Lab3: Process Management

Yihua Xu

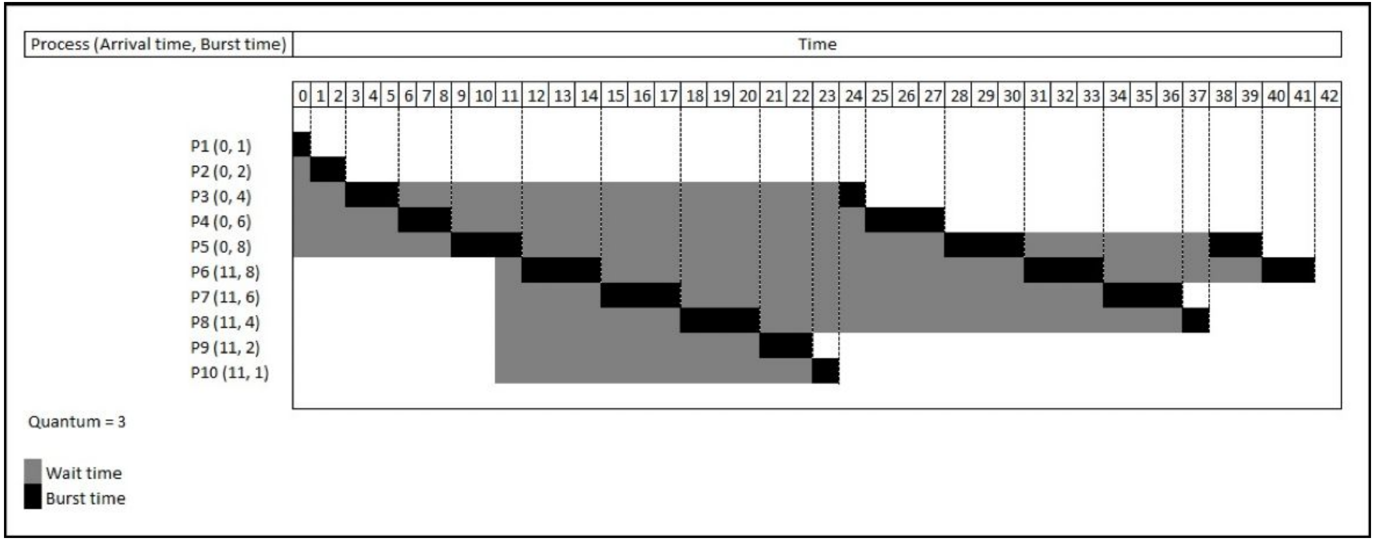
Overview

- Process scheduling algorithm:
  - Round-robin scheduling v.s. Priority scheduling
- Hints for lab 3
- Xv6 book link: https://pdos.csail.mit.edu/6.828/2018/xv6/book-rev10.pdf

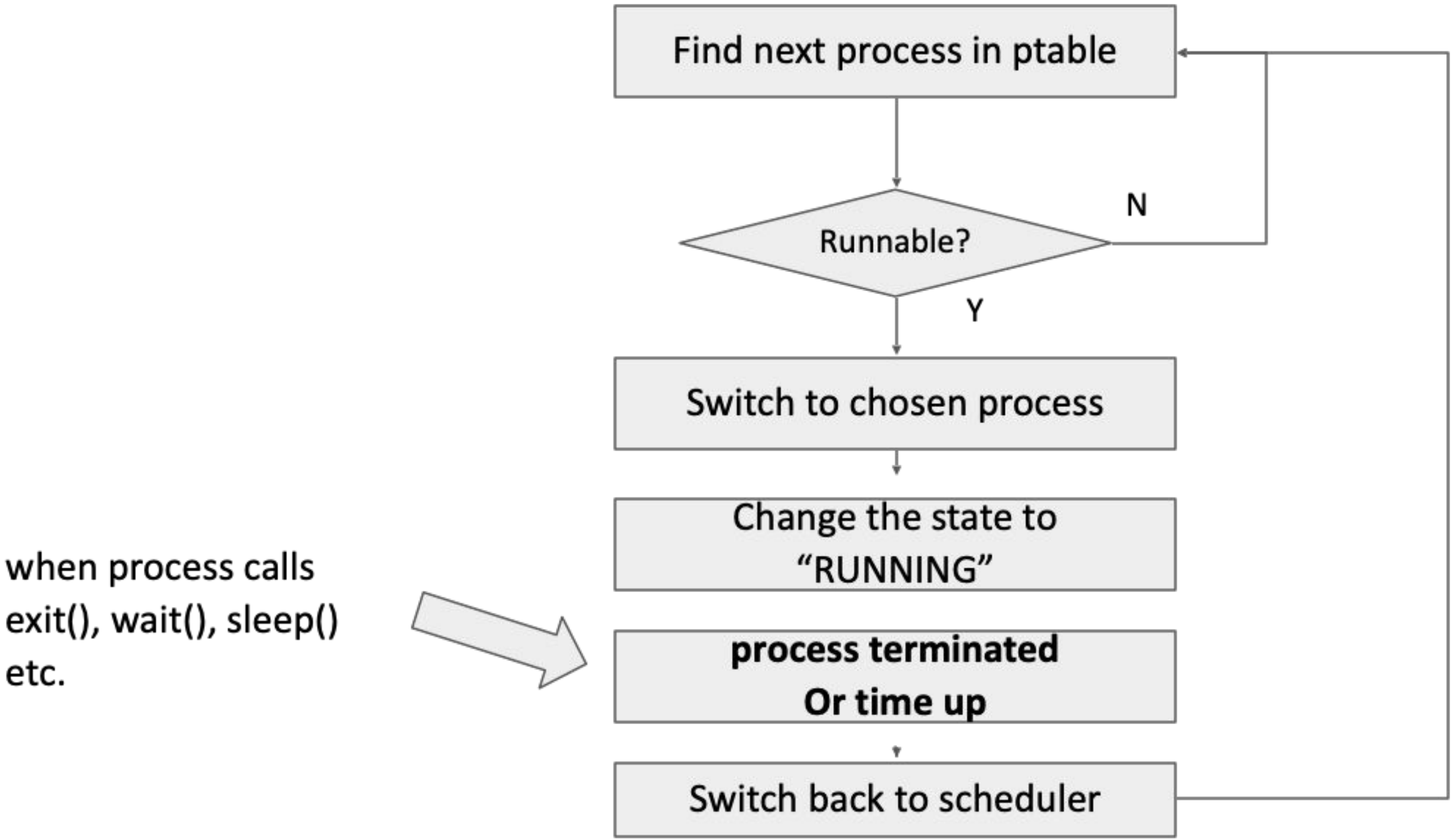
Quick review

- Scheduler
  - OS module that manipulates the job queues, moving jobs to and from;
- Scheduling algorithm:
  - Determines which jobs are chosen to run next and what queues they wait on;
- Scheduler runs
  - When a job switches from running to waiting;
  - When an interrupt occurs;
  - When a job is created or terminated

scheduling algorithm

- Round Robin (RR)
  - Each task gets resource for a fixed period of time (time quantum). Unfinished task goes back in line
  - 

RR scheduler (current implementation)

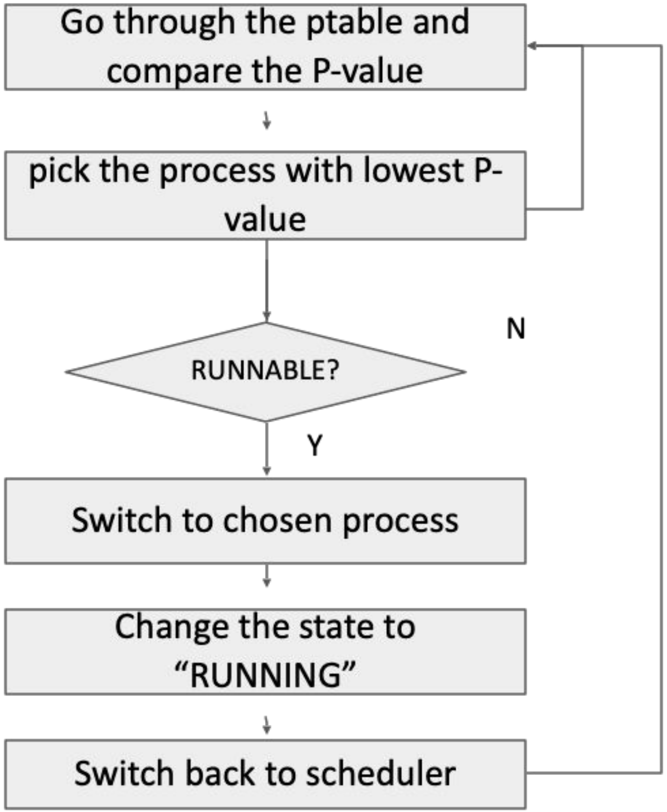


Priority scheduler (your task)

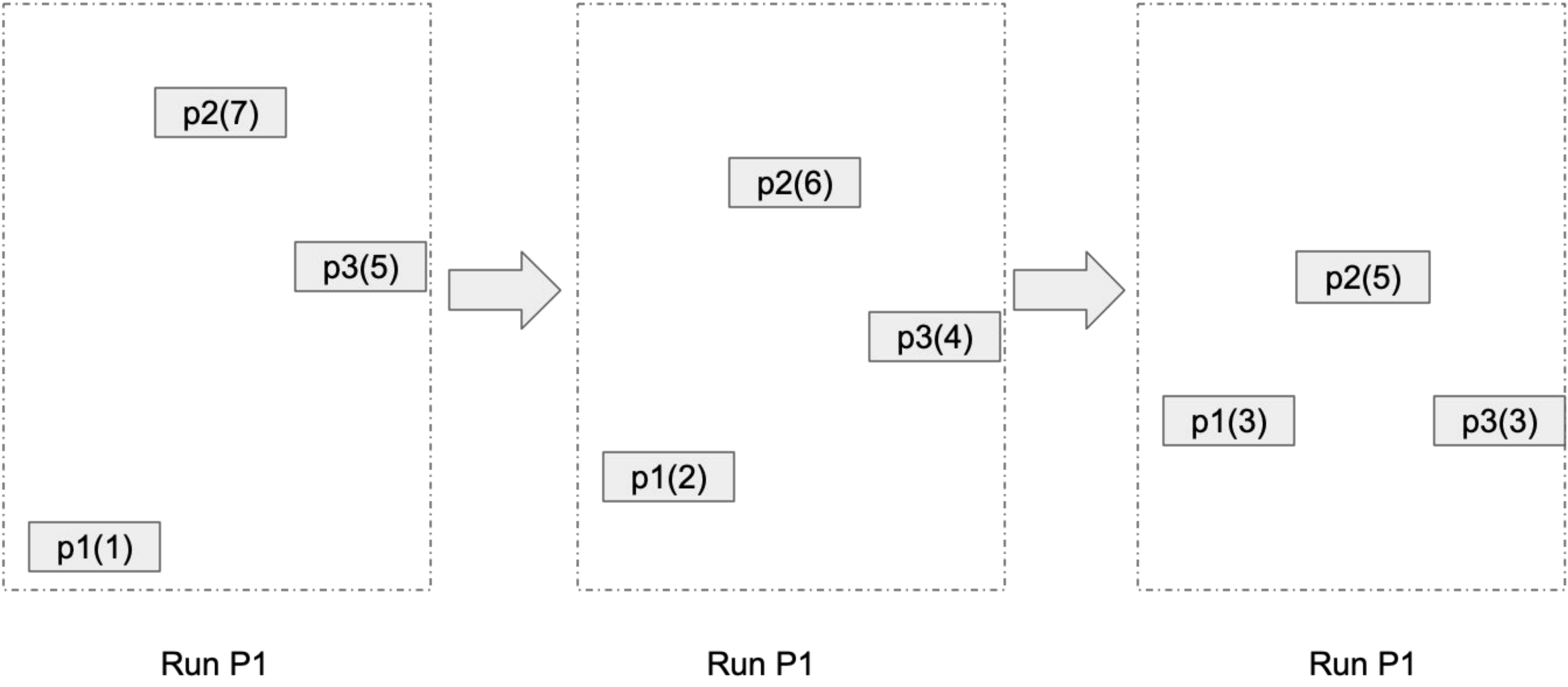
P-value range: [0, 31]

- 0: the highest priority;
- 31: the lowest priority;

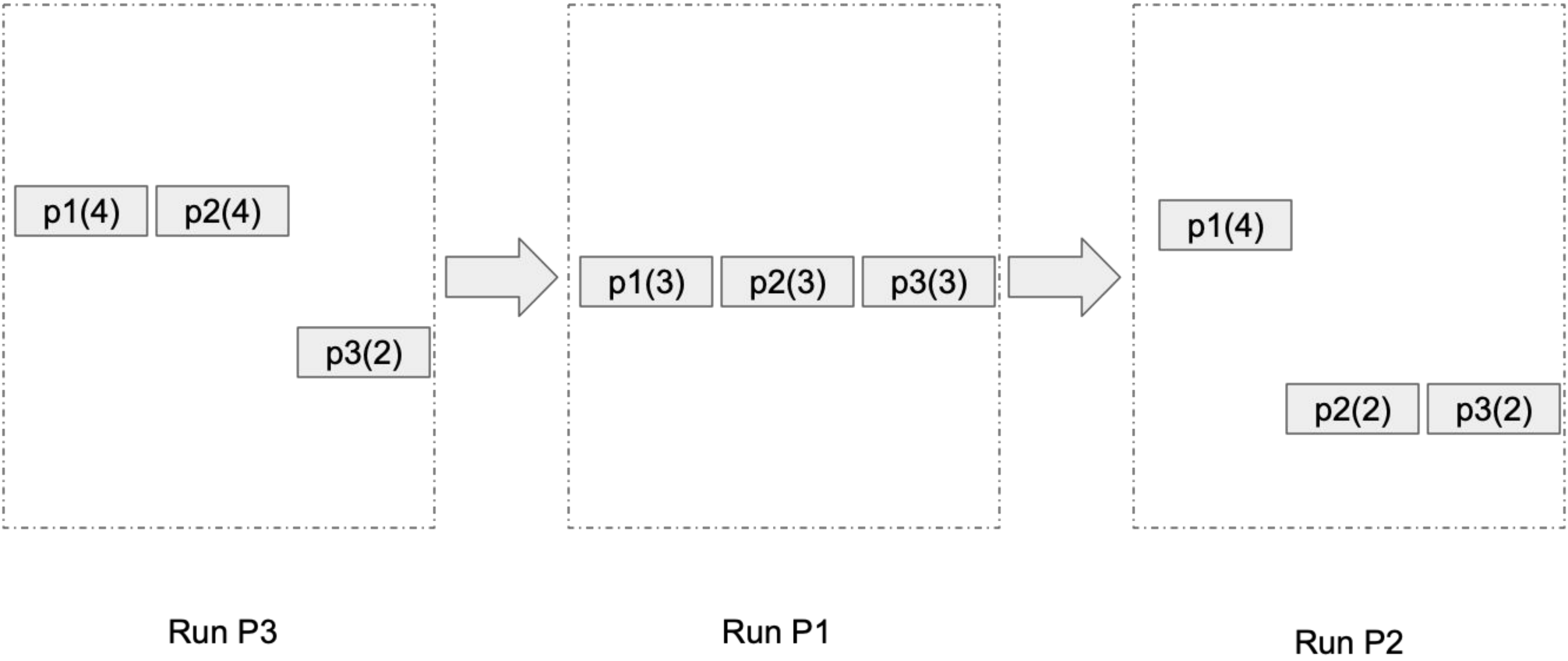
Implement aging of priority to avoid starvation.



How to age processes



How to age processes



Hints for lab3

1. Maintain a new field in proc structure to hold priority value;
2. add a new system call to modify the value anytime;
3. Modify round robin scheduler to priority scheduler;
4. Implement aging of priority to avoid starvation;
5. Compute the turnaround time and wait time for each process;
6. Develop a test program to illustrate priority scheduling.

Step 1: add new field to proc structure

- Add another field in proc struct:
  - E.g. int prior_val;
  - Value range: [0, 31]; 0: highest; 31: lowest
- Initialization of prior_val:
  - allocproc()
  - fork()
    - child should inherit parent's priority value
- Updating prior_val:
  - New system call;
  - Aging of priority;
  - Hint: to give up CPU resources when current proc's priority is lower than other procs:

proc.h > proc

```c
// 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)
};
```

Step 2: add system call to update priority value

- Follow the same steps in Lab2;
- void update_prior(int prior_lvl) OR int update_prior(int prior_lvl)
    - change the priority value of the current proc;
    - Then transfer control to scheduler immediately (call sched()) because the priority list has been changed;
        - Pay attention to :
            - the lock, or directly call yield();
            - sched() is not scheduler()

Step 3: modify scheduler()

Round robin

- Loop over process table
    - Only choose the "RUNNABLE" process, which is ready to execute;
    - Next RUNNABLE process p acquires CPU resource and start RUNNING;

Priority scheduling:

- *Loop over process table*
    - *find RUNNABLE proc with the highest priority;*
- *The chosen RUNNABLE process acquires CPU resources and start RUNNING;*

proc.c > scheduler(void)

```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);

  }
}
```

Step 4: aging of priority

- If a process waits increase its priority (decrease its value). When it runs, decrease it (increase its value).
- Modify scheduler() function to update procs' priority value;
- Pay attention to the legal range of priority value: [0, 31]

Step 5: turnaround & waiting time

- Turnaround time = T_finish - T_start
  - Add fields inside proc struct for tracking scheduling performance of each proc
  - Acquire time ticket from exec() => proc start time
  - Acquire time ticket from exit() => proc finish time
- Waiting time = Turnaround time - burst time
  - burst time: how many times it’s been scheduled, each time is a time ticket;

```
switch(tf->trapno){
case T_IRQ0 + IRQ_TIMER:
  if(cpuid() == 0){
    acquire(&tickslock);
    ticks++;
    wakeup(&ticks);
    release(&tickslock);
  }
```

Step 6: test your scheduler

- Test programs should be able to run for a while;
- To run multiple programs simultaneously:
  - $ proc1&;proc2&;proc3
  - Try different order ($ proc3&;proc1&;proc2)
- Compare the turnaround time and waiting time of processes with different priority;
- Illustrate priority scheduling with the result;

```
int main(int argc, char *argv[]) {
    if (argc >= 1){
      //int new = atoi(argv[1]);
      update_prior_val(1);
      int limit = 4300;
      int i, j;
      for (i=0;i<limit;i++) {
        asm("nop");
        for (j=0;j<limit;j++) {
            asm("nop");
        }
      }
    }
    exit();
}
```