```
#ifndef _SCHED_H_
#define _SCHED_H_

#include "context.h"

#define STACK_SIZE 0xffff

#define SCHED_NPROC    512
#define SCHED_READY    0
#define SCHED_RUNNING  1
#define SCHED_SLEEPING 2
#define SCHED_ZOMBIE   3

struct sched_proc {
    int state;
    int pid;
    int ppid;
    int nice;
    unsigned accumulated;
    unsigned cpu_time;
    void *stack;
    struct savectx context;
    int exit_code;
    int priority;
};

struct sched_waitq {
    struct sched_proc * procs [SCHED_NPROC];
    unsigned nprocs;
};

int create_pid();
void * create_stack();
struct sched_proc * create_proc(int ppid);
int sched_init(void (*init_fn)());
int sched_fork();
void sched_exit(int code);
int sched_wait(int *exit_code);
void sched_nice(int val);
int sched_getpid();
int sched_getppid();
int sched_gettick();
void sched_ps();
int getPriority(int pid);
void sched_switch();
void sched_tick();


#endif /* _SCHED_H_ */
```

```
#include "sched.h"
#include "adjstack.h"
#include "context.h"

#include <sys/time.h>
#include <sys/mman.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <string.h>

#define HOLD_SIGNALS(x) \
    sigprocmask(SIG_BLOCK, &all_signals, NULL); \
    x \
    sigprocmask(SIG_UNBLOCK, &all_signals, NULL);

#define BLOCK_SIGNALS()   sigprocmask(SIG_BLOCK, &all_signals, NULL)
#define UNBLOCK_SIGNALS() sigprocmask(SIG_UNBLOCK, &all_signals, NULL)

struct sched_proc * current = NULL;
struct sched_waitq * queue = NULL;
unsigned nticks;
sigset_t all_signals;

int create_pid() {
    if (queue == NULL) {
        fprintf(stderr, "Running process list has not been allocated!");
        return -1;
    }
    // Skip PID 0, since scheduler should probably have that
    for (unsigned i = 1; i < SCHED_NPROC; i++){
        if (queue->procs[i] == NULL)
            return i;
    }
    fprintf(stderr, "Ran out of PIDs!\n");
    return -1;
}

void * create_stack() {
    void * newsp;
    if ((newsp = mmap(0, STACK_SIZE, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_A
NONYMOUS, 0, 0)) == MAP_FAILED) {
        perror("mmap()");
        return MAP_FAILED;
    }
    return newsp;
}

struct sched_proc * create_proc(int ppid) {
    int pid = create_pid();
    if (pid == -1) {

        return 0;
    }
    struct sched_proc * proc = (struct sched_proc *) malloc(sizeof(struct sched_
proc));
    if (proc == NULL) {
        perror("malloc()");
```

```
        return NULL;
    }
    void * newsp = create_stack();
    if (newsp == -1) {
        free(proc);
        return NULL;
    }

    // Set contents
    proc->state       = SCHED_READY;
    proc->pid         = pid;
    proc->ppid        = ppid;
    proc->nice        = 0;
    proc->accumulated = 0;
    proc->cpu_time    = 0;
    proc->stack       = newsp;

    queue->procs[pid] = proc;
    queue->nprocs++;

    return proc;
}

int sched_init(void (*init_fn)()) {
    // Initialize process queue
    queue = (struct sched_waitq *) calloc(1, sizeof(struct sched_waitq));
    if (queue == NULL) {
        perror("malloc()");
        return -1;
    }

    // Set up ticks
    sigfillset(&all_signals);
    nticks = 0;
    struct timeval t;
    t.tv_sec = 0;
    t.tv_usec = 1e5;
    struct itimerval it;
    it.it_interval = t;
    it.it_value = t;
    if(setitimer(ITIMER_VIRTUAL, &it, NULL) == -1) {
        perror("Failed to set timer");
        return -1;
    }
    signal(SIGVTALRM, sched_tick);

    // Create init process and switch control
    current = create_proc(0);
    struct savectx ctx;
    ctx.regs[JB_BP] = current->stack + STACK_SIZE;
    ctx.regs[JB_SP] = current->stack + STACK_SIZE;
    ctx.regs[JB_PC] = init_fn;
    restorectx(&ctx, current->pid);

    return 0;
}

int sched_fork() {
    int ret;
```

```
    HOLD_SIGNALS(
        struct sched_proc * newproc = create_proc(current->pid);
        newproc->cpu_time = current->cpu_time;
        memcpy(newproc->stack, current->stack, STACK_SIZE);
        adjstack(newproc->stack, newproc->stack + STACK_SIZE, newproc->stack - c
urrent->stack);

        ret = savectx(&newproc->context);
        if (ret == 0) {
            newproc->context.regs[JB_BP] += newproc->stack - current->stack;
            newproc->context.regs[JB_SP] += newproc->stack - current->stack;
        }

    );
    if (ret == 0) {
        return newproc->pid;
    } else {
        current = queue->procs[ret];
        return 0;
    }
}

void sched_exit(int code) {
    HOLD_SIGNALS(
        current->state = SCHED_ZOMBIE;
        current->exit_code = code;
        queue->nprocs--;

        // Check for wait()
        if ((queue->procs[current->ppid] != NULL)
         && (queue->procs[current->ppid]->state == SCHED_SLEEPING)) {
            queue->procs[current->ppid]->state = SCHED_READY;
        }
    );
    sched_switch();
}

int sched_wait(int *exit_code) {
    while(1) {
        BLOCK_SIGNALS();
        int found = 0;
        int zombie = 0;
        found = 0;
        zombie = 0;
        for (int i = 1; i < SCHED_NPROC; i++) {
            if (queue->procs[i] != NULL
             && queue->procs[i]->ppid == current->pid) {
                found = 1;
                if (queue->procs[i]->state == SCHED_ZOMBIE) {
                    zombie = 1;
                    current->state = SCHED_READY;
                    *exit_code = queue->procs[i]->exit_code;
                    free(queue->procs[i]);
                    queue->procs[i] = NULL;
                    break;
                }
            }

        }
```

```
        if (found == 0) {
            UNBLOCK_SIGNALS();
            return -1;
        } else if (found == 1 && zombie == 1) {
            UNBLOCK_SIGNALS();
            return 0;
        } else {
            UNBLOCK_SIGNALS();
            current->state = SCHED_SLEEPING;
            sched_switch();
        }
    }
}

void sched_nice(int val) {
    val = val >  19 ?  19 : val;
    val = val < -20 ? -20 : val;
    current->nice = val;
}

int sched_getpid() {
    return current->pid;
}

int sched_getppid() {
    return current->ppid;
}

int sched_gettick() {
    return nticks;
}

void sched_ps() {
    printf("PID   PPID   STATE   STACK   STATIC   DYNAMIC   TIME   \n");
    for (int i = 1; i < SCHED_NPROC; i++) {
        if (queue->procs[i] != NULL) {
            printf("%-9d %-9d %-9d %-9x %-9d %-9d %-9d\n",
                queue->procs[i]->pid,
                queue->procs[i]->ppid,
                queue->procs[i]->state,
                (unsigned)queue->procs[i]->stack,
                20 - queue->procs[i]->nice,
                queue->procs[i]->priority,
                queue->procs[i]->accumulated);
        }
    }

}

int getPriority(int pid) {
    if (queue->procs[pid] != NULL) {
        int priority = 20 - queue->procs[pid]->nice - (int)queue->procs[pid]->ac
cumulated/2;
        queue->procs[pid]->priority = priority;
        return priority;
    } else {
        return -1;
    }
```

```
}

void sched_switch() {
    BLOCK_SIGNALS();
    if (current->state != SCHED_ZOMBIE && current->state != SCHED_SLEEPING)
        current->state = SCHED_READY;

    int best = -1;
    int best_pid = -1;
    int p;
    for (int pid = 1; pid < SCHED_NPROC; pid++) {
        if ((queue->procs[pid] != NULL)
         && (queue->procs[pid]->state == SCHED_READY)
         && ((p = getPriority(pid)) > best)) {
            best = p;
            best_pid = pid;
        }
    }

    if (best_pid == -1)
        return;

    if (best_pid == current->pid) {
        current->state = SCHED_RUNNING;
        current->cpu_time = 0;
        UNBLOCK_SIGNALS();
        return;
    }

    if(savectx(&current->context) == 0) {
        current = queue->procs[best_pid];
        current->cpu_time = 0;
        current->state = SCHED_RUNNING;
        UNBLOCK_SIGNALS();
        restorectx(&current->context, current->pid);
    }
}

void sched_tick() {
    sched_ps();
    nticks++;
    current->accumulated++;
    current->cpu_time++;
    sched_switch();
    return;
}
```

```
#include "sched.h"

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

void child() {
  // priority: 6, 4, 5, 3, 2
  switch (sched_getpid()) {
    case 2:
      sched_nice(10);
      break;
    case 3:
      sched_nice(8);
      break;
    case 4:
      sched_nice(4);
      break;
    case 5:
      sched_nice(6);
      break;
    case 6:
      sched_nice(2);
      break;
  }
  printf("In child with pid %d, ppid %d\n", sched_getpid(), sched_getppid());
  unsigned long max = 1e8;
  for (unsigned long i = 0; i < max; i++) {
    getpid();
  }
  sched_exit(sched_getpid());
}

void init() {
  printf("Hello\n");
  int pid;
  for (int i = 0; i < 5; i++) {
    switch(pid = sched_fork()) {
      case -1:
        printf("sched_fork() error\n");
        sched_exit(0);
      case 0:
        child();
      default:
        printf("Created pid %d\n", pid);
        break;
    }
  }
  printf("In parent with pid %d\n", sched_getpid());
  int code;
  int order [5];
  for (int i = 0; i < 5; i++) {
    sched_wait(&code);
    order[i] = code;
    printf("Child returned with code %d\n", code);
  }
  printf("Children returned in order:");
  for (int i = 0; i < 5; i++)
```

```
    printf(" %d", order[i]);
  printf("\n");
  sched_exit(0);
  printf("Ending init process...\n");
  exit(0);
}

int main(int argc, char **argv) {
  sched_init(init);
}
```

```
Hello
Created pid 2
Created pid 3
Created pid 4
Created pid 5
Created pid 6
In parent with pid 1
In child with pid 2, ppid 1
PID       PPID      STATE     STACK     STATIC    DYNAMIC   TIME
1         0         2         49f98000  20        0         0
2         1         1         49f7a000  10        20        0
3         1         0         49f6a000  20        20        0
4         1         0         49f5a000  20        20        0
5         1         0         49f4a000  20        20        0
6         1         0         49f3a000  20        20        0
In child with pid 3, ppid 1
PID       PPID      STATE     STACK     STATIC    DYNAMIC   TIME
1         0         2         49f98000  20        0         0
2         1         0         49f7a000  10        10        1
3         1         1         49f6a000  12        20        0
4         1         0         49f5a000  20        20        0
5         1         0         49f4a000  20        20        0
6         1         0         49f3a000  20        20        0
In child with pid 4, ppid 1
PID       PPID      STATE     STACK     STATIC    DYNAMIC   TIME
1         0         2         49f98000  20        0         0
2         1         0         49f7a000  10        10        1
3         1         0         49f6a000  12        12        1
4         1         1         49f5a000  16        20        0
5         1         0         49f4a000  20        20        0
6         1         0         49f3a000  20        20        0
In child with pid 5, ppid 1
PID       PPID      STATE     STACK     STATIC    DYNAMIC   TIME
1         0         2         49f98000  20        0         0
2         1         0         49f7a000  10        10        1
3         1         0         49f6a000  12        12        1
4         1         0         49f5a000  16        16        1
5         1         1         49f4a000  14        20        0
6         1         0         49f3a000  20        20        0
In child with pid 6, ppid 1
PID       PPID      STATE     STACK     STATIC    DYNAMIC   TIME
1         0         2         49f98000  20        0         0
2         1         0         49f7a000  10        10        1
3         1         0         49f6a000  12        12        1
4         1         0         49f5a000  16        16        1
5         1         0         49f4a000  14        14        1
6         1         1         49f3a000  18        20        0

...

PID       PPID      STATE     STACK     STATIC    DYNAMIC   TIME
1         0         2         49f98000  20        0         0
2         1         0         49f7a000  10        10        1
3         1         0         49f6a000  12        10        4
4         1         0         49f5a000  16        10        12
5         1         0         49f4a000  14        10        8
6         1         1         49f3a000  18        11        14
PID       PPID      STATE     STACK     STATIC    DYNAMIC   TIME
1         0         2         49f98000  20        0         0
```

```
2       1       0       49f7a000  10      10      1
3       1       0       49f6a000  12      10      4
4       1       0       49f5a000  16      10      12
5       1       0       49f4a000  14      10      8
6       1       1       49f3a000  18      11      15
PID     PPID    STATE   STACK     STATIC  DYNAMIC TIME
1       0       2       49f98000  20      0       0
2       1       1       49f7a000  10      10      1
3       1       0       49f6a000  12      10      4
4       1       0       49f5a000  16      10      12
5       1       0       49f4a000  14      10      8
6       1       0       49f3a000  18      10      16
PID     PPID    STATE   STACK     STATIC  DYNAMIC TIME
1       0       2       49f98000  20      0       0
2       1       0       49f7a000  10      9       2
3       1       1       49f6a000  12      10      4
4       1       0       49f5a000  16      10      12
5       1       0       49f4a000  14      10      8
6       1       0       49f3a000  18      10      16
PID     PPID    STATE   STACK     STATIC  DYNAMIC TIME
1       0       2       49f98000  20      0       0
2       1       0       49f7a000  10      9       2
3       1       1       49f6a000  12      10      5
4       1       0       49f5a000  16      10      12
5       1       0       49f4a000  14      10      8
6       1       0       49f3a000  18      10      16
PID     PPID    STATE   STACK     STATIC  DYNAMIC TIME
1       0       2       49f98000  20      0       0
2       1       0       49f7a000  10      9       2
3       1       0       49f6a000  12      9       6
4       1       1       49f5a000  16      10      12
5       1       0       49f4a000  14      10      8
6       1       0       49f3a000  18      10      16
PID     PPID    STATE   STACK     STATIC  DYNAMIC TIME
1       0       2       49f98000  20      0       0
2       1       0       49f7a000  10      9       2
3       1       0       49f6a000  12      9       6
4       1       1       49f5a000  16      10      13
5       1       0       49f4a000  14      10      8
6       1       0       49f3a000  18      10      16
PID     PPID    STATE   STACK     STATIC  DYNAMIC TIME
1       0       2       49f98000  20      0       0
2       1       0       49f7a000  10      9       2
3       1       0       49f6a000  12      9       6
4       1       0       49f5a000  16      9       14
5       1       1       49f4a000  14      10      8
6       1       0       49f3a000  18      10      16
PID     PPID    STATE   STACK     STATIC  DYNAMIC TIME
1       0       2       49f98000  20      0       0
2       1       0       49f7a000  10      9       2
3       1       0       49f6a000  12      9       6
4       1       0       49f5a000  16      9       14
5       1       1       49f4a000  14      10      9
6       1       0       49f3a000  18      10      16
PID     PPID    STATE   STACK     STATIC  DYNAMIC TIME
1       0       2       49f98000  20      0       0
2       1       0       49f7a000  10      9       2
3       1       0       49f6a000  12      9       6
4       1       0       49f5a000  16      9       14
```

```
5          1          0          49f4a000  14         9          10
6          1          1          49f3a000  18         10         16

...

Child returned with code 3
PID        PPID       STATE      STACK     STATIC     DYNAMIC    TIME
1          0          2          49f98000  20         20         0
2          1          1          49f7a000  10         1          18
PID        PPID       STATE      STACK     STATIC     DYNAMIC    TIME
1          0          2          49f98000  20         20         0
2          1          1          49f7a000  10         1          19
PID        PPID       STATE      STACK     STATIC     DYNAMIC    TIME
1          0          2          49f98000  20         20         0
2          1          1          49f7a000  10         0          20
Child returned with code 2
Children returned in order: 6 4 5 3 2
Ending init process...
```