xv6©-rev10
(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)
Scheduler

Carmi Merimovich

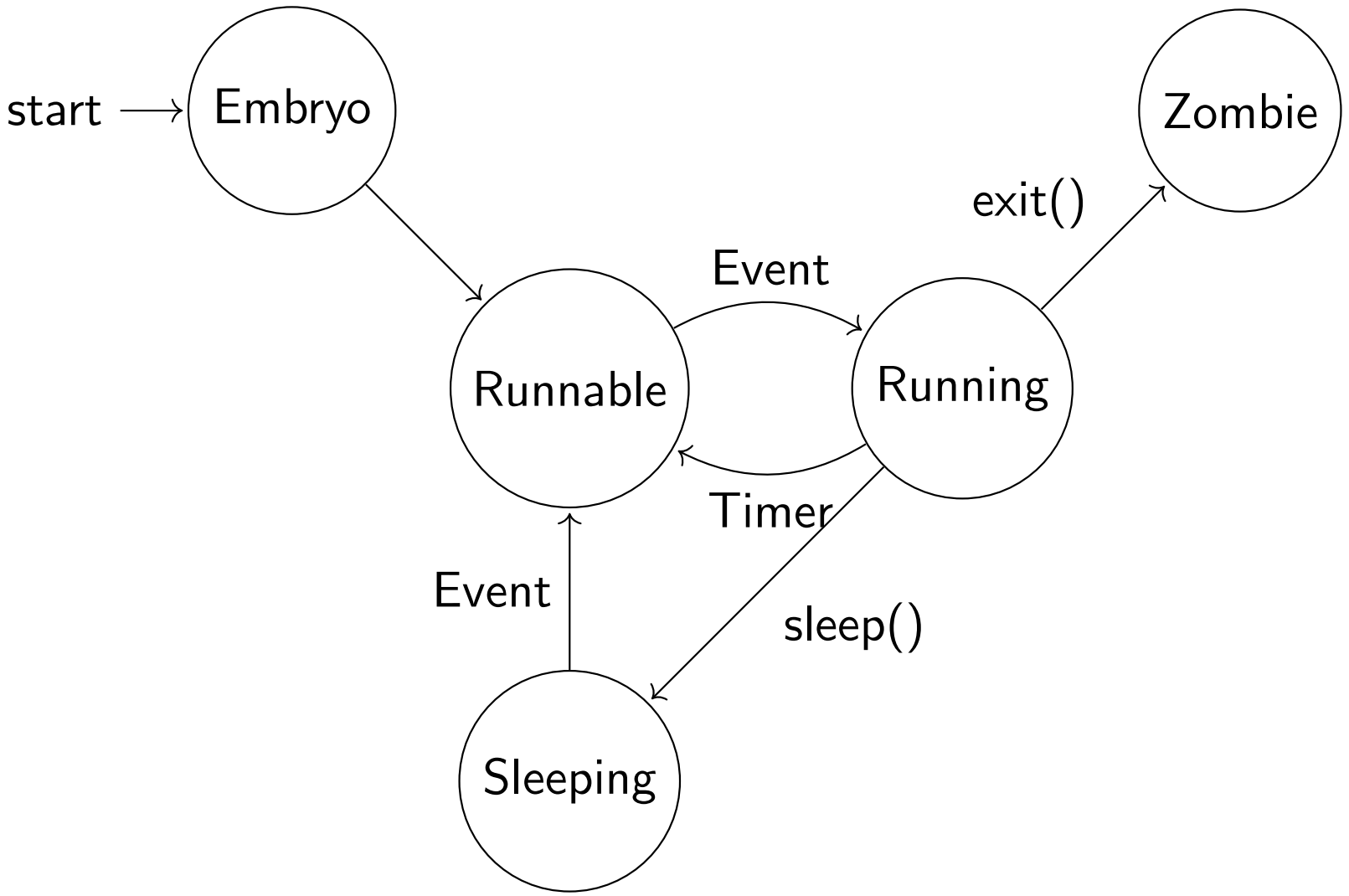
Tel-Aviv Academic College

November 20, 2017

struct proc

```
2337 struct proc {
     uint sz; // Size of process memory (bytes)
     pde_t* pgdir; // Page table
     char *kstack; // Bottom of kernel stack for this pr
     enum procstate state; // Process state
     volatile int pid; // Process ID
     struct proc *parent; // Parent process
     struct trapframe *tf; // Trap frame for current sys
     struct context *context; // swtch() here to run pro
     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)
    };
```

state transition



All transition are due to INTERRUPTS.

proc structure storage

```
2334 enum procstate { UNUSED, EMBRYO, SLEEPING,
                      RUNNABLE, RUNNING, ZOMBIE }

2409 struct {
      struct spinlock lock;
      struct proc proc[NPROC];
     } ptable;
```

Context

```
1219    kinit1(end, P2V(4*1024*1024)); // phys page alloca
        kvmalloc();          // kernel page table
           ⋮
1222    seginit();          // set up segments
           ⋮
1224    pinit();             // process table
           ⋮
           ⋮
1226
1227    kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must co
        userinit();          // first user process
        mpmain();
```

mpmain()

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

scheduler

```
2758 void scheduler(void) {
     struct proc *p;
     struct cpu *c = mycpu();
     c->proc = 0;
     for(;;) { sti();
       acquire(&ptable.lock);
       for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
         if (p->state != RUNNABLE) continue;

         c->proc = p;
         switchuvm(p);
         p->state = RUNNING;
         swtch(&c->scheduler, p->context);
         switchkvm();

         c->proc = 0;
       }
       release(&ptable.lock);
     }
```

scheduler() operation

- For each proc struct p with state RUNNABLE the following is executed:
  - **c−>proc = p**;
  - **switchuvm()**.
  - **swtch()**.
  - **switchkvm()**.
  - **c−>proc=NULL**.

Auxiliary context

- The primary processor begins its C code in **main()**.
- The auxiliary processors begins their C code in **mpenter()**.
- The state on entering either **main()** or **mpenter()** is the same.
- There is a separate stack of each processor.

```
static void mpenter(void) {
  switchkvm();
  seginit();
  lapicinit();
  mpmain();
}
```

mycpu()

```
2436 struct cpu* mycpu(void) {
      int apicid, i;

      if (readeflags()&FL_IF)
        panic("mycpu called with interrupts enabled\n");

      apicid = lapicid();
      for (i = 0; i < ncpu; ++i) {
        if (cpus[i].apicid == apicid)
          return &cpus[i];
      }
      panic("unknown apicid\n");
    }
```