**Планировщик EDF**

**Руководство системного программиста**

Санкт - Петербург

2017

**Содержание**

1. Назначение и условия применения программы ……………………… .. 3
2. Структура программы …………………………………………………… .3
3. Установка и настройка программы ………………………………………4
4. Тестирование ………………………………………………………………5

\

1. **Назначение и условия применения программы**

Данная программа предназначена для ускорения работы ОС для некоторых типов задач за счёт добавления нового вида планировщика.

1. **Структура программы**

Структурно программа представлена в виде патча, добавляющего к ядру ОС Linux версии 2.6.24 новую политику планирования. Данный патч изменяет следующие файлы исходного ядра:

* sched.h
* sched.c
* Makefile
* .config
* proc\_misc.h
* proc\_misc.c
* sched\_casio.h
* sched\_casio.c

**3.Установка и настройка программы**

Перед настройкой ядра необходимо загрузить следующее программное обеспечение для сборки ядра:

* linux-source-2.6.24
* kernel-package
* libncurses5-dev
* build-essential

1. С помощью утилиты wget загружаем исходный код ядра:

sudo wget https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.24.tar.bz2

1. Распаковываем архив:

tar –xjvf linux-2.6.24.tar.bz2

1. Создаем конфигурационный файл из текущего системного конфигурационного файла:

sudo cp /boot/config-2.6.24 .config

1. Меняем параметр extraversion в Makefile, чтобы отличать собираемое ядро от других версий

EXTRAVERSION= -edf

1. Компилируем ядро:

sudo make oldconfig

sudo make-kpkg –initrd kernel\_image 2>../errors

1. Начинается компиляция ядра, и если все идет хорошо, создается сжатый образ ядра, в противном случае можно проверить ошибки в файле ошибки, созданном в домашнем каталоге
2. Установим скомпилированную версию ядра, сгенерированную предыдущим шагом

sudo dpkg -i kernel\_image-2.6.24\_xxxx.deb

1. Для завершения процесса необходимо перезагрузить систему и выбрать загрузку нового ядра:



Рис.1. Загрузка с новой версии ядра

**4. Тестирование**

Для тестирования был написан следующий код, устанавливающий процессу политику планирования SCHED\_EDF:

#include <sched.h>

#include <stdio.h>

void printPolicy(){

int policy = sched\_getscheduler(0);

switch (policy) {

case SCHED\_OTHER:

printf ("Обычная политика\n");

break;

case SCHED\_RR:

printf ("Политика карусели\n");

break;

case SCHED\_FIFO:

printf("Политика FIFO\n");

break;

case SCHED\_DEADLINE:

printf("Политика Deadline\n");

break;

case -1:

printf("sched\_getscheduler");

break;

default:

printf("Неизвестная политика!\n");

}

}

int main(){

printPolicy();

struct sched\_param sp = { .sched\_priority = 1 };

int ret;

ret = sched\_setscheduler(0, SCHED\_RR, &sp);

printPolicy();

if (sched\_yield() == 0)

printf("sched\_yield\n");

return 0;

}

Также данный код был апробирован в версиях ядра 3.16 и 4.11, где уже реализована политика планирования EDF под названием SCHED\_DEADLINE.

В завершение был использован набор тестов[5] для проверки политики планирования. Результаты меняются в зависимости от задаваемых приоритетов и временных отрезков:

[1455]

missed deadlines = 2856

missed periods = 821

Total adjustments = 48064

us deadline : 1000

us runtime : 400 us

nr\_periods : 6368

[1460]

missed deadlines = 3802

missed periods = 1026

Total adjustments = 20386 us

deadline : 1000 us

runtime : 400 us

nr\_periods : 5188