Yacine Manseur

12/19/15

Professor Hakner

Operating Systems

PROBLEM SET 9

Output from test.c:

yacine@yacine-N53SV ~/Documents/OS/OS/PS9 $ ./test.exe

Hooray made it to init\_fn, stkaddr 0x7fe2fdd75ff4

Asked to adjust child stack by 0XFFFFFFFFFFFDF000 bytes bet 0x7fe2fdd45000 and 0x7fe2fdd55000

Adjusted saved bp @0x7fe2fdd54fd8 to 0x7fe2fdd54ff8

Adjusted saved bp @0x7fe2fdd54ff8 to 0x7fe2fdd55000

Enough already, saved BP @0x7fe2fdd55000 is (nil)

<<in parent addr 0x7fe2fdd75ff4>>

Wow, made it to parent, stkaddr=0x7fe2fdd75fd4

Asked to adjust child stack by 0XFFFFFFFFFFFCF000 bytes bet 0x7fe2fdd35000 and 0x7fe2fdd45000

Adjusted saved bp @0x7fe2fdd44fb8 to 0x7fe2fdd44fd8

Adjusted saved bp @0x7fe2fdd44fd8 to 0x7fe2fdd44ff8

Adjusted saved bp @0x7fe2fdd44ff8 to 0x7fe2fdd45000

Enough already, saved BP @0x7fe2fdd45000 is (nil)

<<in child addr 0x7fe2fdd54ff4>>

CHILD 1: START &y=0x7fe2fdd54fd4

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 20 1

3 1 READY 0x7fe2fdd35000 0 20 0

CHILD 2: START &y=0x7fe2fdd44fb4

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 RUNNING 0x7fe2fdd45000 0 21 2

3 1 READY 0x7fe2fdd35000 4 24 1

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 21 3

3 1 READY 0x7fe2fdd35000 4 24 1

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 22 4

3 1 READY 0x7fe2fdd35000 4 24 1

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 22 5

3 1 READY 0x7fe2fdd35000 4 24 1

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 23 6

3 1 READY 0x7fe2fdd35000 4 24 1

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 23 7

3 1 READY 0x7fe2fdd35000 4 24 1

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 24 8

3 1 READY 0x7fe2fdd35000 4 24 1

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 24 9

3 1 READY 0x7fe2fdd35000 4 24 1

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 25 10

3 1 RUNNING 0x7fe2fdd35000 4 24 1

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 RUNNING 0x7fe2fdd45000 0 25 10

3 1 READY 0x7fe2fdd35000 4 25 2

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 25 11

3 1 READY 0x7fe2fdd35000 4 25 2

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 26 12

3 1 RUNNING 0x7fe2fdd35000 4 25 2

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 26 12

3 1 READY 0x7fe2fdd35000 4 25 3

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 RUNNING 0x7fe2fdd45000 0 26 12

3 1 READY 0x7fe2fdd35000 4 26 4

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 26 13

3 1 READY 0x7fe2fdd35000 4 26 4

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 27 14

3 1 RUNNING 0x7fe2fdd35000 4 26 4

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 RUNNING 0x7fe2fdd45000 0 27 14

3 1 READY 0x7fe2fdd35000 4 27 5

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 27 15

3 1 READY 0x7fe2fdd35000 4 27 5

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 28 16

3 1 RUNNING 0x7fe2fdd35000 4 27 5

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 28 16

3 1 READY 0x7fe2fdd35000 4 27 6

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 RUNNING 0x7fe2fdd45000 0 28 16

3 1 READY 0x7fe2fdd35000 4 28 7

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 28 17

3 1 READY 0x7fe2fdd35000 4 28 7

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 29 18

3 1 RUNNING 0x7fe2fdd35000 4 28 7

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 RUNNING 0x7fe2fdd45000 0 29 18

3 1 READY 0x7fe2fdd35000 4 29 8

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 29 19

3 1 READY 0x7fe2fdd35000 4 29 8

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 30 20

3 1 RUNNING 0x7fe2fdd35000 4 29 8

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 30 20

3 1 READY 0x7fe2fdd35000 4 29 9

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 RUNNING 0x7fe2fdd45000 0 30 20

3 1 READY 0x7fe2fdd35000 4 30 10

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

2 1 READY 0x7fe2fdd45000 0 30 21

3 1 READY 0x7fe2fdd35000 4 30 10

CHILD 1: DONE y=536870912

Process 2 exited with code 22

Process 1 is now waking up

pid ppid current state base addr nice static total ticks

1 0 RUNNING 0x7fe2fdd66000 0 20 0

2 1 ZOMBIE 0x7fe2fdd45000 0 30 21

3 1 READY 0x7fe2fdd35000 4 30 10

Zombie Process 2 killed by Parent Process 1.

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 RUNNING 0x7fe2fdd35000 4 30 10

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 READY 0x7fe2fdd35000 4 30 11

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 READY 0x7fe2fdd35000 4 31 12

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 READY 0x7fe2fdd35000 4 32 13

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 READY 0x7fe2fdd35000 4 32 14

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 READY 0x7fe2fdd35000 4 33 15

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 READY 0x7fe2fdd35000 4 34 16

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 READY 0x7fe2fdd35000 4 34 17

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 READY 0x7fe2fdd35000 4 35 18

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 READY 0x7fe2fdd35000 4 35 19

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 READY 0x7fe2fdd35000 4 36 20

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 READY 0x7fe2fdd35000 4 37 21

pid ppid current state base addr nice static total ticks

1 0 SLEEPING 0x7fe2fdd66000 0 20 0

3 1 READY 0x7fe2fdd35000 4 37 22

CHILD 2: DONE y=536870912

Process 3 exited with code 11

Process 1 is now waking up

pid ppid current state base addr nice static total ticks

1 0 RUNNING 0x7fe2fdd66000 0 20 0

3 1 ZOMBIE 0x7fe2fdd35000 4 37 22

Zombie Process 3 killed by Parent Process 1.

init\_fn Finished

Output for test2.c is extremely long and was simply created for stress testing. All processes reached approximately 34 ticks and did so based off of the randomized nice values. The process successfully exited once all processes completed and exited.

**TEST.C**

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 9

// test.c

#include "sched.h"

#include <stdlib.h>

#include <stdio.h>

#define DELAY\_FACTOR 29

void init\_fn()

{

int x;

fprintf(stderr,"Hooray made it to init\_fn, stkaddr %p\n",&x);

switch(sched\_fork())

{

case -1:

perror("fork failed!!");

exit(-1);

case 0:

fprintf(stderr,"<<in child addr %p>>\n",&x);

child\_fn1();

fprintf(stderr,"!!BUG!! at %s:%d\n",\_\_FILE\_\_,\_\_LINE\_\_);

return;

default:

fprintf(stderr,"<<in parent addr %p>>\n",&x);

parent\_fn();

break;

}

printf("init\_fn Finished\n");

exit(0);

}

parent\_fn()

{

int y,p;

fprintf(stderr,"Wow, made it to parent, stkaddr=%p\n",&y);

switch(sched\_fork())

{

case -1:

perror("fork failed!!");

exit(-1);

case 0:

child\_fn2();

sched\_exit(11);

fprintf(stderr,"!!BUG!! at %s:%d\n",\_\_FILE\_\_,\_\_LINE\_\_);

return;

default:

sched\_wait(&y);

return;

}

}

child\_fn1()

{

int y;

fprintf(stderr,"CHILD 1: START &y=%p\n",&y);

for(y=0;y<1<<DELAY\_FACTOR;y++)

;

for(y=0;y<1<<DELAY\_FACTOR;y++)

;

fprintf(stderr,"CHILD 1: DONE y=%d\n",y);

sched\_exit(22);

}

child\_fn2()

{

int y;

sched\_nice(4);

fprintf(stderr,"CHILD 2: START &y=%p\n",&y);

for(y=0;y<1<<DELAY\_FACTOR;y++)

;

for(y=0;y<1<<DELAY\_FACTOR;y++)

;

fprintf(stderr,"CHILD 2: DONE y=%d\n",y);

}

main ()

{

sched\_init(init\_fn);

fprintf(stderr,"Whoops\n");

}

**TEST2.C**

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 9

// test2.c

#include "sched.h"

#include <stdlib.h>

#include <stdio.h>

#define DELAY\_FACTOR 29

void init\_fn()

{

int i,x;

for(i = 0; i < 30; i++)

{

switch(sched\_fork())

{

case -1:

perror("fork failed!!");

exit(-1);

case 0:

fprintf(stderr,"<<in child addr %p>>\n",&x);

child\_fn();

fprintf(stderr,"!!BUG!! at %s:%d\n",\_\_FILE\_\_,\_\_LINE\_\_);

return;

default:

fprintf(stderr,"<<in parent addr %p>>\n",&x);

break;

}

}

sched\_wait(&x);

printf("init\_fn Finished\n");

exit(0);

}

child\_fn()

{

int y;

int nice = 19 - (rand() % 40);

sched\_nice(nice);

for(y=0;y<1<<DELAY\_FACTOR;y++)

;

int pid = sched\_getpid();

sched\_exit(pid);

}

main ()

{

sched\_init(init\_fn);

fprintf(stderr,"Whoops\n");

}

**SCHED.H**

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 9

// sched.h

#ifndef SCHED\_H

#define SCHED\_H

#define SCHED\_NPROC 256

#define SCHED\_READY 0

#define SCHED\_RUNNING 1

#define SCHED\_SLEEPING 2

#define SCHED\_ZOMBIE 4

#define STACK\_SIZE 65536

#include "savectx64.h"

struct sched\_proc {

int state; /\* Task State \*/

int nice; /\* Nice Value \*/

int priority; /\* Static Priority \*/

int accumulated; /\* Accumulated Processing Time \*/

int cpuTime; /\* CPU Time \*/

void \*stack; /\* Stack address \*/

int pid; /\* Process ID \*/

int ppid; /\* Parent Process ID \*/

int code; /\* Exit code \*/

int children; /\* Number of Living Children \*/

struct savectx ctx; /\* For restoring context \*/

};

struct sched\_waitq {

struct sched\_proc \*procs[SCHED\_NPROC];

};

void sched\_init(void (\*init\_fn)());

int sched\_fork();

void sched\_exit(int code);

int sched\_wait(int \*exit\_code);

void sched\_nice(int niceval);

int sched\_getpid();

int sched\_getppid();

int sched\_gettick();

void sched\_ps();

void sched\_switch();

void sched\_tick();

#endif

**SCHED.C**

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 9

// sched.c

#include "sched.h"

#include <stdlib.h>

#include <stdio.h>

#include <errno.h>

#include <signal.h>

#include <string.h>

#include <sys/mman.h>

#include <sys/time.h>

#include <sys/types.h>

// Global Variables

struct sched\_proc \*currentProc;

struct sched\_waitq \*q;

int numTicks;

int NEED\_RESCHED;

sigset\_t mask;

void sched\_init(void (\*init\_fn)())

{

void \*newsp; /\* Stack Area \*/

struct itimerval timer;

struct sigaction sa;

NEED\_RESCHED = 0;

// Initialize the periodic interval timer to tick every 100ms

timer.it\_value.tv\_sec = 0;

timer.it\_value.tv\_usec = 100000;

timer.it\_interval.tv\_sec = 0;

timer.it\_interval.tv\_usec = 100000;

// Set periodic interval timer

if((setitimer(ITIMER\_VIRTUAL, &timer, NULL)) < 0)

{

perror("setitimer failed");

exit(-1);

}

sigemptyset(&mask);

sigaddset(&mask,SIGVTALRM);

sa.sa\_flags=0;

sa.sa\_handler = &sched\_tick;

sigemptyset(&sa.sa\_mask);

sigaction(SIGVTALRM,&sa,NULL);

sa.sa\_handler=sched\_ps;

sigaction(SIGABRT,&sa,NULL);

// Initialize stack area

if ((newsp = mmap(0, STACK\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, 0, 0)) == MAP\_FAILED)

{

perror("mmap failed");

exit(-1);

}

// Malloc first process

if((currentProc = malloc(sizeof(struct sched\_proc))) < 0)

{

perror("malloc failed in sched\_init");

exit(-1);

}

// Malloc Process Queue

if((q = malloc(sizeof(struct sched\_waitq))) < 0)

{

perror("malloc failed in sched\_init");

exit(-1);

}

// Initialize task info

currentProc->state = SCHED\_RUNNING;

currentProc->stack = newsp;

currentProc->pid = 1;

currentProc->ppid = 0;

currentProc->accumulated = 0;

currentProc->cpuTime = 0;

currentProc->nice = 0;

savectx(&(currentProc->ctx));

currentProc->ctx.regs[JB\_BP] = currentProc->stack + STACK\_SIZE;

currentProc->ctx.regs[JB\_SP] = currentProc->stack + STACK\_SIZE;

currentProc->ctx.regs[JB\_PC] = init\_fn;

q->procs[1] = currentProc;

restorectx(&(currentProc->ctx),0);

}

int sched\_fork()

{

// Block Signals

sigprocmask(SIG\_BLOCK,&mask,NULL);

void \*newsp;

struct sched\_proc \*child;

// Assign new pid

int i, newPid = -1;

for(i = 1; i < SCHED\_NPROC; i++)

{

if(!q->procs[i])

{

newPid = i;

break;

}

}

if(newPid == -1)

{

fprintf(stderr, "Maximum Number Of Processes Reached\n");

sigprocmask(SIG\_UNBLOCK,&mask,NULL);

return -1;

}

if((child = malloc(sizeof(struct sched\_proc))) < 0)

{

perror("malloc failed in fork");

exit(-1);

}

// Initialize stack area

if ((newsp = mmap(0, STACK\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, 0, 0)) == MAP\_FAILED)

{

perror("mmap failed");

exit(-1);

}

//Assign values

child->pid = newPid;

child->ppid = currentProc->pid;

child->state = SCHED\_READY;

child->accumulated = 0;

child->stack = newsp;

child->nice = currentProc->nice;

child->cpuTime = currentProc->cpuTime;

child->accumulated = 0;

q->procs[newPid] = child;

if(!savectx(&child->ctx))

{

memcpy(child->stack, currentProc->stack, STACK\_SIZE);

adjstack(child->stack, child->stack + STACK\_SIZE, child->stack - currentProc->stack);

child->ctx.regs[JB\_BP] += child->stack - currentProc->stack;

child->ctx.regs[JB\_SP] += child->stack - currentProc->stack;

}

else

{

sigprocmask(SIG\_UNBLOCK,&mask,NULL);

return 0;

}

sigprocmask(SIG\_UNBLOCK,&mask,NULL);

return child->pid;

}

void sched\_exit(int code)

{

// Block Signals

sigprocmask(SIG\_BLOCK,&mask,NULL);

currentProc->state = SCHED\_ZOMBIE;

currentProc->code = code;

printf("Process %d exited with code %d\n", currentProc->pid, code);

// wake up parent

int i;

for(i = 0; i < SCHED\_NPROC; i++)

{

if(q->procs[i] && q->procs[i]->pid == currentProc->ppid && q->procs[i]->state == SCHED\_SLEEPING)

{

printf("Process %d is now waking up\n", q->procs[i]->pid);

q->procs[i]->state = SCHED\_READY;

break;

}

}

sched\_switch();

}

int sched\_wait(int \*exit\_code)

{

// Block Signals

sigprocmask(SIG\_BLOCK,&mask,NULL);

int i;

int children = 0;

for(i = 0; i < SCHED\_NPROC; i++)

if(q->procs[i])

if(q->procs[i]->ppid == currentProc->pid)

children++;

if(children == 0)

return -1;

// Check for zombie children

for(i=0; i< SCHED\_NPROC; i++)

{

if(q->procs[i] && q->procs[i]->ppid == currentProc->pid)

{

if (q->procs[i]->state == SCHED\_ZOMBIE)

{

printf("Zombie Process %d killed by Parent Process %d.\n", q->procs[i]->pid, currentProc->pid);

\*exit\_code = q->procs[i]->code;

if(munmap(q->procs[i]->stack, STACK\_SIZE) < 0)

{

perror("munmap failed in sched\_wait");

exit(-1);

}

free(q->procs[i]);

q->procs[i] = NULL;

sched\_switch();

}

else

{

currentProc->state = SCHED\_SLEEPING;

sigprocmask(SIG\_BLOCK,&mask,NULL);

sched\_switch();

sigprocmask(SIG\_UNBLOCK,&mask,NULL);

// Woken up because child died

printf("Zombie Process %d killed by Parent Process %d.\n", q->procs[i]->pid, currentProc->pid);

\*exit\_code = q->procs[i]->code;

if(munmap(q->procs[i]->stack, STACK\_SIZE) < 0)

{

perror("munmap failed in sched\_wait");

exit(-1);

}

free(q->procs[i]);

q->procs[i] = NULL;

}

}

}

for(i = 0; i < SCHED\_NPROC; i++)

if(q->procs[i])

if(q->procs[i]->ppid == currentProc->pid)

children++;

if(children == 0)

return -1;

return 0;

}

void sched\_nice(int niceval)

{

if(niceval >= 19)

currentProc->nice = 19;

else if(niceval <=-20)

currentProc->nice = -20;

else

currentProc->nice = niceval;

}

int sched\_getpid()

{

return currentProc->pid;

}

int sched\_getppid()

{

return currentProc->ppid;

}

int sched\_gettick()

{

return numTicks;

}

void sched\_ps()

{

printf("pid\tppid\tcurrent state\tbase addr\tnice\tstatic\ttotal ticks\n");

int i;

for(i = 1; i < SCHED\_NPROC; i++)

{

if(q->procs[i])

{

printf("%d\t", q->procs[i]->pid);

printf("%d\t", q->procs[i]->ppid);

switch(q->procs[i]->state)

{

case SCHED\_READY: printf("READY \t");break;

case SCHED\_RUNNING: printf("RUNNING \t");break;

case SCHED\_SLEEPING:printf("SLEEPING\t");break;

case SCHED\_ZOMBIE: printf("ZOMBIE \t");break;

}

printf("%p\t", q->procs[i]->stack);

printf("%d\t", q->procs[i]->nice);

printf("%d\t", q->procs[i]->priority);

printf("%d\n", q->procs[i]->accumulated);

}

}

}

void sched\_switch()

{

if(NEED\_RESCHED)

NEED\_RESCHED = 0;

// Update priorities

int i;

for(i = 0; i < SCHED\_NPROC; i++)

if(q->procs[i])

{

q->procs[i]->priority = (20 + q->procs[i]->nice) + (10 \* q->procs[i]->accumulated / (20 - q->procs[i]->nice));

if (q->procs[i]->priority > 39)

{

q->procs[i]->priority = 39;

}

}

if((currentProc->state != SCHED\_SLEEPING) && (currentProc->state != SCHED\_ZOMBIE))

{

currentProc->state = SCHED\_READY;

}

int bestPriority = 39, bestIndex = -1;

// Find best new process

for(i = 0; i <SCHED\_NPROC; i++)

if(q->procs[i] && q->procs[i]->state == SCHED\_READY)

if(q->procs[i]->priority < bestPriority)

{

bestPriority = q->procs[i]->priority;

bestIndex = i;

}

if(bestIndex == -1)

{

sched\_ps();

sigprocmask(SIG\_UNBLOCK,&mask,NULL);

return;

}

struct sched\_proc \*bestProc = q->procs[bestIndex];

if(bestProc->pid == currentProc->pid)

{

currentProc->cpuTime = 0;

currentProc->state = SCHED\_READY;

sched\_ps();

sigprocmask(SIG\_UNBLOCK,&mask,NULL);

return;

}

if(savectx(&(currentProc->ctx)) == 0)

{

currentProc = bestProc;

currentProc->cpuTime = 0;

currentProc->state = SCHED\_RUNNING;

sigprocmask(SIG\_UNBLOCK,&mask,NULL);

restorectx(&(currentProc->ctx),1);

}

sched\_ps();

return;

}

void sched\_tick()

{

sigprocmask(SIG\_BLOCK,&mask,NULL);

int window = 0;

// Determine number of processes

int i, numProcs = 0;

for(i = 0; i < SCHED\_NPROC; i++)

if(q->procs[i] && q->procs[i]->state != SCHED\_ZOMBIE)

numProcs++;

if(numProcs != 0)

window = (int)(4/numProcs);

numTicks++;

currentProc->accumulated++;

currentProc->cpuTime++;

//if(currentProc->cpuTime > window){

NEED\_RESCHED = 1;

sigprocmask(SIG\_UNBLOCK,&mask,NULL);

sched\_switch();

//}

sigprocmask(SIG\_UNBLOCK,&mask,NULL);

}