562855-2-45P AID: 258164| 20/10/2020

**The following show the program planning of the scheduling problem:**

1. Create c language file “schedproc.h” and “schedule.c”.
2. The “schedule.c” file contain the scheduling policy for SCHED.
3. The entry points are: a)do\_noquantum: Called on behalf of process' that run out of quantum, b)do\_start\_scheduling: Request to start scheduling a proc ,c) do\_stop\_scheduling Request to stop scheduling a proc, d) do\_nice: Request to change the nice level on a proc, e) init\_scheduling : Called from main.c to set up/prepare scheduling.
4. The “schedproc.h” file contain scheduling information for each process.
5. Compile the c program , using c complier.

**Following show the code of the depreciation:**

schedproc.h

#ifdef \_MAIN

#undef EXTERN

#define EXTERN

#endif

#ifndef CONFIG\_SMP

#define CONFIG\_MAX\_CPUS 1

#endif

EXTERN struct schedproc {

endpoint\_t endpoint; /\* process endpoint id \*/

endpoint\_t parent; /\* parent endpoint id \*/

unsigned flags; /\* flag bits \*/

unsigned max\_priority;

/\* this process' highest allowed priority \*/

unsigned priority;

/\* the process' current priority \*/

unsigned time\_slice;

/\* this process's time slice \*/

unsigned cpu;

/\* what CPU is the process running on \*/

bitchunk\_t cpu\_mask[BITMAP\_CHUNKS(CONFIG\_MAX\_CPUS)];

int timesCalled;

} schedproc[NR\_PROCS];

#define IN\_USE 0x00001

schedule.c

#include "sched.h"

#include "schedproc.h"

#include <assert.h>

#include <minix/com.h>

#include <machine/archtypes.h>

#include "kernel/proc.h" /\* for queue constants \*/

PRIVATE timer\_t sched\_timer;

PRIVATE unsigned balance\_timeout;

#define BALANCE\_TIMEOUT 5 /\* how often to balance queues in seconds \*/

FORWARD \_PROTOTYPE( int schedule\_process, (struct schedproc \* rmp,

unsigned flags));

FORWARD \_PROTOTYPE( void balance\_queues, (struct timer \*tp) );

#define SCHEDULE\_CHANGE\_PRIO 0x1

#define SCHEDULE\_CHANGE\_QUANTUM 0x2

#define SCHEDULE\_CHANGE\_CPU 0x4

#define SCHEDULE\_CHANGE\_ALL (

SCHEDULE\_CHANGE\_PRIO

SCHEDULE\_CHANGE\_QUANTUM

SCHEDULE\_CHANGE\_CPU

)

#define schedule\_process\_local(p)

schedule\_process(p,SCHEDULE\_CHANGE\_PRIO SCHEDULE\_CHANGE\_QUANTUM)

#define schedule\_process\_migrate(p)

schedule\_process(p, SCHEDULE\_CHANGE\_CPU)

#define CPU\_DEAD -1

#define cpu\_is\_available(c) (cpu\_proc[c] >= 0)

#define DEFAULT\_USER\_TIME\_SLICE 200

/\* processes created by RS are sysytem processes \*/

#define is\_system\_proc(p) ((p)->parent == RS\_PROC\_NR)

PRIVATE unsigned cpu\_proc[CONFIG\_MAX\_CPUS];

PRIVATE void pick\_cpu(struct schedproc \* proc)

{

#ifdef CONFIG\_SMP

unsigned cpu, c;

unsigned cpu\_load = (unsigned) -1;

if (machine.processors\_count == 1) {

proc->cpu = machine.bsp\_id;

return;

}

/\* schedule sysytem processes only on the boot cpu \*/

if (is\_system\_proc(proc)) {

proc->cpu = machine.bsp\_id;

return;

}

/\* if no other cpu available, try BSP \*/

cpu = machine.bsp\_id;

for (c = 0; c < machine.processors\_count; c++) {

/\* skip dead cpus \*/

if (!cpu\_is\_available(c))

continue;

if (c != machine.bsp\_id && cpu\_load > cpu\_proc[c]) {

cpu\_load = cpu\_proc[c];

cpu = c;

}

}

proc->cpu = cpu;

cpu\_proc[cpu]++;

#else

proc->cpu = 0;

#endif

}

do\_noquantum

PUBLIC int do\_noquantum(message \*m\_ptr)

{

register struct schedproc \*rmp;

int rv, proc\_nr\_n;

if (sched\_isokendpt(m\_ptr->m\_source, &proc\_nr\_n) != OK) {

printf("SCHED: WARNING: got an invalid endpoint in OOQ msg %u.\n",

m\_ptr->m\_source);

return EBADEPT;

}

rmp = &schedproc[proc\_nr\_n];

if (rmp->priority < MIN\_USER\_Q) {

rmp->priority += 1; /\* lower priority \*/

}

if ((rv = schedule\_process\_local(rmp)) != OK) {

return rv;

}

return OK;

}

PUBLIC int do\_stop\_scheduling(message \*m\_ptr)

{

register struct schedproc \*rmp;

int proc\_nr\_n;

/\* check who can send you requests \*/

if (!accept\_message(m\_ptr))

return EPERM;

if (sched\_isokendpt(m\_ptr->SCHEDULING\_ENDPOINT, &proc\_nr\_n) != OK) {

printf("SCHED: WARNING: got an invalid endpoint in OOQ msg "

"%ld\n", m\_ptr->SCHEDULING\_ENDPOINT);

return EBADEPT;

}

rmp = &schedproc[proc\_nr\_n];

schedproc->timesCalled+=1;

#ifdef CONFIG\_SMP

cpu\_proc[rmp->cpu]--;

#endif

rmp->flags = 0; /\*&= ~IN\_USE;\*/

return OK;

}

PUBLIC int do\_start\_scheduling(message \*m\_ptr)

{

register struct schedproc \*rmp;

int rv, proc\_nr\_n, parent\_nr\_n;

/\* we can handle two kinds of messages here \*/

assert(m\_ptr->m\_type == SCHEDULING\_START ||

m\_ptr->m\_type == SCHEDULING\_INHERIT);

/\* check who can send you requests \*/

if (!accept\_message(m\_ptr))

return EPERM;

/\* Resolve endpoint to proc slot. \*/

if ((rv = sched\_isemtyendpt(m\_ptr->SCHEDULING\_ENDPOINT, &proc\_nr\_n))

!= OK) {

return rv;

}

rmp = &schedproc[proc\_nr\_n];

/\* Populate process slot \*/

rmp->endpoint = m\_ptr->SCHEDULING\_ENDPOINT;

rmp->parent = m\_ptr->SCHEDULING\_PARENT;

rmp->max\_priority = (unsigned) m\_ptr->SCHEDULING\_MAXPRIO;

if (rmp->max\_priority >= NR\_SCHED\_QUEUES) {

return EINVAL;

}

/\* Inherit current priority and time slice from parent. Since there

\* is currently only one scheduler scheduling the whole system, this

\* value is local and we assert that the parent endpoint is valid \*/

if (rmp->endpoint == rmp->parent) {

/\* We have a special case here for init, which is the first

process scheduled, and the parent of itself. \*/

rmp->priority = USER\_Q;

rmp->time\_slice = DEFAULT\_USER\_TIME\_SLICE;

/\*

\* Since kernel never changes the cpu of a process, all are

\* started on the BSP and the userspace scheduling hasn't

\* changed that yet either, we can be sure that BSP is the

\* processor where the processes run now.

\*/

#ifdef CONFIG\_SMP

rmp->cpu = machine.bsp\_id;

/\* FIXME set the cpu mask \*/

#endif

}

switch (m\_ptr->m\_type) {

case SCHEDULING\_START:

/\* We have a special case here for system processes, for which

\* quanum and priority are set explicitly rather than inherited

\* from the parent \*/

rmp->priority = rmp->max\_priority;

rmp->time\_slice = (unsigned) m\_ptr->SCHEDULING\_QUANTUM;

break;

if ((rv = sched\_isokendpt(m\_ptr->SCHEDULING\_PARENT,

&parent\_nr\_n)) != OK)

return rv;

rmp->priority = schedproc[parent\_nr\_n].priority;

rmp->time\_slice = schedproc[parent\_nr\_n].time\_slice;

break;

default:

/\* not reachable \*/

assert(0);

}

if ((rv = sys\_schedctl(0, rmp->endpoint, 0, 0, 0)) != OK) {

printf("Sched: Error taking over scheduling for %d, kernel said %d\n",

rmp->endpoint, rv);

return rv;

}

rmp->flags = IN\_USE;

/\* Schedule the process, giving it some quantum \*/

pick\_cpu(rmp);

while ((rv = schedule\_process(rmp, SCHEDULE\_CHANGE\_ALL)) == EBADCPU) {

/\* don't try this CPU ever again \*/

cpu\_proc[rmp->cpu] = CPU\_DEAD;

pick\_cpu(rmp);

}

if (rv != OK) {

printf("Sched: Error while scheduling process, kernel replied %d\n",

rv);

return rv;

}

/\* Mark ourselves as the new scheduler.

\* By default, processes are scheduled by the parents scheduler. In case

\* this scheduler would want to delegate scheduling to another

\* scheduler, it could do so and then write the endpoint of that

\* scheduler into SCHEDULING\_SCHEDULER

\*/

m\_ptr->SCHEDULING\_SCHEDULER = SCHED\_PROC\_NR;

return OK;

}

PUBLIC int do\_nice(message \*m\_ptr)

{

struct schedproc \*rmp;

int rv;

int proc\_nr\_n;

unsigned new\_q, old\_q, old\_max\_q;

/\* check who can send you requests \*/

if (!accept\_message(m\_ptr))

return EPERM;

if (sched\_isokendpt(m\_ptr->SCHEDULING\_ENDPOINT, &proc\_nr\_n) != OK) {

printf("SCHED: WARNING: got an invalid endpoint in OOQ msg "

"%ld\n", m\_ptr->SCHEDULING\_ENDPOINT);

return EBADEPT;

}

rmp = &schedproc[proc\_nr\_n];

new\_q = (unsigned) m\_ptr->SCHEDULING\_MAXPRIO;

if (new\_q >= NR\_SCHED\_QUEUES) {

return EINVAL;

}

/\* Store old values, in case we need to roll back the changes \*/

old\_q = rmp->priority;

old\_max\_q = rmp->max\_priority;

/\* Update the proc entry and reschedule the process \*/

rmp->max\_priority = rmp->priority = new\_q;

if ((rv = schedule\_process\_local(rmp)) != OK) {

/\* Something went wrong when rescheduling the process, roll

\* back the changes to proc struct \*/

rmp->priority = old\_q;

rmp->max\_priority = old\_max\_q;

}

return rv;

}

PRIVATE int schedule\_process(struct schedproc \* rmp, unsigned flags)

{

int err;

int new\_prio, new\_quantum, new\_cpu;

pick\_cpu(rmp);

if (flags & SCHEDULE\_CHANGE\_PRIO)

new\_prio = rmp->priority;

else

new\_prio = -1;

if (flags & SCHEDULE\_CHANGE\_QUANTUM)

new\_quantum = rmp->time\_slice;

else

new\_quantum = -1;

if(flags & SCHEDULE\_CHANGE\_CPU)

new\_cpu = rmp->cpu;

else

new\_cpu = -1;

if ((err = sys\_schedule(rmp->endpoint, new\_prio,

new\_quantum, new\_cpu)) != OK) {

printf("PM: An error occurred when trying to schedule %d: %d\n",

rmp->endpoint, err);

}

return err;

}

PUBLIC void init\_scheduling(void)

{

balance\_timeout = BALANCE\_TIMEOUT \* sys\_hz();

init\_timer(&sched\_timer);

set\_timer(&sched\_timer, balance\_timeout, balance\_queues, 0);

}

PRIVATE void balance\_queues(struct timer \*tp)

{

struct schedproc \*rmp;

struct schedproc \*lesser;

rmp = schedproc;

lesser = rmp;

int proc\_nr;

for (proc\_nr=0, rmp=schedproc; proc\_nr < NR\_PROCS; proc\_nr++, rmp++){

if (rmp->timesCalled < lesser->timesCalled)

{

rmp[proc\_nr]= lesser[proc\_nr];

}

}

lesser->priotity -=1;

for (proc\_nr=0, rmp=schedproc; proc\_nr < NR\_PROCS; proc\_nr++, rmp++) {

if (rmp->flags & IN\_USE) {

if (rmp->priority > rmp->max\_priority) {

rmp->priority -= 1; /\* increase priority \*/

schedule\_process\_local(rmp);

}

}

}

set\_timer(&sched\_timer, balance\_timeout, balance\_queues, 0);

}