Project #2: Scheduling

Instructor: Sungyong Ahn

부산대학교
PUSAN NATIONAL UNIVERSITY

Process Control Block in xv6

■ **`struct proc`** in proc.h

```
35 enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
36
37 // Per-process state
38 struct proc {
39   uint sz;                     // Size of process memory (bytes)
40   pde_t* pgdir;                // Page table
41   char *kstack;                // Bottom of kernel stack for this process
42   enum procstate state;        // Process state
43   int pid;                     // Process ID
44   struct proc *parent;         // Parent process
45   struct trapframe *tf;        // Trap frame for current syscall
46   struct context *context;     // swtch() here to run process
47   void *chan;                  // If non-zero, sleeping on chan
48   int killed;                  // If non-zero, have been killed
49   struct file *ofile[NOFILE];  // Open files
50   struct inode *cwd;           // Current directory
51   char name[16];               // Process name (debugging)
52 };
53
54 // Process memory is laid out contiguously, low addresses first:
55 //   text
56 //   original data and bss
57 //   fixed-size stack
58 //   expandable heap
```

Process Control Block in xv6

■ **`struct ptable` in proc.c**

```
10 struct {
11   struct spinlock lock;
12   struct proc proc[NPROC];
13 } ptable;
```

■ **in param.h**

```
1 #define NPROC        64    // maximum number of processes
```

■ **Process states (`procstate` in proc.h)**

  ▪ UNUSED: Not used

  ▪ EMBRYO: Newly allocated (not ready for running yet)

  ▪ SLEEPING: Waiting for I/O, child process, or time

  ▪ RUNNABLE: Ready to run

  ▪ RUNNING: Running on CPU

  ▪ ZOMBIE: Exited

Xv6 Process Scheduler

■ `main()` in `main.c`

```
int
main(void)
{
  userinit();        // first user process
  mpmain();          // finish this processor's setup
}
```

■ `mpmain()` in `main.c`

```
// Common CPU setup code.
static void
mpmain(void)
{
  cprintf("cpu%d: starting %d\n", cpuid(), cpuid());
  idtinit();         // load idt register
  xchg(&(mycpu()->started), 1); // tell startothers() we're up
  scheduler();       // start running processes
}
```

Xv6 Process Scheduler (Cont'd)

■ **`scheduler()` in proc.c**

- Round-robin fashion

Choose a next process →

Start to execute chosen process →

```c
322 void
323 scheduler(void)
324 {
325   struct proc *p;
326   struct cpu *c = mycpu();
327   c->proc = 0;
328
329   for(;;){
330     // Enable interrupts on this processor.
331     sti();
332
333     // Loop over process table looking for process to run.
334     acquire(&ptable.lock);
335     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
336       if(p->state != RUNNABLE)
337         continue;
338
339       // Switch to chosen process.  It is the process's job
340       // to release ptable.lock and then reacquire it
341       // before jumping back to us.
342       c->proc = p;
343       switchuvm(p);
344       p->state = RUNNING;
345
346       swtch(&(c->scheduler), p->context);
347       switchkvm();
348
349       // Process is done running for now.
350       // It should have changed its p->state before coming back.
351       c->proc = 0;
352     }
353     release(&ptable.lock);
354
355   }
356 }
```

Xv6 Process Scheduler (Cont'd)

■ **`swtch()`** in swtch.S

Read Ch.5 Scheduling
of xv6 commentary book!!

in proc.h

```
struct context {
  uint edi;
  uint esi;
  uint ebx;
  uint ebp;
  uint eip;
};
```

```asm
 9  .globl swtch
10  swtch:
11    movl 4(%esp), %eax
12    movl 8(%esp), %edx
13
14    # Save old callee-save registers
15    pushl %ebp
16    pushl %ebx
17    pushl %esi
18    pushl %edi
19
20    # Switch stacks
21    movl %esp, (%eax)
22    movl %edx, %esp
23
24    # Load new callee-save registers
25    popl %edi
26    popl %esi
27    popl %ebx
28    popl %ebp
29    ret
```

Xv6 Entering Scheduler

■ **When?**

1. Exiting process (`exit()` in proc.c)

```
264     // Jump into the scheduler, never to return.
265     curproc->state = ZOMBIE;
266     sched();
```

2. Sleeping process (`sleep()` in proc.c)

```
438     // Go to sleep.
439     p->chan = chan;
440     p->state = SLEEPING;
441
442     sched();
```

Xv6 Entering Scheduler (Cont'd)

■ **When?**

3. Yielding CPU due to timer interrupt

- `trap()` in trap.c

```
103    // Force process to give up CPU on clock tick.
104    // If interrupts were on while locks held, would need to check nlock.
105    if(myproc() && myproc()->state == RUNNING &&
106      tf->trapno == T_IRQ0+IRQ_TIMER)
107      yield();
```
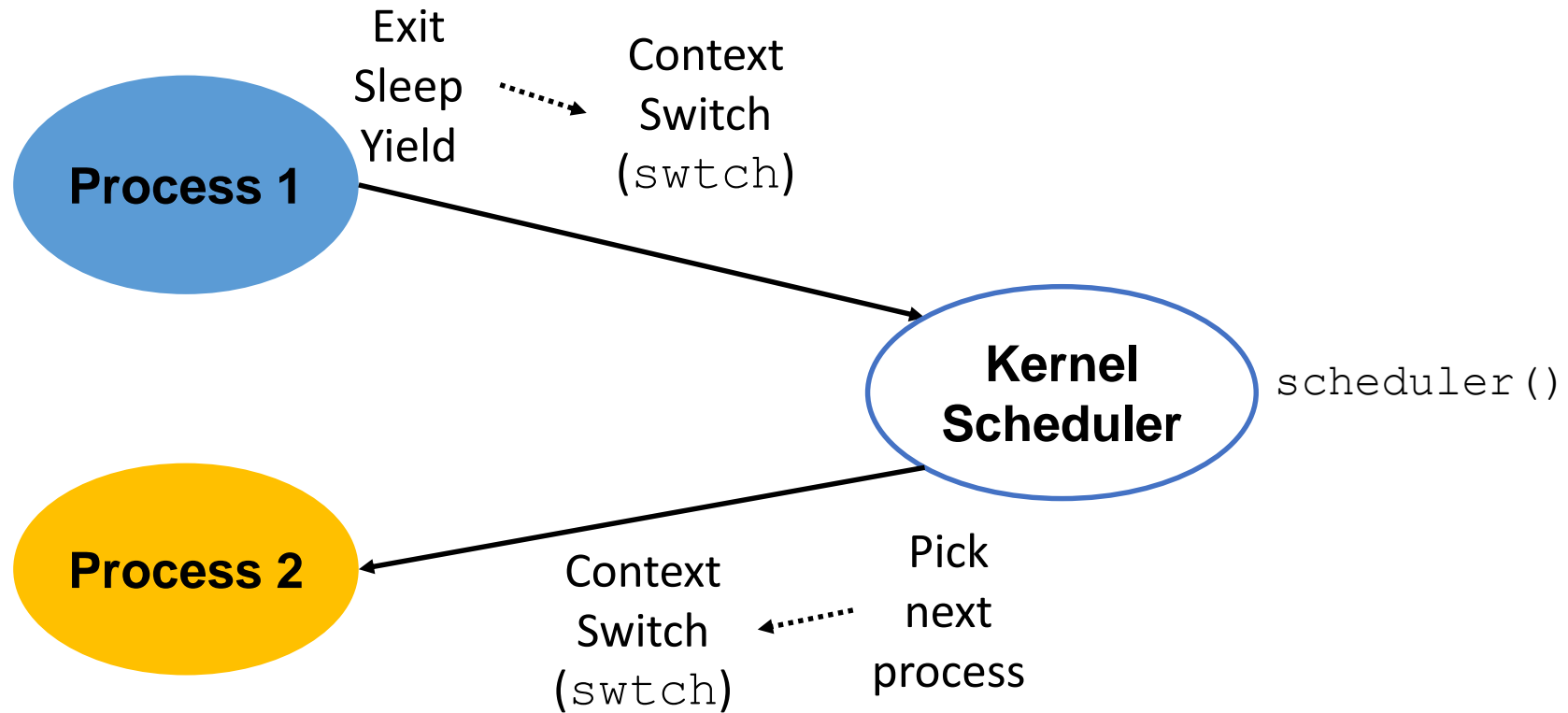
- `yield()` in proc.c

```
384 // Give up the CPU for one scheduling round.
385 void
386 yield(void)
387 {
388   acquire(&ptable.lock);  //DOC: yieldlock
389   myproc()->state = RUNNABLE;
390   sched();
391   release(&ptable.lock);
392 }
```

Xv6 Entering Scheduler (Cont'd)

■ **`sched()`** in proc.c

```c
358 // Enter scheduler.  Must hold only ptable.lock
359 // and have changed proc->state. Saves and restores
360 // intena because intena is a property of this
361 // kernel thread, not this CPU. It should
362 // be proc->intena and proc->ncli, but that would
363 // break in the few places where a lock is held but
364 // there's no process.
365 void
366 sched(void)
367 {
368   int intena;
369   struct proc *p = myproc();
370
371   if(!holding(&ptable.lock))
372     panic("sched ptable.lock");
373   if(mycpu()->ncli != 1)
374     panic("sched locks");
375   if(p->state == RUNNING)
376     panic("sched running");
377   if(readeflags()&FL_IF)
378     panic("sched interruptible");
379   intena = mycpu()->intena;
380   swtch(&p->context, mycpu()->scheduler);
381   mycpu()->intena = intena;
382 }
```

Xv6 Scheduler Summary

Process 1

Exit
Sleep
Yield

Context
Switch
(`swtch`)

Kernel
Scheduler

`scheduler()`

Process 2

Context
Switch
(`swtch`)

Pick
next
process

Project #2. Priority Scheduler

1. **Implement system calls related to process priority**
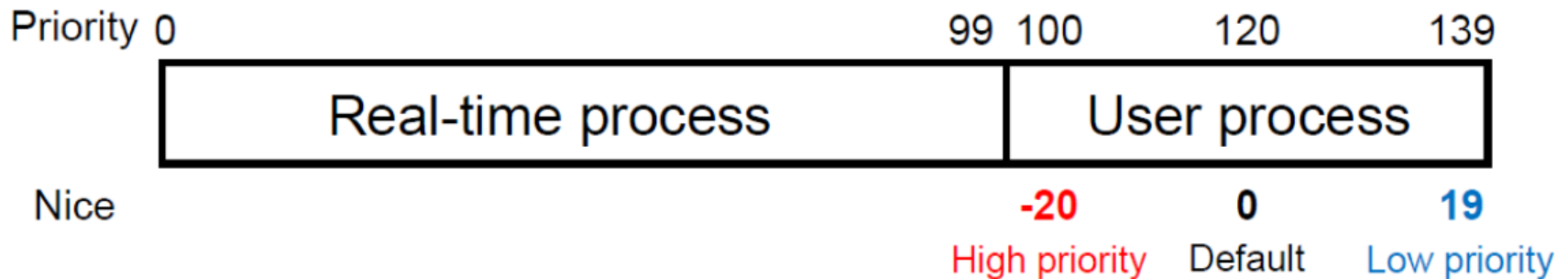   - **`setnice(), getnice(), ps()`**

2. **Implement priority-based scheduler on xv6**
    - The lower nice value, the higher priority
    - The highest priority process is selected for next running
        - Tiebreak: FIFO fashion

Project #2-1: Process Priority

■ **For represent weight between processes**

■ **In Linux**

■ **In Xv6**

- Not implemented!

Project #2-1: Process Priority

■ **Make three system calls (`getnice, setnice, ps`)**

■ **Description of nice value**

- The default nice value is 20.

- The range of valid nice value is [0, 40].

- Lower nice values cause more favorable scheduling.

- When a process calls `fork()` system call, the nice value of child process is same as its parent process.

Project #2-1: Process Priority

■ **Make three system calls (`getnice, setnice, ps`)**

■ **`int setnice(int pid, int nice_value)`**
- Get the nice value of process (min: 0 / max: 40)
- Return 0 on success.
- Return -1 if there is no process corresponding to the pid or the nice value is invalid.

■ **`int getnice(int pid)`**
- Return the nice value of pid(process)
- Return -1 if there is no process corresponding to the pid.

Project #2-1: Process Priority

■ **`void ps(int pid)`**

- The `ps` function prints out process(s)'s information, which includes **pid, ppid, nice, status,** and **name of each process**. If the pid is 0, print out all processes' information. Otherwise, print out corresponding process's information. If there is no process corresponding to the pid, print out nothing.

- No return value

- Make a user program (**minitop**) displays process status using `ps` system call

```
qemu-system-i386 -nographic -drive file=
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nl
init: starting sh
$ minitop
pid     ppid     prio     state     name
1       1        20       sleep     init
2       1        20       sleep     sh
3       2        20       run       minitop
$
```

■ **Reading** Chap. 3 **of xv6-commentary will help your project**

Project #2-1: Process Priority

- **To-do list for process priority**
  - Create a variable for a nice value
  - Initialization code for a nice value
    - What is default nice value?
    - How can child process inherit the nice value of parent process?
- **Then, implement three system calls related to nice value**
  - `setnice(), getnice(), ps()`

`fork()`

- **in proc.c**

```
180 int
181 fork(void)
182 {
183   int i, pid;
184   struct proc *np;
185   struct proc *curproc = myproc();
186
187   // Allocate process.
188   if((np = allocproc()) == 0){
189     return -1;
190   }
191
192   // Copy process state from proc.
193   if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
194     kfree(np->kstack);
195     np->kstack = 0;
196     np->state = UNUSED;
197     return -1;
198   }
199   np->sz = curproc->sz;
200   np->parent = curproc;
201   *np->tf = *curproc->tf;
```

Process Dump in proc.c

```c
//PAGEBREAK: 36
// Print a process listing to console.   For debugging.
// Runs when user types ^P on console.
// No lock to avoid wedging a stuck machine further.
void
procdump(void)
{
  static char *states[] = {
  [UNUSED]     "unused",
  [EMBRYO]     "embryo",
  [SLEEPING]   "sleep ",
  [RUNNABLE]   "runble",
  [RUNNING]    "run   ",
  [ZOMBIE]     "zombie"
  };
  int i;
  struct proc *p;
  char *state;
  uint pc[10];

  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->state == UNUSED)
      continue;
    if(p->state >= 0 && p->state < NELEM(states) && states[p->state])
      state = states[p->state];
    else
      state = "???";
    cprintf("%d %s %s", p->pid, state, p->name);
    if(p->state == SLEEPING){
      getcallerpcs((uint*)p->context->ebp+2, pc);
      for(i=0; i<10 && pc[i] != 0; i++)
        cprintf(" %p", pc[i]);
    }
    cprintf("\n");
  }
}
```

Process Dump in proc.c


```
user@ubuntu:~/xv6-SSE3044$ make qemu-nox
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.01316 s, 389 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000992285 s, 516 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
334+1 records in
334+1 records out
171164 bytes (171 kB, 167 KiB) copied, 0.000756917 s, 226 MB/s
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive f
ormat=raw -smp 2 -m 512
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ 1 sleep  init 80103dc7 80103e69 80104837 801058d9 80105638
2 sleep  sh 80103d8c 801002c2 80100f8c 80104b32 80104837 801058d9 80105638

$
```

Project #2-2. Priority Scheduler

■ **Ignore to yield CPU on clock tick (`trap()` in trap.c)**

```
// Force process to give up CPU on clock tick.
// If interrupts were on while locks held, would need to check nlock.
/*
if(myproc && myproc->state == RUNNING && tf->trapno == T_IRQ0+IRQ_TIMER)
  yield();
*/
```

■ **Entering scheduler when**
   1. Exiting process
   2. Sleeping process
   3. Yielding CPU
   4. Changing priority (setnice)

Sample Test Program

```c
#include "types.h"
#include "stat.h"
#include "user.h"

int main(int argc, char **argv){
    int pid;
    int mypid;

    setnice(1, 19);
    setnice(getpid(), 2);

    pid = fork();

    if(pid == 0){    //Child
        printf(1, "##### State 2 #####\n");
    }
    else{           //Parent
        setnice(pid, 10);    //Set nice value of Child
        printf(1, "##### State 1 #####\n");
        wait();              //Scheduling
        printf(1, "##### State 3 #####\n");
    }

    mypid = getpid();
    printf(1, "PID %d is finished\n", mypid);

    exit();
}
```

```
$ test2
##### State 1 #####
##### State 2 #####
PID 4 is finished
##### State 3 #####
PID 3 is finished
```

부산대학교
PUSAN NATIONAL UNIVERSITY

Project #2. Template Code

- **Download xv6-pnu-p2.tar.gz from PLATO**
- **Modifications**
  - CPUS=1 (`Makefile`)

  ```
  ifndef CPUS
  CPUS := 1
  endif
  ```

  - Ignore to yield CPU on clock tick
  - `yield` system call
    - Yield CPU

Submission

- **Submit your 학번-p2.tar.gz file through PLATO**
  - `$tar -czvf StudentID-p2.tar.gz ./xv6-pnu-p2-학번`
  - Please command `$make clean` before compressing

- **Submit your 학번-p2.patch file through PLATO**
  - `$diff -urN ./xv6-pnu-2 ./xv6-pnu-p2-학번 > 학번-p2.patch`

- **Due date: 4/29 (Wed), 23:59**
  - -25% penalty of total mark per day

- **PLEASE DO NOT COPY !!**
  - YOU WILL GET F IF YOU COPIED