}

for (it = queue2.begin(); it != queue2.end(); it++) {

if (it->starttime == time) {

q2.push\_back(\*it);

}

}

}

Результати:

tasks1.txt

id starttime runtime

1 0 5

2 0 12

3 3 1

4 3 6

5 10 3

tasks2.txt

id starttime runtime

6 0 2

7 0 3

8 5 2

9 10 3

time task

0 1

1 1

2 6

3 6

4 2

5 2

6 3

6 4

7 4

8 1

9 1

10 2

11 2

12 8

13 8

14 4

15 4

16 1

16 2

17 2

18 4

19 4

19 2

20 2

21 7

22 7

23 7

24 2

25 2

26 2

27 2

27 free

28 free

29 free

30 free

31 free

32 free

33 free

34 free

35 free

36 free

37 free

38 free

39 free

40 free

41 free

42 free

43 free

44 free

45 free

46 free

47 free

48 free

49 free

id starttime runtime endtime delaytime

6 0 2 4 2

3 3 1 6 3

8 5 2 14 7

1 0 5 16 12

4 3 6 19 11

7 0 3 24 21

2 0 12 27 16

delay time = 10.2857