

**PODER EXECUTIVO**

**MINISTÉRIO DA EDUCAÇÃO**

**UNIVERSIDADE FEDERAL DE RORAIMA**

**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**TUTORIAL DO PROJETO: IMPLEMENTAÇÃO DA POLÍTICA SCHEDULE BACKGROUND**

**ALUNOS:**

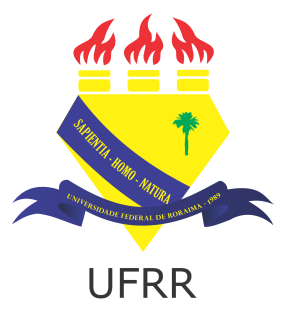
**Victor Hugo Souza Costa**

**Giovana Oliveira Moraes de Lima**

**Ryan Kayky Marques Rolins Bastos**

**Setembro de 2024**

**Boa Vista/Roraima**



**PODER EXECUTIVO**

**MINISTÉRIO DA EDUCAÇÃO**

**UNIVERSIDADE FEDERAL DE RORAIMA**

**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**SISTEMAS OPERACIONAIS I**

**TUTORIAL DO PROJETO: IMPLEMENTAÇÃO DA POLÍTICA SCHEDULE BACKGROUND**

**Setembro de 2024**

**Boa Vista/Roraima**

**Resumo**

Este projeto tem como objetivo modificar o escalonador de processos do Linux para implementar uma nova classe de escalonador chamada SCHED\_BACKGROUND. Esta nova política é projetada para suportar processos que só precisam ser executados quando o sistema não possui outras tarefas ativas. O escalonador em background deve garantir que processos com essa política sejam executados apenas na ausência de processos nas classes SCHED\_OTHER, SCHED\_RR, ou SCHED\_FIFO. Quando múltiplos processos SCHED\_BACKGROUND estão prontos para execução, eles competem pela CPU de forma semelhante aos processos da classe SCHED\_OTHER.

**PRIMEIROS PASSOS:**

1. **Verificar a versão do kernel**

uname -r

1. **Baixar o código fonte (estou usando a 5.15.0)**

wget https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/snapshot/linux-5.15.tar.gz

1. **Extraia o código fonte**

tar -xzvf linux-5.15.tar.gz

1. **Instale as dependências necessárias**

sudo apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev

1. **Modificar arquivos relevantes:**

* **/INCLUDE/UAPI/LINUX/SCHED.H**: Defina a nova política

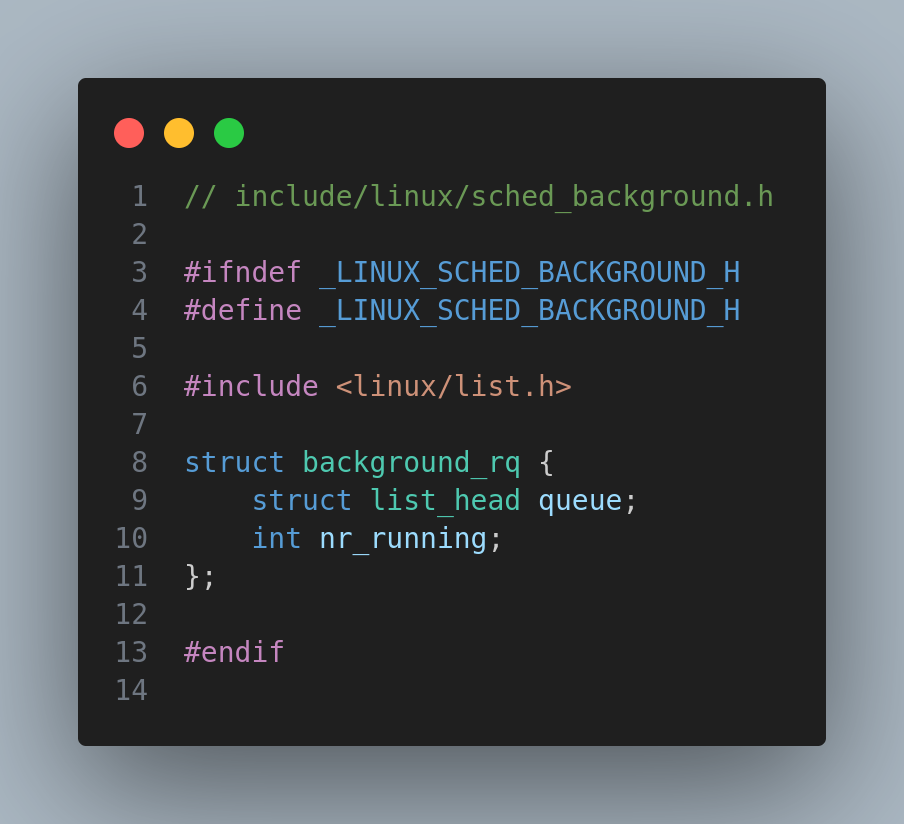
#define SCHED\_BACKGROUND 7

* **/KERNEL/SCHED:** criar dois novos arquivos, *sched\_background.c e sched\_background.h* (especificados nas imagens abaixo)

**/KERNEL/SCHED/SCHED\_BACKGROUND.C:**



**/KERNEL/SCHED/SCHED\_BACKGROUND.H:**

****

**/KERNEL/SCHED/SCHED.H:** adicionar o include #include "sched\_background.h" e adicionar novas funções.

1. **Criar uma função *background\_policy* (preferencialmente próximo da função *valid policy*):**

static inline int background\_policy(int policy){

return policy == SCHED\_BACKGROUND;}

1. **Adicionar na função *valid\_policy* a nova política:**

static inline bool valid\_policy(int policy){

return idle\_policy(policy) || fair\_policy(policy) ||

rt\_policy(policy) || dl\_policy(policy) || background\_policy(policy);}

1. **Criar uma função *task\_has\_background\_policy*:**

static inline int task\_has\_background\_policy(struct task\_struct \*p){

return (p->policy == SCHED\_BACKGROUND);}

1. **Na estrutura *struct rq* adicionar junto das outras estruturas a background:**

struct cfs\_rq cfs;

struct rt\_rq rt;

struct dl\_rq dl;

struct background\_rq background;

1. **Junto das outras classes *sched\_class* adicionar a background:**

…

extern const struct sched\_class idle\_sched\_class;

extern const struct sched\_class background\_sched\_class;

1. **Criar *task\_struct* da background:**

…

extern struct task\_struct \*pick\_next\_task\_idle(struct rq \*rq);

extern struct task\_struct \*pick\_next\_task\_background(struct rq \*rq);

**/KERNEL/SCHED/CORE.C:** Modificar algumas funções.

1. **Modificar a função *task\_struct \* \_\_pick\_next\_task***

static inline struct task\_struct \*

\_\_pick\_next\_task(struct rq \*rq, struct task\_struct \*prev, struct rq\_flags \*rf){

const struct sched\_class \*class;

struct task\_struct \*p;

if (likely(prev->sched\_class <= &fair\_sched\_class &&

rq->nr\_running == rq->cfs.h\_nr\_running)) {

p = pick\_next\_task\_fair(rq, prev, rf);

if (unlikely(p == RETRY\_TASK))

goto restart;

if (!p) {

put\_prev\_task(rq, prev);

p = pick\_next\_task\_idle(rq);}

return p;}

restart:

put\_prev\_task\_balance(rq, prev, rf);

for\_each\_class(class) {

p = class->pick\_next\_task(rq);

if (p)

return p;}

/\* Novo caso para SCHED\_BACKGROUND \*/

if (likely(rq->background.nr\_running > 0)) {

p = pick\_next\_task\_background(rq);

if (p)

return p;}

BUG();}

1. **Modificar a função *\_\_sched\_setscheduler.***

static int \_\_sched\_setscheduler(struct task\_struct \*p,

const struct sched\_attr \*attr,

bool user, bool pi){

int oldpolicy = -1, policy = attr->sched\_policy;

int retval, oldprio, newprio, queued, running;

const struct sched\_class \*prev\_class;

struct callback\_head \*head;

struct rq\_flags rf;

int reset\_on\_fork;

int queue\_flags = DEQUEUE\_SAVE | DEQUEUE\_MOVE | DEQUEUE\_NOCLOCK;

struct rq \*rq;

/\* The pi code expects interrupts enabled \*/

BUG\_ON(pi && in\_interrupt());

recheck:

/\* Double check policy once rq lock held: \*/

if (policy < 0) {

reset\_on\_fork = p->sched\_reset\_on\_fork;

policy = oldpolicy = p->policy; }

else {

reset\_on\_fork = !!(attr->sched\_flags & SCHED\_FLAG\_RESET\_ON\_FORK);

if (!valid\_policy(policy))

return -EINVAL;}

if (attr->sched\_flags & ~(SCHED\_FLAG\_ALL | SCHED\_FLAG\_SUGOV))

return -EINVAL;

/\*

\* Valid priorities for SCHED\_FIFO and SCHED\_RR are

\* 1..MAX\_RT\_PRIO-1, valid priority for SCHED\_NORMAL,

\* SCHED\_BATCH, SCHED\_IDLE, and SCHED\_BACKGROUND is 0.

\*/

if (attr->sched\_priority > MAX\_RT\_PRIO-1)

return -EINVAL;

if ((dl\_policy(policy) && !\_\_checkparam\_dl(attr)) ||

(rt\_policy(policy) != (attr->sched\_priority != 0)))

return -EINVAL;

/\*

\* Allow unprivileged RT tasks to decrease priority:

\*/

if (user && !capable(CAP\_SYS\_NICE)) {

if (fair\_policy(policy)) {

if (attr->sched\_nice < task\_nice(p) &&

!can\_nice(p, attr->sched\_nice))

return -EPERM;}

if (rt\_policy(policy)) {

unsigned long rlim\_rtprio =

task\_rlimit(p, RLIMIT\_RTPRIO);

/\* Can't set/change the rt policy: \*/

if (policy != p->policy && !rlim\_rtprio)

return -EPERM;

/\* Can't increase priority: \*/

if (attr->sched\_priority > p->rt\_priority && attr->sched\_priority > rlim\_rtprio)

return -EPERM;}

/\*

\* Can't set/change SCHED\_DEADLINE policy at all for now

\* (safest behavior); in the future we would like to allow

\* unprivileged DL tasks to increase their relative deadline

\* or reduce their runtime (both ways reducing utilization)

\*/

if (dl\_policy(policy))

return -EPERM;

/\*

\* Treat SCHED\_IDLE as nice 20. Only allow a switch to

\* SCHED\_NORMAL if the RLIMIT\_NICE would normally permit it.

\*/

if (task\_has\_idle\_policy(p) && !idle\_policy(policy)) {

if (!can\_nice(p, task\_nice(p)))

return -EPERM;}

/\*

Trate SCHED\_BACKGROUND de forma semelhante a SCHED\_IDLE:

\*/

if (task\_has\_background\_policy(p) && !background\_policy(policy)) {

if (!can\_nice(p, task\_nice(p)))

return -EPERM;}

/\* Can't change other user's priorities: \*/

if (!check\_same\_owner(p))

return -EPERM;

/\* Normal users shall not reset the sched\_reset\_on\_fork flag: \*/

if (p->sched\_reset\_on\_fork && !reset\_on\_fork)

return -EPERM;}

if (user){

if (attr->sched\_flags & SCHED\_FLAG\_SUGOV)

return -EINVAL;

retval = security\_task\_setscheduler(p);

if (retval)

return retval;}

/\* Update task specific "requested" clamps \*/

if (attr->sched\_flags & SCHED\_FLAG\_UTIL\_CLAMP) {

retval = uclamp\_validate(p, attr);

if (retval)

return retval;}

if (pi)

cpuset\_read\_lock();

/\*

\* Make sure no PI-waiters arrive (or leave) while we are

\* changing the priority of the task:

\*

\* To be able to change p->policy safely, the appropriate

\* runqueue lock must be held.

\*/

rq = task\_rq\_lock(p, &rf);

update\_rq\_clock(rq);

/\*

\* Changing the policy of the stop threads is a very bad idea:

\*/

if (p == rq->stop) {

retval = -EINVAL;

goto unlock;}

/\*

\* If not changing anything there's no need to proceed further,

\* but store a possible modification of reset\_on\_fork.

\*/

if (unlikely(policy == p->policy)) {

if (fair\_policy(policy) && attr->sched\_nice != task\_nice(p))

goto change;

if (rt\_policy(policy) && attr->sched\_priority != p->rt\_priority)

goto change;

if (dl\_policy(policy) && dl\_param\_changed(p, attr))

goto change;

if (attr->sched\_flags & SCHED\_FLAG\_UTIL\_CLAMP)

goto change;

p->sched\_reset\_on\_fork = reset\_on\_fork;

retval = 0;

goto unlock;}

change:

if (user) {

#ifdef CONFIG\_RT\_GROUP\_SCHED

/\*

\* Do not allow realtime tasks into groups that have no runtime

\* assigned.

\*/

if (rt\_bandwidth\_enabled() && rt\_policy(policy) &&

task\_group(p)->rt\_bandwidth.rt\_runtime == 0 &&

!task\_group\_is\_autogroup(task\_group(p))) {

retval = -EPERM;

goto unlock;}

#endif

#ifdef CONFIG\_SMP

if (dl\_bandwidth\_enabled() && dl\_policy(policy) &&

!(attr->sched\_flags & SCHED\_FLAG\_SUGOV)) {

cpumask\_t \*span = rq->rd->span;

/\*

\* Don't allow tasks with an affinity mask smaller than

\* the entire root\_domain to become SCHED\_DEADLINE. We

\* will also fail if there's no bandwidth available.

\*/

if (!cpumask\_subset(span, p->cpus\_ptr) ||

rq->rd->dl\_bw.bw == 0) {

retval = -EPERM;

goto unlock;

}

}

#endif

}

/\* Re-check policy now with rq lock held: \*/

if (unlikely(oldpolicy != -1 && oldpolicy != p->policy)) {

policy = oldpolicy = -1;

task\_rq\_unlock(rq, p, &rf);

if (pi)

cpuset\_read\_unlock();

goto recheck;}

/\*

\* If setscheduling to SCHED\_DEADLINE (or changing the parameters

\* of a SCHED\_DEADLINE task) we need to check if enough bandwidth

\* is available.

\*/

if ((dl\_policy(policy) || dl\_task(p)) && sched\_dl\_overflow(p, policy, attr)) {

retval = -EBUSY;

goto unlock;}

p->sched\_reset\_on\_fork = reset\_on\_fork;

oldprio = p->prio;

newprio = \_\_normal\_prio(policy, attr->sched\_priority, attr->sched\_nice);

if (pi) {

/\*

\* Take priority boosted tasks into account. If the new

\* effective priority is unchanged, we just store the new

\* normal parameters and do not touch the scheduler class and

\* the runqueue. This will be done when the task deboost

\* itself.

\*/

newprio = rt\_effective\_prio(p, newprio);

if (newprio == oldprio)

queue\_flags &= ~DEQUEUE\_MOVE;}

queued = task\_on\_rq\_queued(p);

running = task\_current(rq, p);

if (queued)

dequeue\_task(rq, p, queue\_flags);

if (running)

put\_prev\_task(rq, p);

prev\_class = p->sched\_class;

if (!(attr->sched\_flags & SCHED\_FLAG\_KEEP\_PARAMS)) {

\_\_setscheduler\_params(p, attr);

\_\_setscheduler\_prio(p, newprio);}

\_\_setscheduler\_uclamp(p, attr);

if (queued) {

/\*

\* We enqueue to tail when the priority of a task is

\* increased (user space view).

\*/

if (oldprio < p->prio)

queue\_flags |= ENQUEUE\_HEAD;

enqueue\_task(rq, p, queue\_flags);}

if (running)

set\_next\_task(rq, p);

check\_class\_changed(rq, p, prev\_class, oldprio);

/\* Avoid rq from going away on us: \*/

preempt\_disable();

head = splice\_balance\_callbacks(rq);

task\_rq\_unlock(rq, p, &rf);

if (pi) {

cpuset\_read\_unlock();

rt\_mutex\_adjust\_pi(p);}

/\* Run balance callbacks after we've adjusted the PI chain: \*/

balance\_callbacks(rq, head);

preempt\_enable();

return 0;

unlock:

task\_rq\_unlock(rq, p, &rf);

if (pi)

cpuset\_read\_unlock();

return retval;}

1. **Modificar também SYSCALL\_DEFINE1(sched\_get\_priority\_max, int, policy) e SYSCALL\_DEFINE1(sched\_get\_priority\_min, int, policy)**

SYSCALL\_DEFINE1(sched\_get\_priority\_max, int, policy){

int ret = -EINVAL;

switch (policy) {

case SCHED\_FIFO:

case SCHED\_RR:

ret = MAX\_RT\_PRIO-1;

break;

case SCHED\_DEADLINE:

case SCHED\_NORMAL:

case SCHED\_BATCH:

case SCHED\_IDLE:

case SCHED\_BACKGROUND:

ret = 0;

break;

}

return ret;

}

/\*\*

\* sys\_sched\_get\_priority\_min - return minimum RT priority.

\* @policy: scheduling class.

\*

\* Return: On success, this syscall returns the minimum

\* rt\_priority that can be used by a given scheduling class.

\* On failure, a negative error code is returned.

\*/

SYSCALL\_DEFINE1(sched\_get\_priority\_min, int, policy)

{

int ret = -EINVAL;

switch (policy) {

case SCHED\_FIFO:

case SCHED\_RR:

ret = 1;

break;

case SCHED\_DEADLINE:

case SCHED\_NORMAL:

case SCHED\_BATCH:

case SCHED\_IDLE:

case SCHED\_BACKGROUND:

ret = 0;

}

return ret;

}

1. **Configure o kernel:**

Existem duas opções de configuração do kernel:

1. Utilizando o comando `cp /boot/config-$(uname -r) .config`, que copia as configurações do kernel atual. No entanto, se o kernel não for o mesmo ou se houver a necessidade de um kernel específico, será necessário realizar modificações.

2. Usando o comando `make defconfig`, que carrega configurações padrão, onde diversos drivers e outros componentes estão desativados.

Para personalizar as configurações, pode-se utilizar o comando `make menuconfig`. Caso essa ferramenta não esteja disponível, pode-se prosseguir sem ela.

1. **Compile o kernel**

make -j$(nproc)

1. **Instale o kernel**

sudo make modules\_install

sudo make install

1. **Atualize o grub e reinicie**

sudo update-grub

sudo reboot

1. **Verifique a nova versão do kernel**

uname -r

1. **Teste a nova política**

Para fins de teste, foram criados dois arquivos .c: *teste\_background.c* e *teste\_performance.c*.

teste\_background.c: Destinado a verificar se a nova política foi criada com sucesso.

#include <stdio.h>

#include <sched.h>

#include <unistd.h>

#define SCHED\_BACKGROUND 7

int main() {

    struct sched\_param param;

    param.sched\_priority = 0;

    if (sched\_setscheduler(0, SCHED\_BACKGROUND, &param) == -1) {

        perror("sched\_setscheduler");

        return 1;}

    printf("SCHED\_BACKGROUND aplicada com sucesso!\n");

    return 0;}

**TESTE\_PERFORMANCE.C:** Para saber a performance da nova política nova

#include <sys/time.h>

#include <sys/resource.h>

#include <stdio.h>

int main() {

struct timeval start, end;

struct rusage usage;

gettimeofday(&start, NULL);

for (long i = 0; i < 1000000000; i++);

gettimeofday(&end, NULL);

getrusage(RUSAGE\_SELF, &usage);

long seconds = end.tv\_sec - start.tv\_sec;

long micros = ((seconds \* 1000000) + end.tv\_usec) - start.tv\_usec;

printf("Tempo de wallclock: %ld microssegundos\n", micros);

printf("Tempo de usuário: %ld microssegundos\n", usage.ru\_utime.tv\_sec \* 1000000 + usage.ru\_utime.tv\_usec);

printf("Tempo de sistema: %ld microssegundos\n", usage.ru\_stime.tv\_sec \* 1000000 + usage.ru\_stime.tv\_usec);

return 0;

}

**Após a criação basta usar os comandos:**

gcc -o teste\_background teste\_background.c

sudo ./teste\_background

gcc -o teste\_performance teste\_performance.c

sudo ./teste\_performance