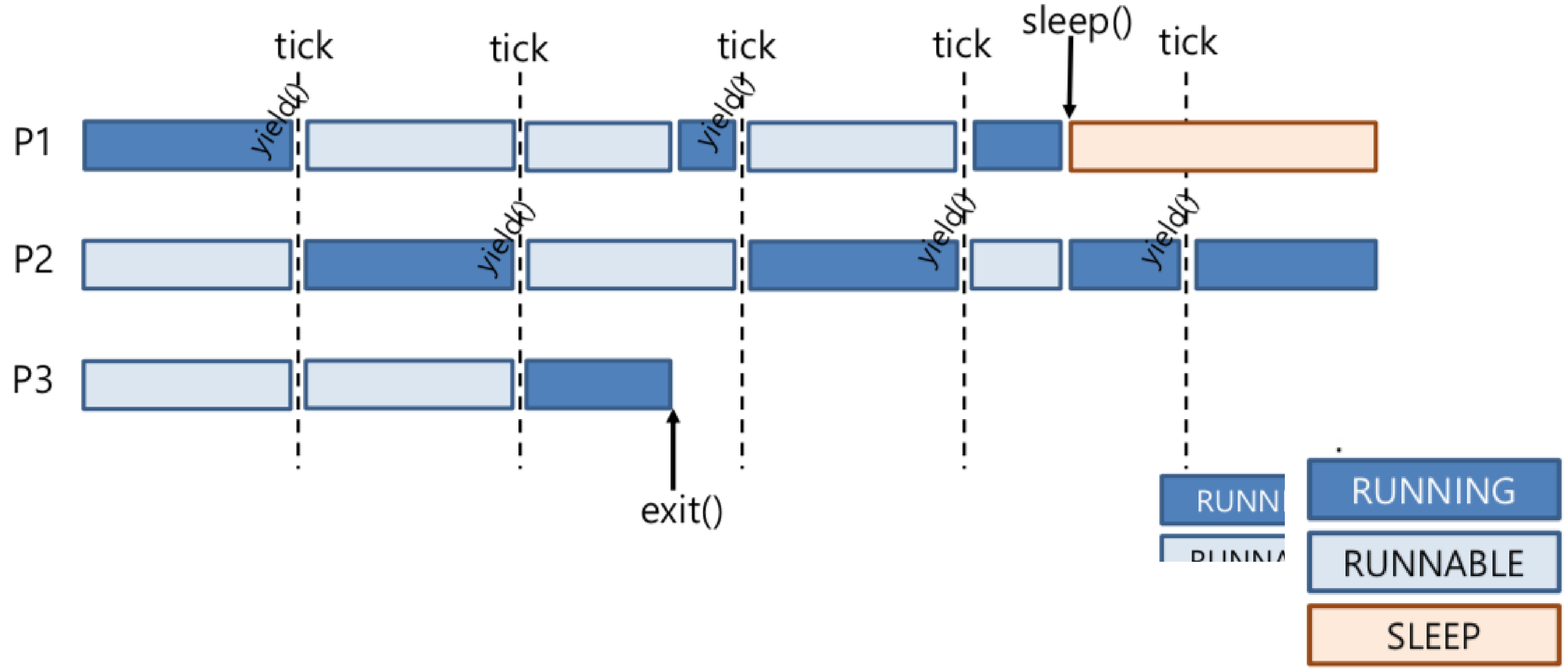
Operating Systems Practice

3: Scheduler

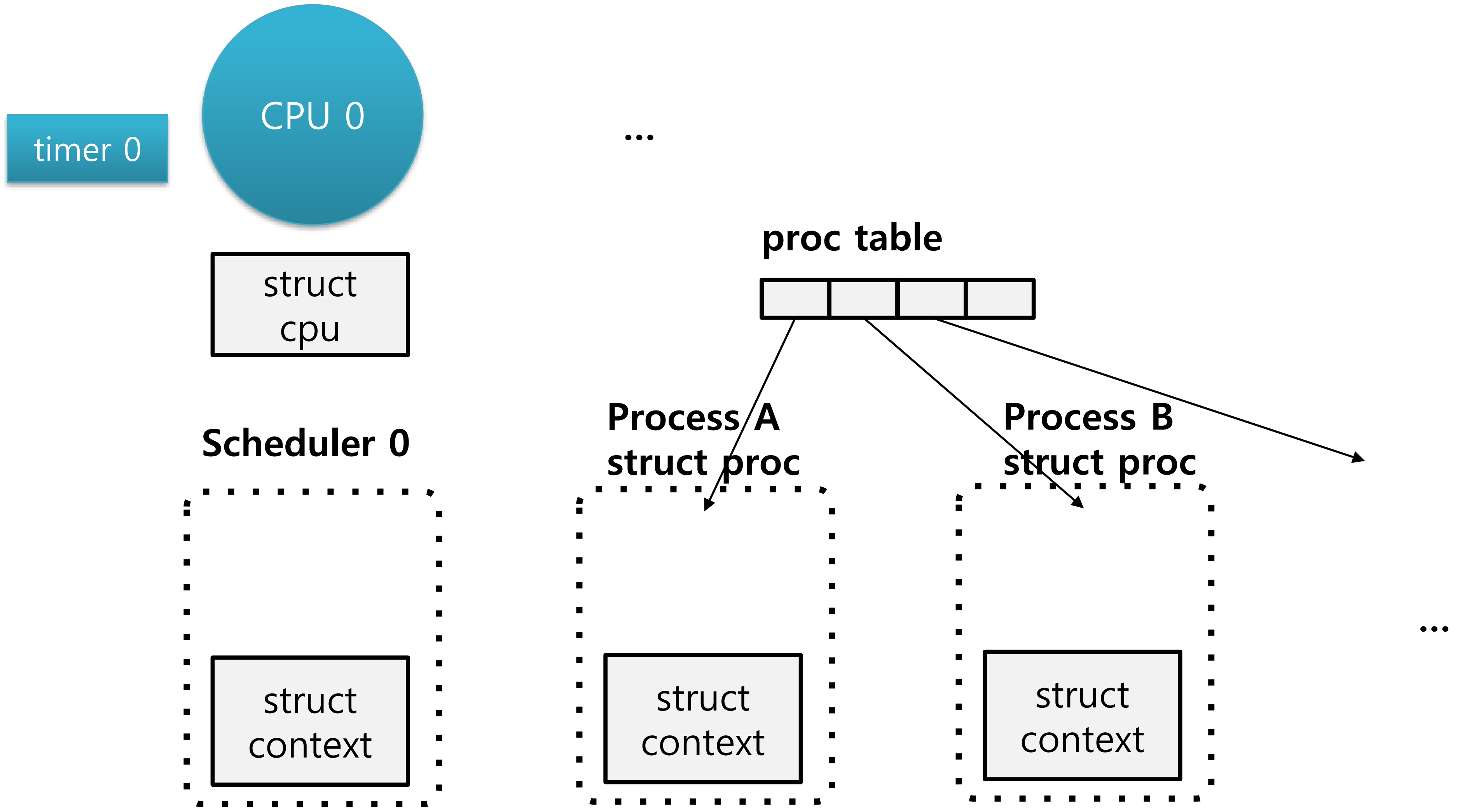
Kilho Lee

XV6 Scheduler – Round Robin

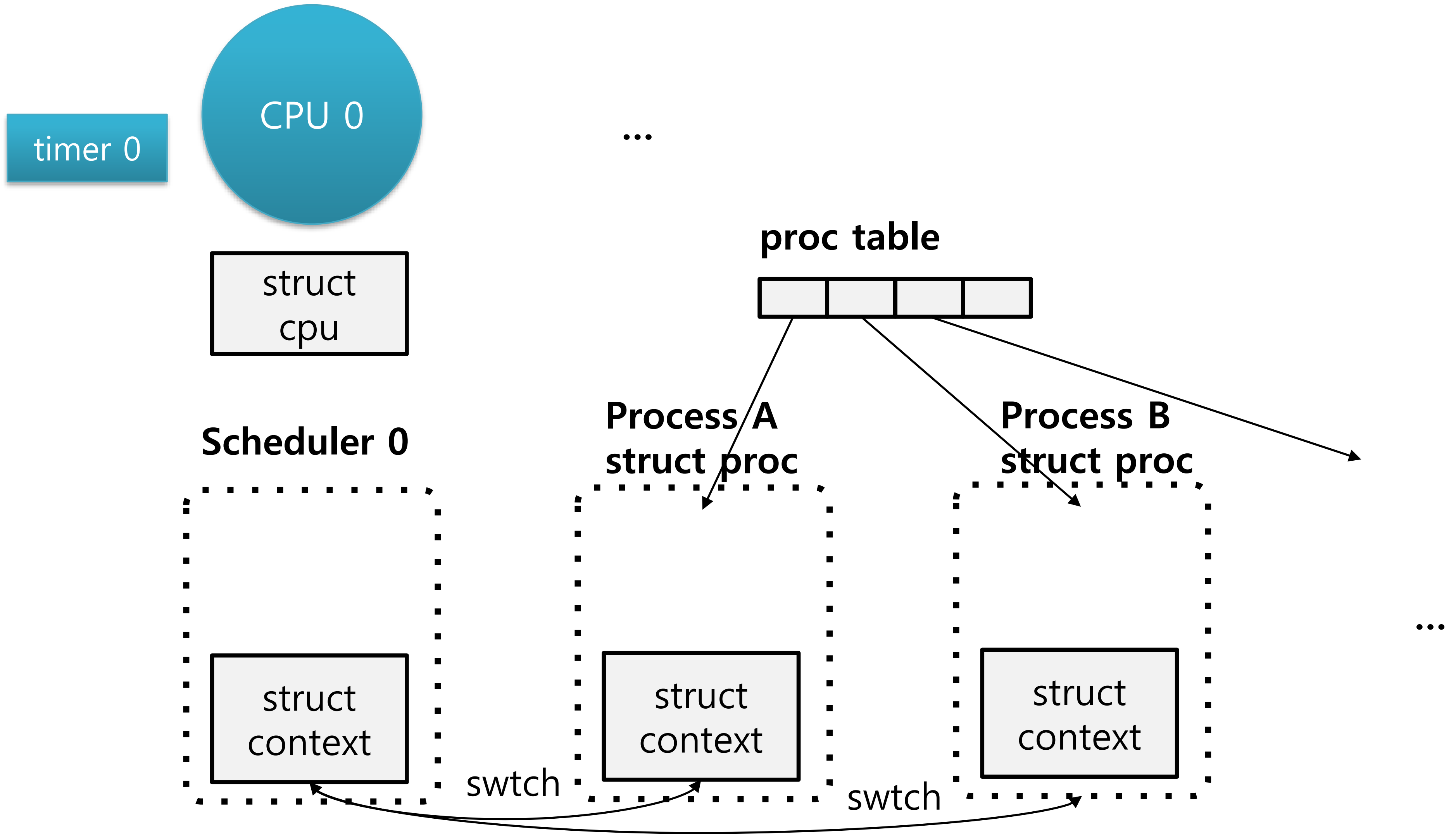
- Timer's interrupt request (IRQ) enforces an yield of a CPU
- A "RUNNABLE" process is chosen to be run in a round-robin manner



XV6 Data Structures for Scheduling

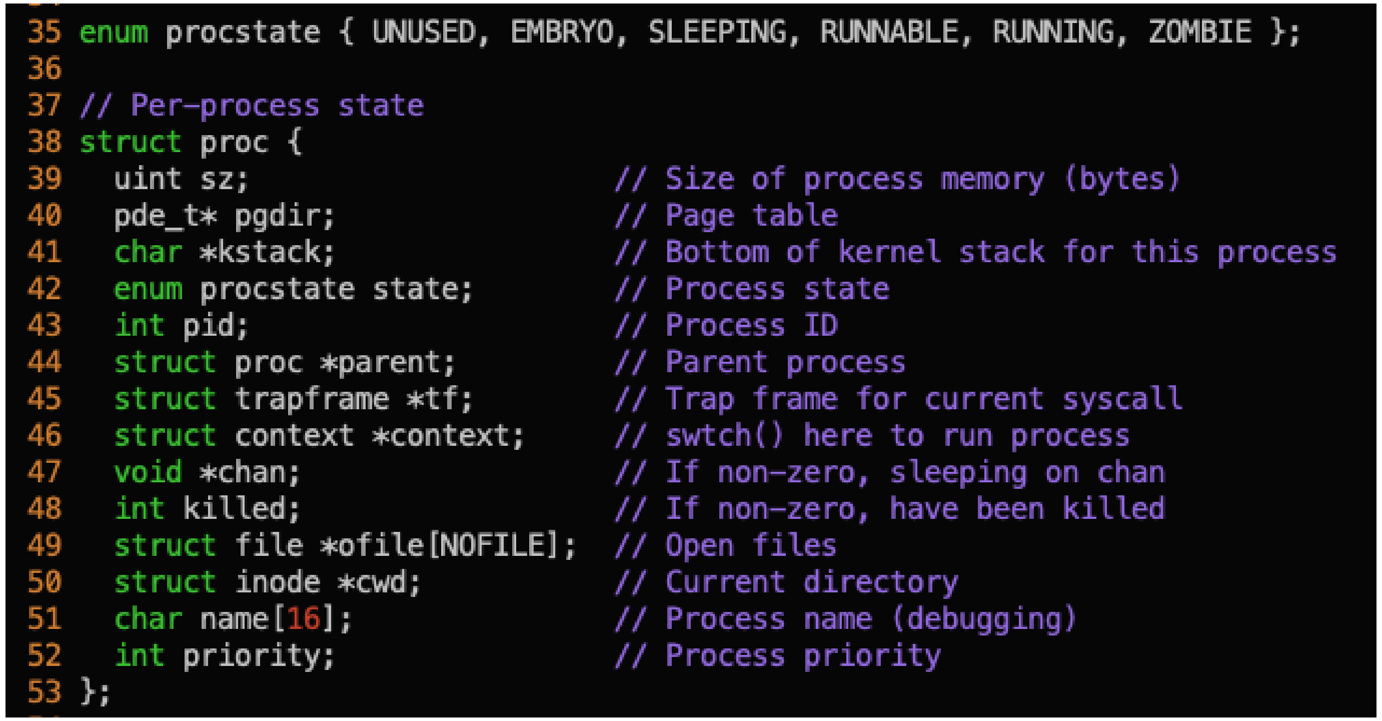


XV6 Data Structures for Scheduling



Process

- proc.h
- procstate
- struct proc



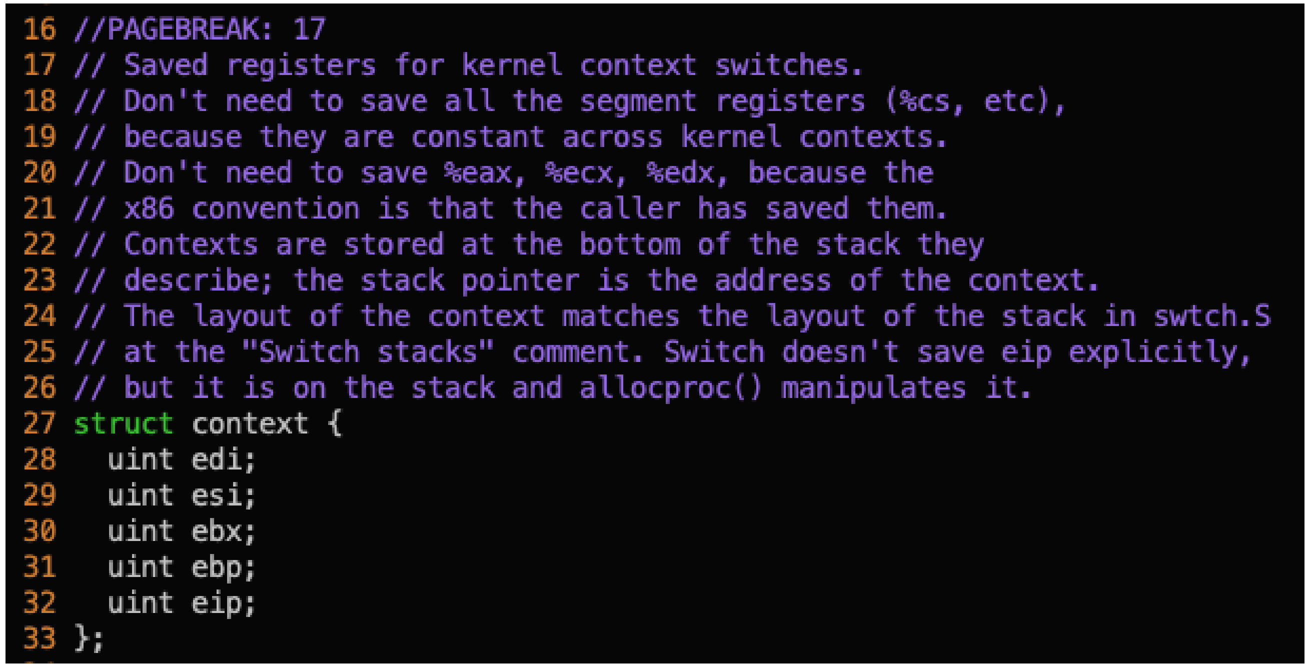
```
enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };

// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  int priority;                // Process priority
};
```

Proc State

- 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

Process

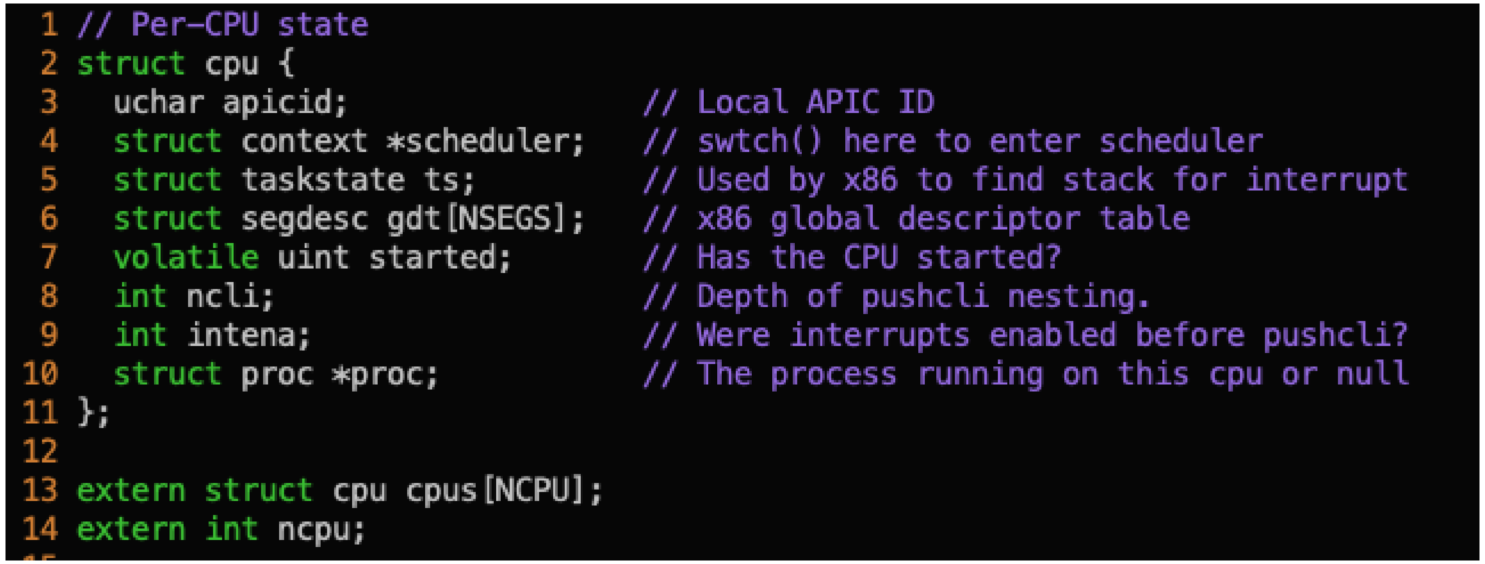
- proc.h
- struct context



```
//PAGEBREAK: 17
// Saved registers for kernel context switches.
// Don't need to save all the segment registers (%cs, etc),
// because they are constant across kernel contexts.
// Don't need to save %eax, %ecx, %edx, because the
// x86 convention is that the caller has saved them.
// Contexts are stored at the bottom of the stack they
// describe; the stack pointer is the address of the context.
// The layout of the context matches the layout of the stack in swtch.S
// at the "Switch stacks" comment. Switch doesn't save eip explicitly,
// but it is on the stack and allocproc() manipulates it.
struct context {
  uint edi;
  uint esi;
  uint ebx;
  uint ebp;
  uint eip;
};
```

Scheduler

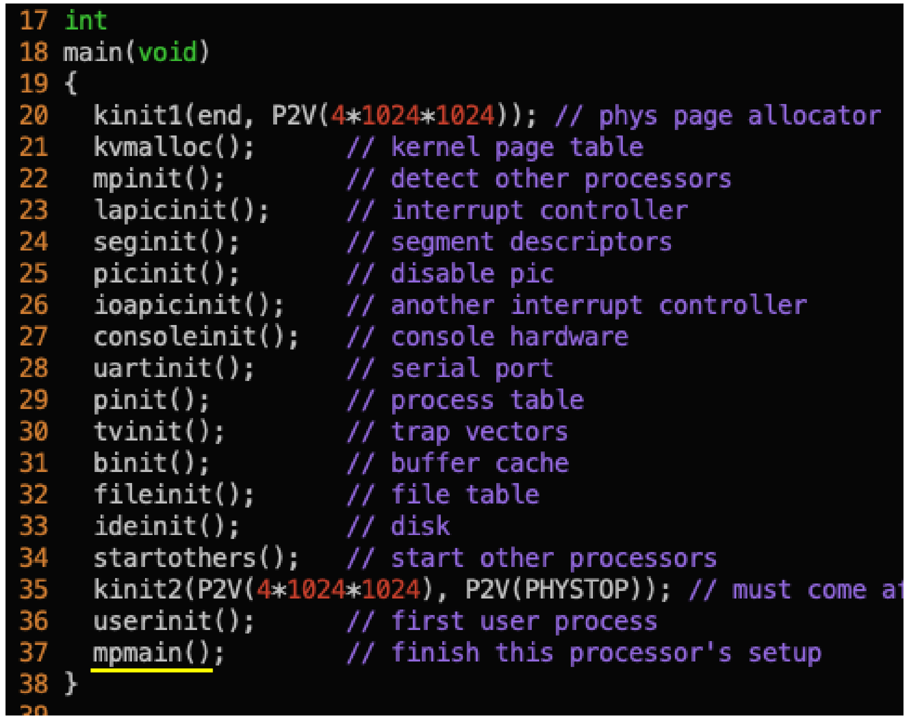
- proc.h
- struct cpu



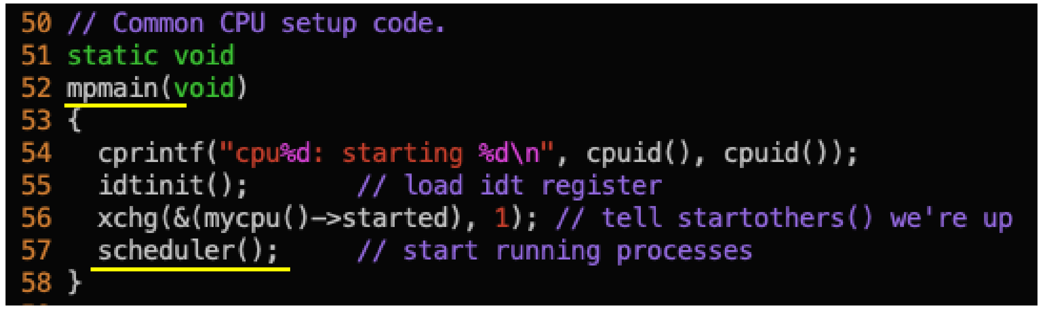
```
// Per-CPU state
struct cpu {
  uchar apicid;                // Local APIC ID
  struct context *scheduler;   // swtch() here to enter scheduler
  struct taskstate ts;         // Used by x86 to find stack for interrupt
  struct segdesc gdt[NSEGS];   // x86 global descriptor table
  volatile uint started;       // Has the CPU started?
  int ncli;                    // Depth of pushcli nesting.
  int intena;                  // Were interrupts enabled before pushcli?
  struct proc *proc;           // The process running on this cpu or null
};

extern struct cpu cpus[NCPU];
extern int ncpu;
```

Scheduler

main.c

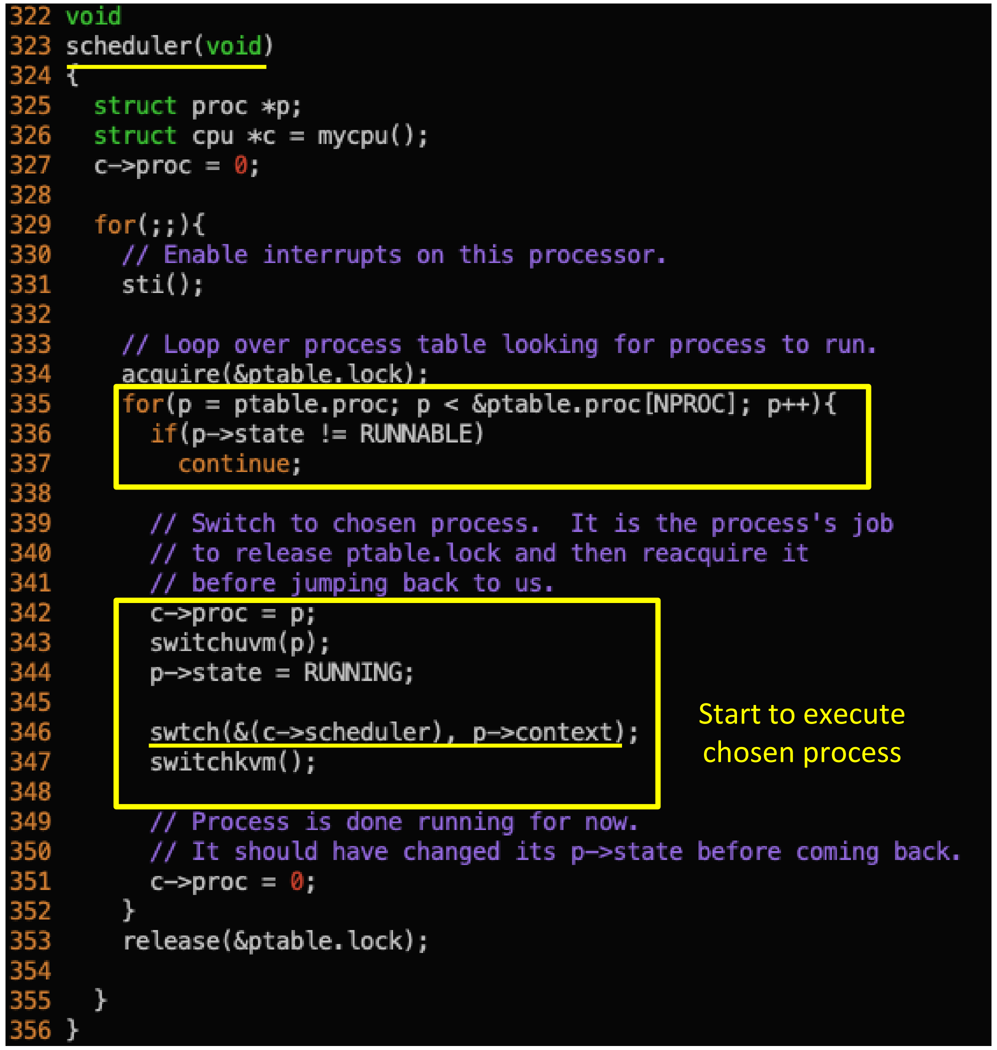


```
int
main(void)
{
  kinit1(end, P2V(4*1024*1024)); // phys page allocator
  kvmalloc();      // kernel page table
  mpinit();        // detect other processors
  lapicinit();     // interrupt controller
  seginit();       // segment descriptors
  picinit();       // disable pic
  ioapicinit();    // another interrupt controller
  consoleinit();   // console hardware
  uartinit();      // serial port
  pinit();         // process table
  tvinit();        // trap vectors
  binit();         // buffer cache
  fileinit();      // file table
  ideinit();       // disk
  startothers();   // start other processors
  kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must come af
  userinit();      // first user process
  mpmain();        // finish this processor's setup
}
```



```
// 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
}
```

proc.c



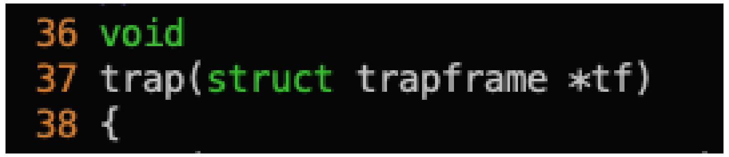
```
void
scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;

  for(;;){
    // Enable interrupts on this processor.
    sti();

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;

      // Switch to chosen process.  It is the process's job
      // to release ptable.lock and then reacquire it
      // before jumping back to us.
      c->proc = p;
      switchuvm(p);
      p->state = RUNNING;

      swtch(&(c->scheduler), p->context);
      switchkvm();

      // Process is done running for now.
      // It should have changed its p->state before coming back.
      c->proc = 0;
    }
    release(&ptable.lock);

  }
}
```

Start to execute chosen process

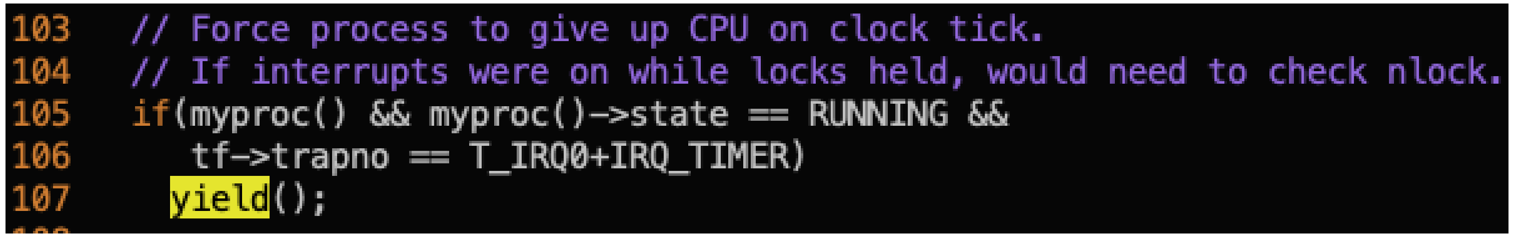
When to Schedule

- exit(), sleep()
- timer interrupt (yield())

trap.c

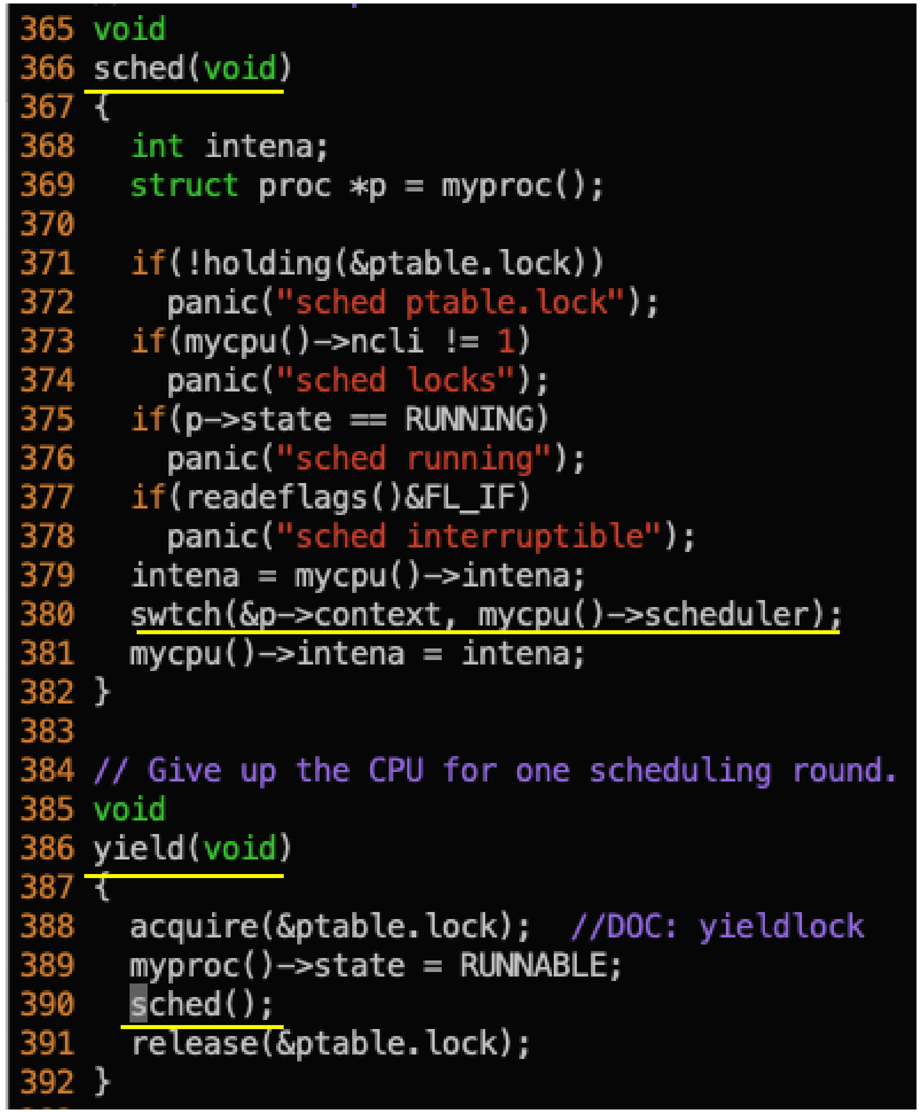


```
void
trap(struct trapframe *tf)
{
```

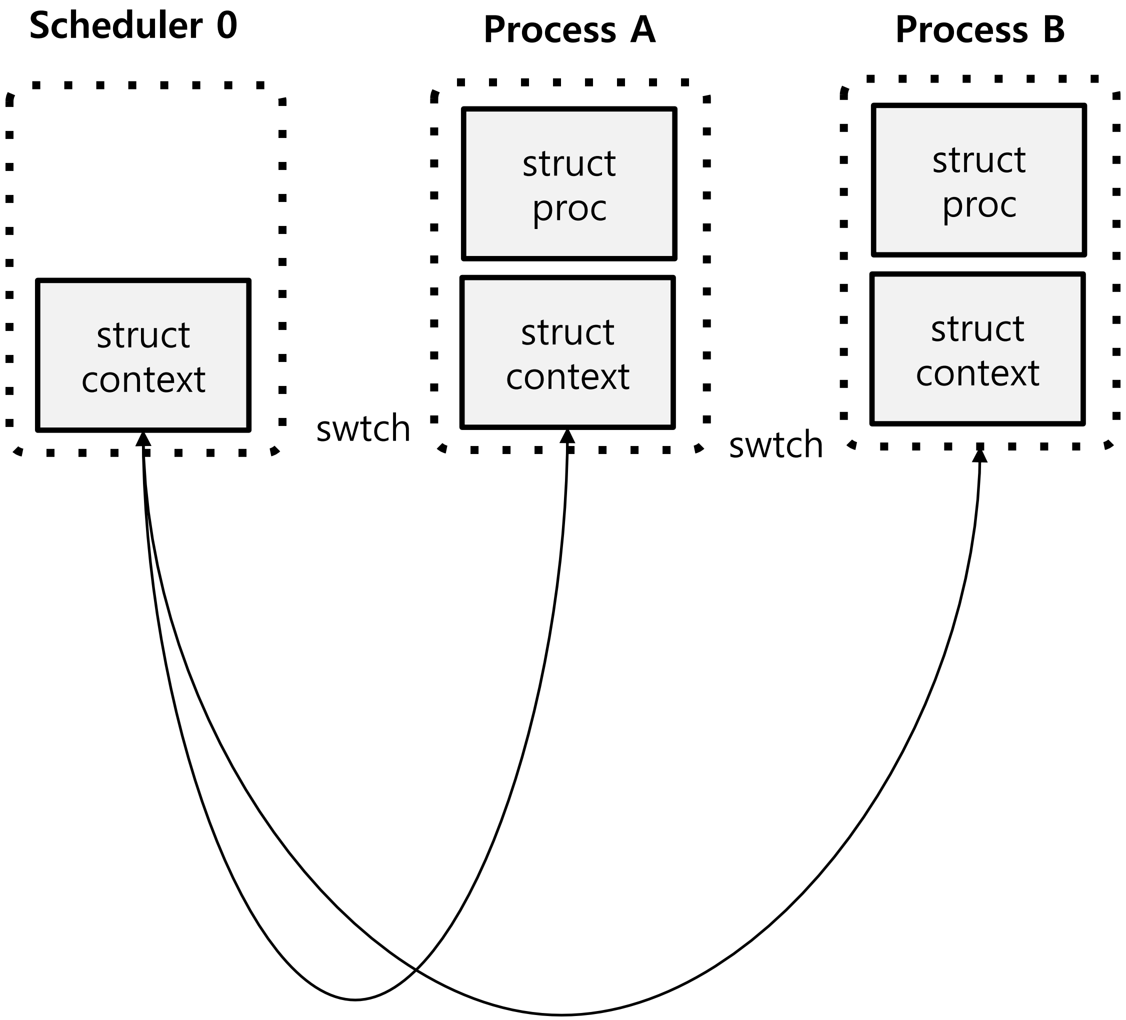


```
  // Force process to give up CPU on clock tick.
  // If interrupts were on while locks held, would need to check nlock.
  if(myproc() && myproc()->state == RUNNING &&
     tf->trapno == T_IRQ0+IRQ_TIMER)
    yield();
```

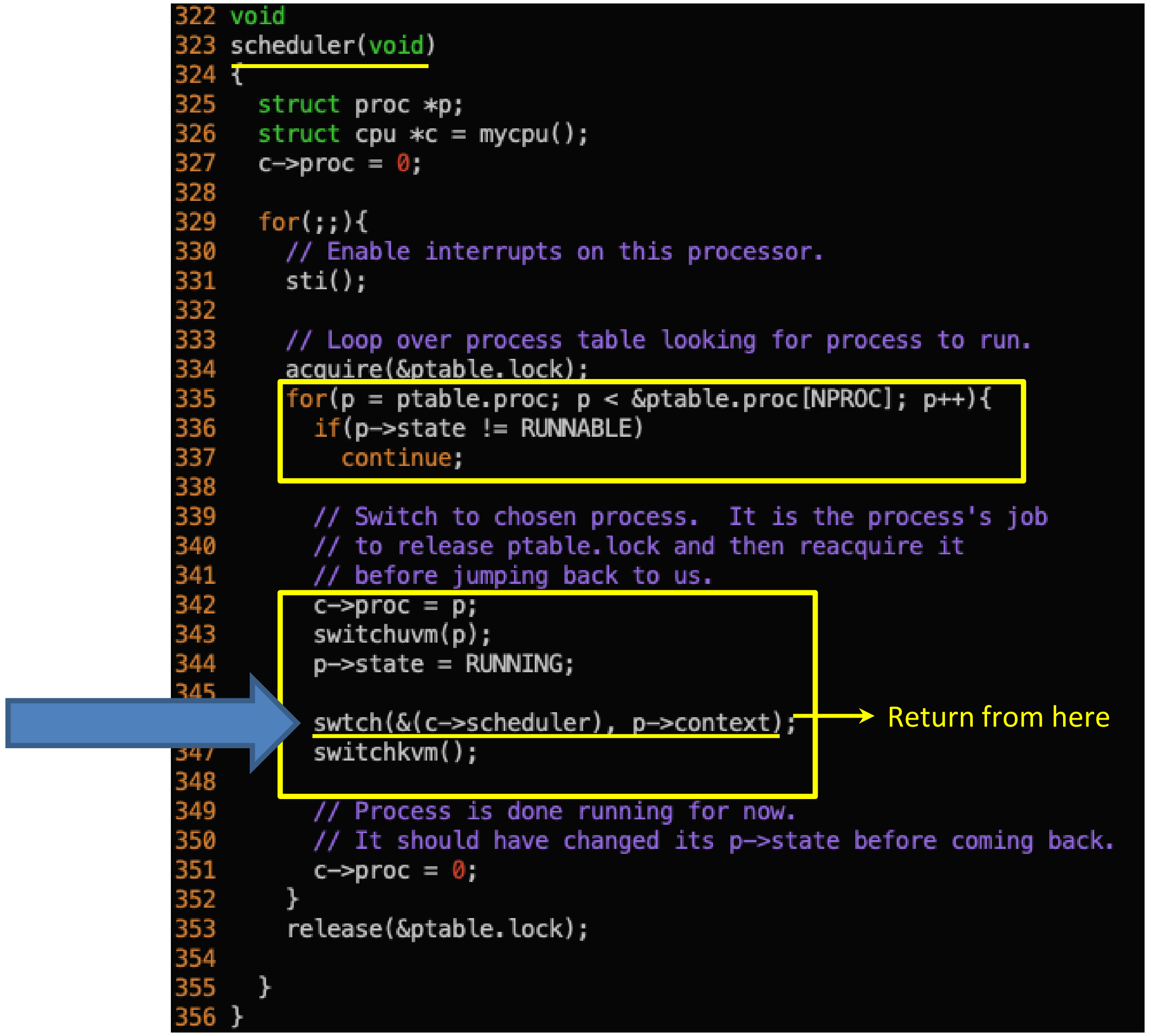
How Scheduler works



```
void
sched(void)
{
  int intena;
  struct proc *p = myproc();

  if(!holding(&ptable.lock))
    panic("sched ptable.lock");
  if(mycpu()->ncli != 1)
    panic("sched locks");
  if(p->state == RUNNING)
    panic("sched running");
  if(readeflags()&FL_IF)
    panic("sched interruptible");
  intena = mycpu()->intena;
  swtch(&p->context, mycpu()->scheduler);
  mycpu()->intena = intena;
}

// Give up the CPU for one scheduling round.
void
yield(void)
{
  acquire(&ptable.lock);   //DOC: yieldlock
  myproc()->state = RUNNABLE;
  sched();
  release(&ptable.lock);
}
```



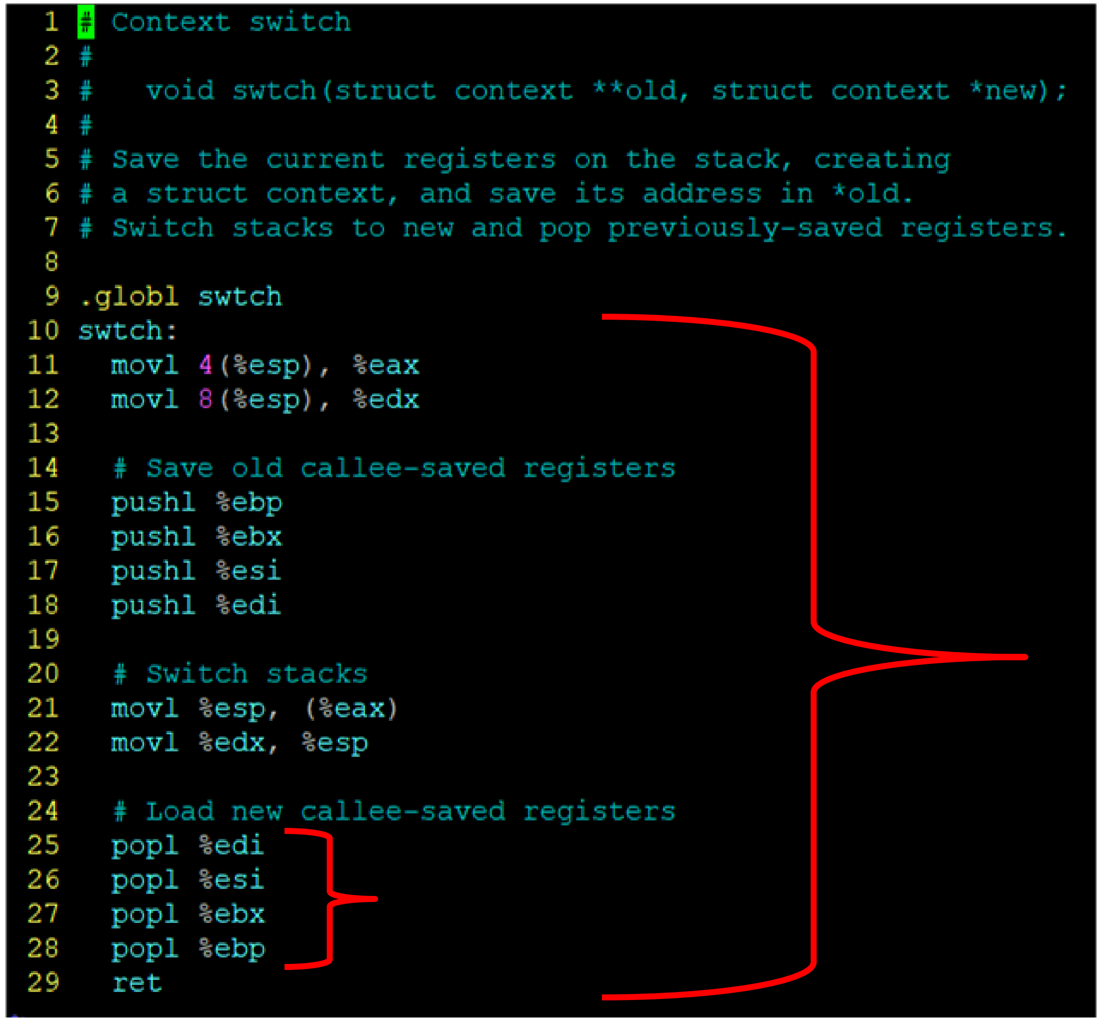
How Scheduler works



```
void
scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;

  for(;;){
    // Enable interrupts on this processor.
    sti();

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;

      // Switch to chosen process.  It is the process's job
      // to release ptable.lock and then reacquire it
      // before jumping back to us.
      c->proc = p;
      switchuvm(p);
      p->state = RUNNING;

      swtch(&(c->scheduler), p->context);
      switchkvm();

      // Process is done running for now.
      // It should have changed its p->state before coming back.
      c->proc = 0;
    }
    release(&ptable.lock);

  }
}
```

Context switch

- swtch.S



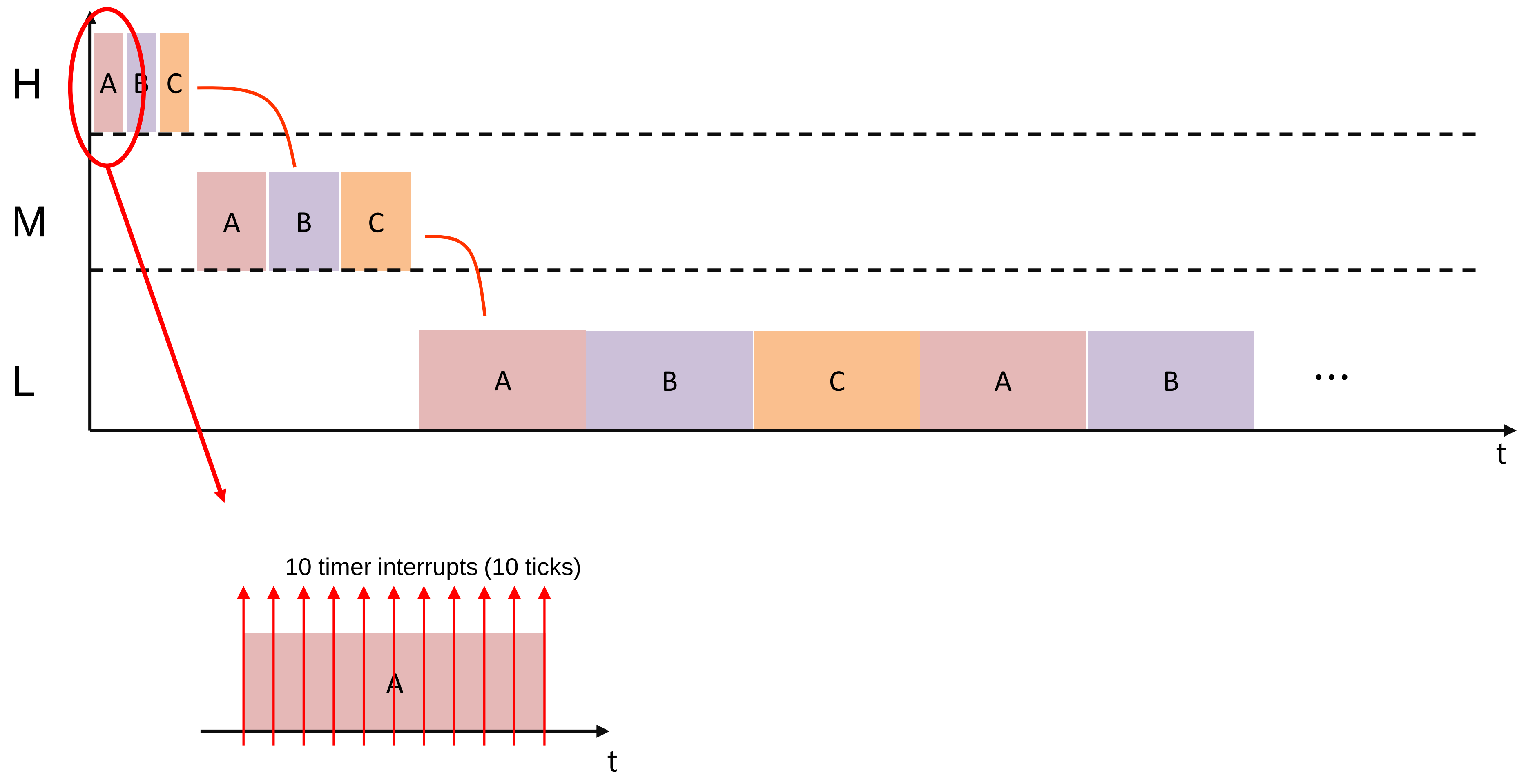
```
# Context switch
#
#   void swtch(struct context **old, struct context *new);
#
# Save the current registers on the stack, creating
# a struct context, and save its address in *old.
# Switch stacks to new and pop previously-saved registers.

.globl swtch
swtch:
  movl 4(%esp), %eax
  movl 8(%esp), %edx

  # Save old callee-saved registers
  pushl %ebp
  pushl %ebx
  pushl %esi
  pushl %edi

  # Switch stacks
  movl %esp, (%eax)
  movl %edx, %esp

  # Load new callee-saved registers
  popl %edi
  popl %esi
  popl %ebx
  popl %ebp
  ret
```

Project 2. Simple MLFQ Scheduling

- Implement a MLFQ scheduler that employs the following rules:
  - If priority (process A) > priority (process B), then run the process A.
  - If priority (process A) == priority (process B), then run the process A and B in the RR manner.
  - It has three queues with different priority levels (HIGH(H)/ MID(M)/ LOW(L)).
    - H, M, L queues have the time slices of 10, 20, and 30 timer ticks.
  - A newly arriving process initially belongs to the H queue.
  - A process which consumes all the time slice moves to the next priority queue.
    - e.g., H → M, M → L
  - If a process gives up the CPU before the time slice is up, it stays at the same priority queue.

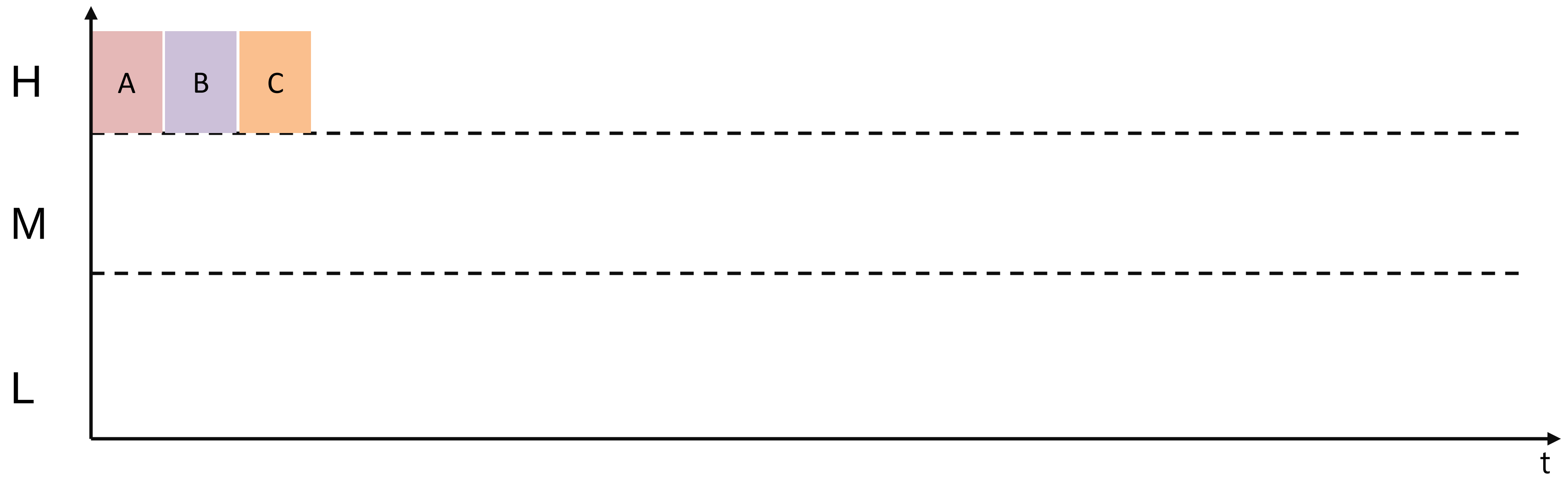
- Note
  - No priority boost.
  - No cumulative accounting.

Example



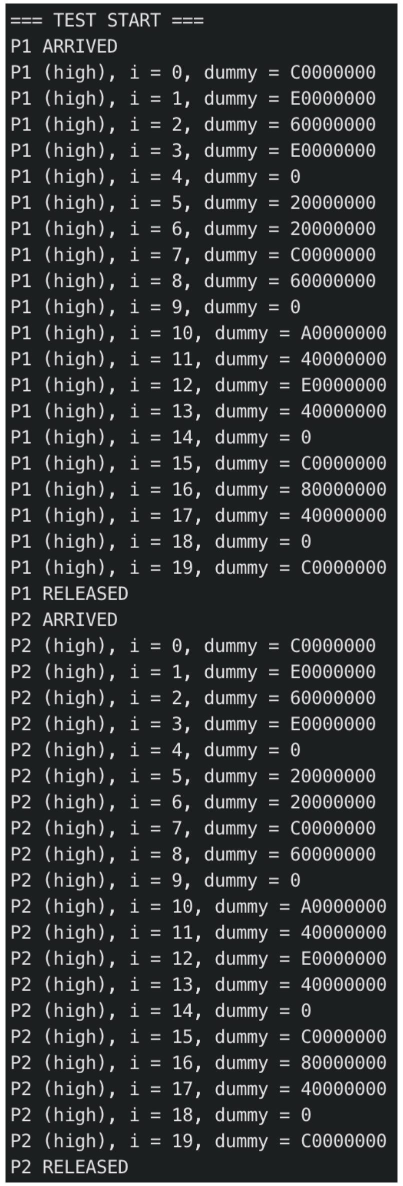
test_rr.c

- Add test_rr.c as user program
- ./test_rr
- Each process will consumes 6-7 time ticks (it will reside in the H queue)

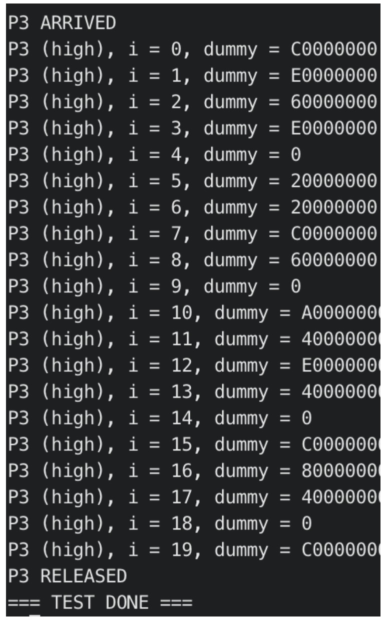


test_mlfq.c

- 출력은 실행환경의 tick 호출 주기에 따라 약간씩 다를 수 있음.
- Xv6의 기본 round-robin scheduler 가 아닌, time slice 를 모두 사용한 이후에 context switch 되도록 구현
  - HIGH 큐 기준 타임 슬라이스는 10 timer ticks.
  - 테스트 케이스의 P1, P2, P3는 약 5~7 ticks 를 사용하도록 구현됨.
  - 실행 시, P1 → P2 → P3 순서로 실행되어야 함.



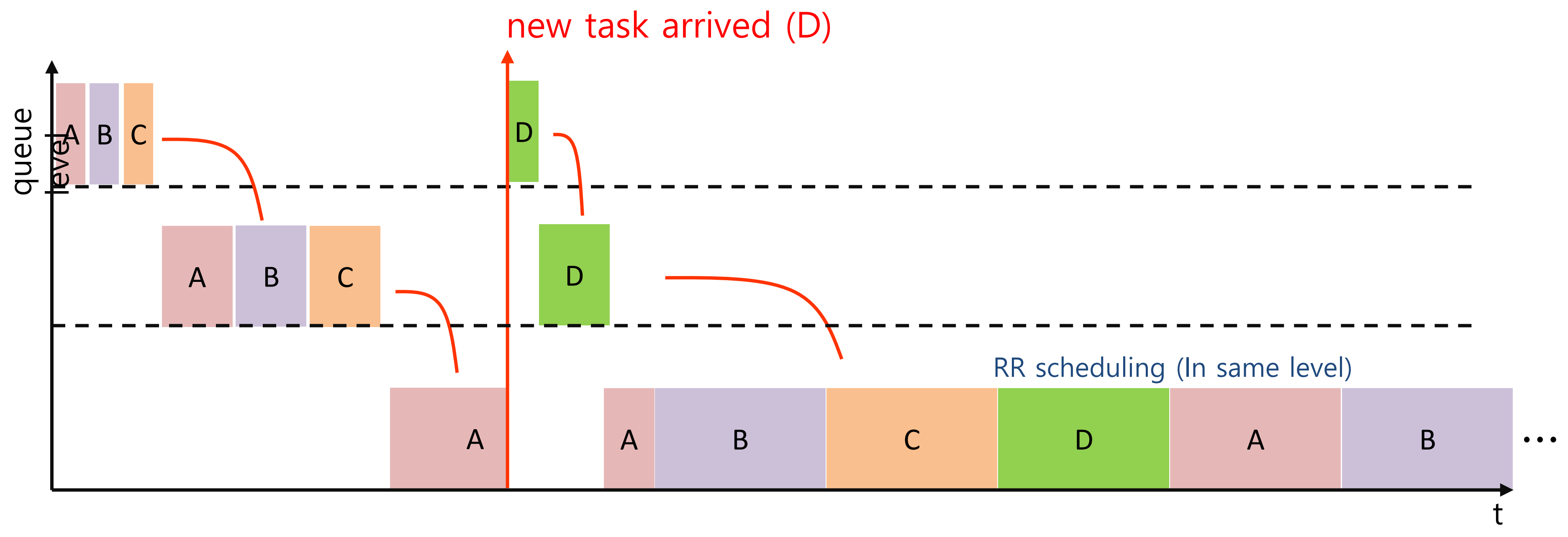
```
=== TEST START ===
P1 ARRIVED
P1 (high), i = 0, dummy = C0000000
P1 (high), i = 1, dummy = E0000000
P1 (high), i = 2, dummy = 60000000
P1 (high), i = 3, dummy = E0000000
P1 (high), i = 4, dummy = 0
P1 (high), i = 5, dummy = 20000000
P1 (high), i = 6, dummy = 20000000
P1 (high), i = 7, dummy = C0000000
P1 (high), i = 8, dummy = 60000000
P1 (high), i = 9, dummy = 0
P1 (high), i = 10, dummy = A0000000
P1 (high), i = 11, dummy = 40000000
P1 (high), i = 12, dummy = E0000000
P1 (high), i = 13, dummy = 40000000
P1 (high), i = 14, dummy = 0
P1 (high), i = 15, dummy = C0000000
P1 (high), i = 16, dummy = 80000000
P1 (high), i = 17, dummy = 40000000
P1 (high), i = 18, dummy = 0
P1 (high), i = 19, dummy = C0000000
P1 RELEASED
P2 ARRIVED
P2 (high), i = 0, dummy = C0000000
P2 (high), i = 1, dummy = E0000000
P2 (high), i = 2, dummy = 60000000
P2 (high), i = 3, dummy = E0000000
P2 (high), i = 4, dummy = 0
P2 (high), i = 5, dummy = 20000000
P2 (high), i = 6, dummy = 20000000
P2 (high), i = 7, dummy = C0000000
P2 (high), i = 8, dummy = 60000000
P2 (high), i = 9, dummy = 0
P2 (high), i = 10, dummy = A0000000
P2 (high), i = 11, dummy = 40000000
P2 (high), i = 12, dummy = E0000000
P2 (high), i = 13, dummy = 40000000
P2 (high), i = 14, dummy = 0
P2 (high), i = 15, dummy = C0000000
P2 (high), i = 16, dummy = 80000000
P2 (high), i = 17, dummy = 40000000
P2 (high), i = 18, dummy = 0
P2 (high), i = 19, dummy = C0000000
P2 RELEASED
```



```
P3 ARRIVED
P3 (high), i = 0, dummy = C0000000
P3 (high), i = 1, dummy = E0000000
P3 (high), i = 2, dummy = 60000000
P3 (high), i = 3, dummy = E0000000
P3 (high), i = 4, dummy = 0
P3 (high), i = 5, dummy = 20000000
P3 (high), i = 6, dummy = 20000000
P3 (high), i = 7, dummy = C0000000
P3 (high), i = 8, dummy = 60000000
P3 (high), i = 9, dummy = 0
P3 (high), i = 10, dummy = A000000
P3 (high), i = 11, dummy = 4000000
P3 (high), i = 12, dummy = E000000
P3 (high), i = 13, dummy = 4000000
P3 (high), i = 14, dummy = 0
P3 (high), i = 15, dummy = C000000
P3 (high), i = 16, dummy = 8000000
P3 (high), i = 17, dummy = 4000000
P3 (high), i = 18, dummy = 0
P3 (high), i = 19, dummy = C000000
P3 RELEASED
=== TEST DONE ===
```

test_mlfq.c

- Add test_mlfq.c as user program
- P1 output may be different
- ./test_mlfq



test_mlfq.c

- 출력은 실행환경의 tick 호출 주기에 따라 약간씩 다를 수 있음.
- 같은 우선순위 큐 내에서 RR 스케줄링 된다면 문제 없음.
  - 각 우선순위 큐에서 타임 슬라이스마다 프로세스들이 비슷한 반복 횟수를 보이면 문제 없음.
    (예: P1/P4: 3번, P2/P3: 4번 □ ok)
  - HIGH, MID 큐의 경우 타임 슬라이스가 작아 프로세스의 남은 작업이 대부분 LOW 큐에서 동작함.
  - LOW 큐의 타임 슬라이스가 크지만 남은 작업을 한 타임 슬라이스 내에 모두 실행하지는 못함.
    (P1, P2, P3, P4가 타임 슬라이스를 재할당 받고 RR scheduling 됨.)
  - 애매한 경우 TA에게 문의

P1, P2, P3 in HIGH queue → RR scheduling.
Each process exhausts its own time slice(10).

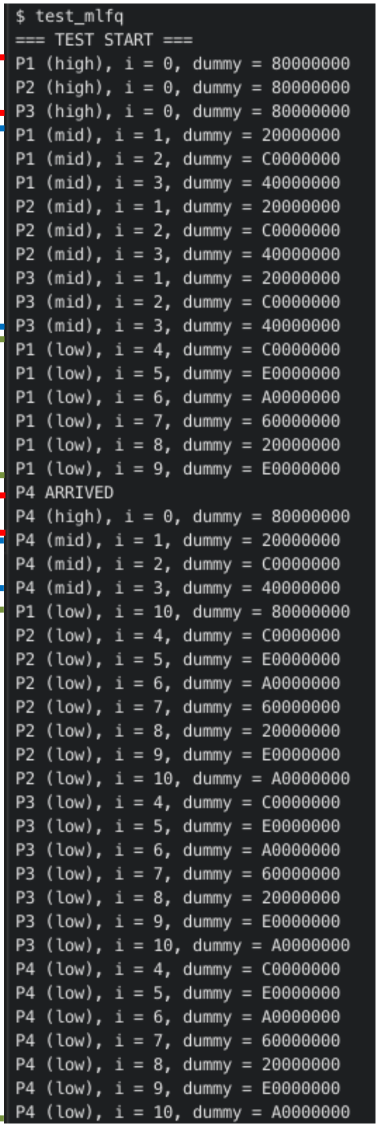
P1, P2, P3 in MID queue→ RR scheduling.
Each process exhausts its own time slice(20).

P1, P2, P3 in LOW queue → RR scheduling.

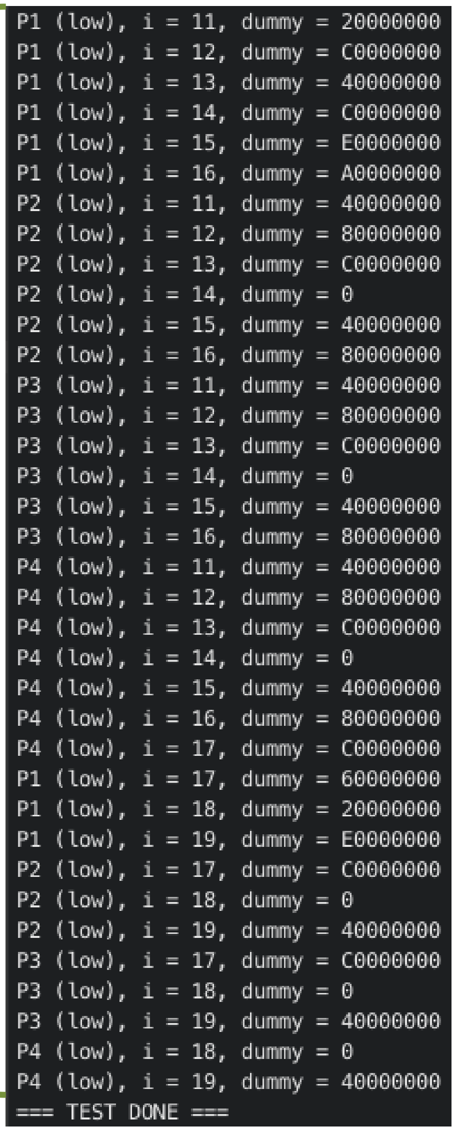
P4 arrived → P4 in HIGH queue

P4 in MID queue

P1, P2, P3, P4 in LOW queue → RR scheduling.
Each process exhausts its own time slice(30)
and reset time slice(30).



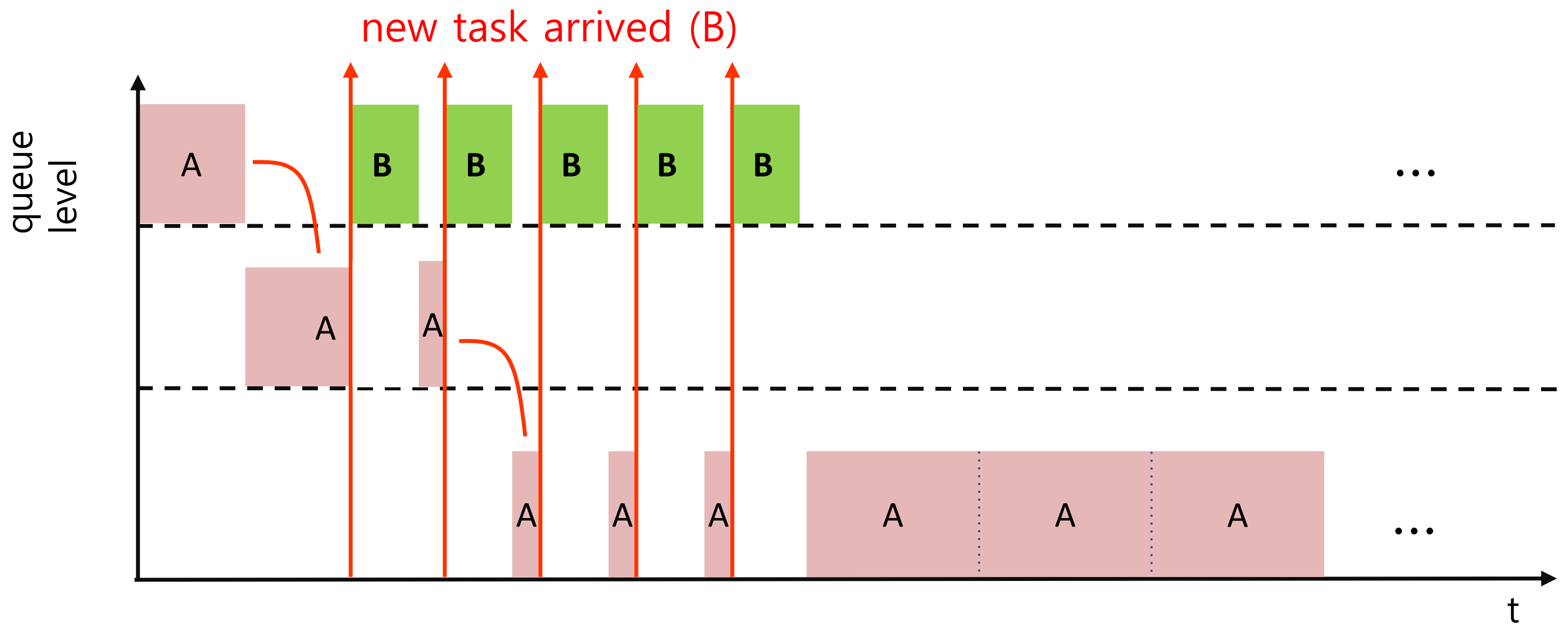
```
$ test_mlfq
=== TEST START ===
P1 (high), i = 0, dummy = 80000000
P2 (high), i = 0, dummy = 80000000
P3 (high), i = 0, dummy = 80000000
P1 (mid), i = 1, dummy = 20000000
P1 (mid), i = 2, dummy = C0000000
P1 (mid), i = 3, dummy = 40000000
P2 (mid), i = 1, dummy = 20000000
P2 (mid), i = 2, dummy = C0000000
P2 (mid), i = 3, dummy = 40000000
P3 (mid), i = 1, dummy = 20000000
P3 (mid), i = 2, dummy = C0000000
P3 (mid), i = 3, dummy = 40000000
P1 (low), i = 4, dummy = C0000000
P1 (low), i = 5, dummy = E0000000
P1 (low), i = 6, dummy = A0000000
P1 (low), i = 7, dummy = 60000000
P1 (low), i = 8, dummy = 20000000
P1 (low), i = 9, dummy = E0000000
P4 ARRIVED
P4 (high), i = 0, dummy = 80000000
P4 (mid), i = 1, dummy = 20000000
P4 (mid), i = 2, dummy = C0000000
P4 (mid), i = 3, dummy = 40000000
P1 (low), i = 10, dummy = 80000000
P2 (low), i = 4, dummy = C0000000
P2 (low), i = 5, dummy = E0000000
P2 (low), i = 6, dummy = A0000000
P2 (low), i = 7, dummy = 60000000
P2 (low), i = 8, dummy = 20000000
P2 (low), i = 9, dummy = E0000000
P2 (low), i = 10, dummy = A0000000
P3 (low), i = 4, dummy = C0000000
P3 (low), i = 5, dummy = E0000000
P3 (low), i = 6, dummy = A0000000
P3 (low), i = 7, dummy = 60000000
P3 (low), i = 8, dummy = 20000000
P3 (low), i = 9, dummy = E0000000
P3 (low), i = 10, dummy = A0000000
P4 (low), i = 4, dummy = C0000000
P4 (low), i = 5, dummy = E0000000
P4 (low), i = 6, dummy = A0000000
P4 (low), i = 7, dummy = 60000000
P4 (low), i = 8, dummy = 20000000
P4 (low), i = 9, dummy = E0000000
P4 (low), i = 10, dummy = A0000000
```



```
P1 (low), i = 11, dummy = 20000000
P1 (low), i = 12, dummy = C0000000
P1 (low), i = 13, dummy = 40000000
P1 (low), i = 14, dummy = C0000000
P1 (low), i = 15, dummy = E0000000
P1 (low), i = 16, dummy = A0000000
P2 (low), i = 11, dummy = 40000000
P2 (low), i = 12, dummy = 80000000
P2 (low), i = 13, dummy = C0000000
P2 (low), i = 14, dummy = 0
P2 (low), i = 15, dummy = 40000000
P2 (low), i = 16, dummy = 80000000
P3 (low), i = 11, dummy = 40000000
P3 (low), i = 12, dummy = 80000000
P3 (low), i = 13, dummy = C0000000
P3 (low), i = 14, dummy = 0
P3 (low), i = 15, dummy = 40000000
P3 (low), i = 16, dummy = 80000000
P4 (low), i = 11, dummy = 40000000
P4 (low), i = 12, dummy = 80000000
P4 (low), i = 13, dummy = C0000000
P4 (low), i = 14, dummy = 0
P4 (low), i = 15, dummy = 40000000
P4 (low), i = 16, dummy = 80000000
P4 (low), i = 17, dummy = C0000000
P1 (low), i = 17, dummy = 60000000
P1 (low), i = 18, dummy = 20000000
P1 (low), i = 19, dummy = E0000000
P2 (low), i = 17, dummy = C0000000
P2 (low), i = 18, dummy = 0
P2 (low), i = 19, dummy = 40000000
P3 (low), i = 17, dummy = C0000000
P3 (low), i = 18, dummy = 0
P3 (low), i = 19, dummy = 40000000
P4 (low), i = 18, dummy = 0
P4 (low), i = 19, dummy = 40000000
=== TEST DONE ===
```

test_mlfq2.c

- Add test_mlfq.c as user program
- P1 output may be different
- ./test_mlfq2



test_mlfq2.c

- 출력은 실행환경의 tick 호출 주기에 따라 다를 수 있다.
- HIGH, MID, LOW time slice 는 각각 10, 20, 30으로 설정.
- P1 process 실행 중, P2 process가 주기적으로 생성되어 HIGH time slice 내로 작업 완료 후 종료되어, P1의 starvation 현상이 발생하는 양상이라면 문제 없음.
  - P2 사이에 P1의 실행 횟수는 tick 호출에 따라 달라질 수 있음.
  - Priority Boosting의 필요성을 보여주는 testcase

P1 in HIGH queue.
P1 consumes its own time slice(10).

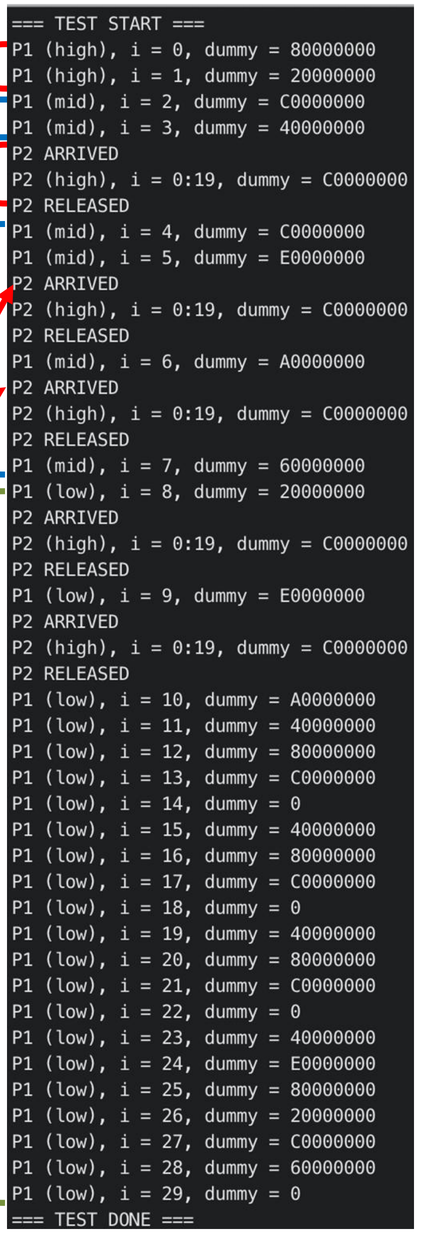
P1 in MID queue.

P2 arrives, it runs in HIGH queue.
P2 is done before it consumes the time slice of HIGH queue.

P1 runs in MID queue
until it consumes all the time slices (20ticks)

P2 arrives repeatedly, it runs in HIGH queue.

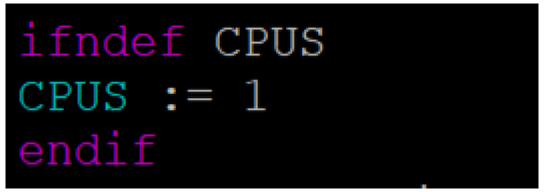
P1 runs in LOW queue



```
=== TEST START ===
P1 (high), i = 0, dummy = 80000000
P1 (high), i = 1, dummy = 20000000
P1 (mid), i = 2, dummy = C0000000
P1 (mid), i = 3, dummy = 40000000
P2 ARRIVED
P2 (high), i = 0:19, dummy = C0000000
P2 RELEASED
P1 (mid), i = 4, dummy = C0000000
P1 (mid), i = 5, dummy = E0000000
P2 ARRIVED
P2 (high), i = 0:19, dummy = C0000000
P2 RELEASED
P1 (mid), i = 6, dummy = A0000000
P2 ARRIVED
P2 (high), i = 0:19, dummy = C0000000
P2 RELEASED
P1 (mid), i = 7, dummy = 60000000
P1 (low), i = 8, dummy = 20000000
P2 ARRIVED
P2 (high), i = 0:19, dummy = C0000000
P2 RELEASED
P1 (low), i = 9, dummy = E0000000
P2 ARRIVED
P2 (high), i = 0:19, dummy = C0000000
P2 RELEASED
P1 (low), i = 10, dummy = A0000000
P1 (low), i = 11, dummy = 40000000
P1 (low), i = 12, dummy = 80000000
P1 (low), i = 13, dummy = C0000000
P1 (low), i = 14, dummy = 0
P1 (low), i = 15, dummy = 40000000
P1 (low), i = 16, dummy = 80000000
P1 (low), i = 17, dummy = C0000000
P1 (low), i = 18, dummy = 0
P1 (low), i = 19, dummy = 40000000
P1 (low), i = 20, dummy = 80000000
P1 (low), i = 21, dummy = C0000000
P1 (low), i = 22, dummy = 0
P1 (low), i = 23, dummy = 40000000
P1 (low), i = 24, dummy = E0000000
P1 (low), i = 25, dummy = 80000000
P1 (low), i = 26, dummy = 20000000
P1 (low), i = 27, dummy = C0000000
P1 (low), i = 28, dummy = 60000000
P1 (low), i = 29, dummy = 0
=== TEST DONE ===
```

Hand-in Procedures (1/2)

- Download template
  - https://github.com/KilhoLee/xv6-ssu.git (pull or clone)
  - `tar xvzf xv6_ssu_mlfq.tar.gz`
- Add test_*.c to your codes and modify Makefile properly
  - `test_rr.c, test_mlfq.c, test_mlfq2.c`
- Build with CPUS=1 flag
  - Makefile



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

Hand-in Procedures (2/2)

- Compress your code (ID: 20221234)
  - `$tar cvzf xv6_ssu_mlfq_20221234.tar.gz xv6_ssu_mlfq`
  - Please command `$make clean` before compressing

- Submit your `tar.gz` file through class.ssu.ac.kr