Andrews Figures, Chapter 01

Geoffrey Matthews
Western Washington University

January 21, 2013

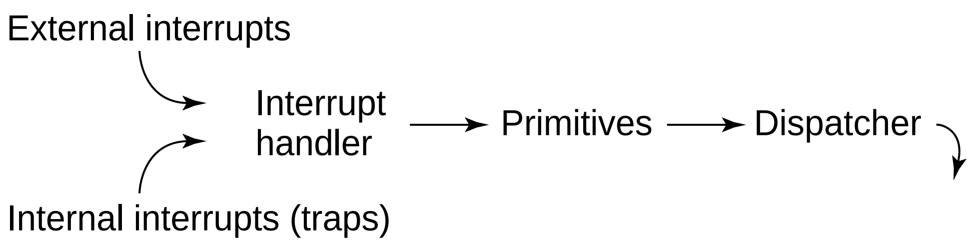

Figure 6.1 Kernel components and flow of control.

```
processType processDescriptor[maxProcs];
int executing = 0;    # index of the executing process
declarations of variables for the free, ready, and waiting lists;
SVC_Handler: {   # entered with interrupts inhibited
  save state of executing;
  determine which primitive was invoked, then call it;
}
Timer_Handler: {   # entered with interrupts inhibited
  insert descriptor of executing at end of ready list;
  executing = 0;
  dispatcher();
}
procedure fork(initial process state) {
  remove a descriptor from the free list and initialize it;
  insert the descriptor on the end of the ready list;
  dispatcher();
}
procedure quit() {
  record that executing has quit;
  insert descriptor of executing at end of free list;
  executing = 0;
  if (parent process is waiting for this child) {
    remove parent from the waiting list; put parent on the ready list; }
  dispatcher();
}
procedure join(name of child process) {
  if (child has not yet quit) {
    put the descriptor of executing on the waiting list;
    executing = 0;
  }
  dispatcher();
}
procedure dispatcher() {
  if (executing == 0) { # current process blocked or quit
    remove descriptor from front of ready list;
    set executing to point to it;
  }
  start the interval timer;
  load state of executing;    # with interrupts enabled
}
```

Figure 6.2 Outline of a single-processor kernel.

```
process Idle {
  while (executing[i] == the Idle process) {
    while (ready list empty) Delay;
    lock ready list;
    if (ready list not empty) {
      remove descriptor from front of ready list;
      set executing[i] to point to it;
    }
    unlock ready list;
  }
  start the interval timer on processor i;
  load state of executing[i];   # with interrupts enabled
}
```

Figure 6.3 Code for the idle process.

```
processType processDescriptor[maxProcs];
int executing[maxProcs];   # one entry per processor
declarations of free, ready, and waiting lists and their locks;

SVC_Handler: {
  # entered with interrupts inhibited on processor i
  save state of executing[i];
  determine which primitive was invoked, then call it;
}

Timer_Handler: {
  # entered with interrupts inhibited on processor i
  lock ready list; insert executing[i] at end; unlock ready list;
  executing[i] = 0;
  dispatcher();
}

procedure fork(initial process state) {
  lock free list; remove a descriptor; unlock free list;
  initialize the descriptor;
  lock ready list; insert descriptor at end; unlock ready list;
  dispatcher();
}

procedure quit() {
  lock free list; insert executing[i] at end; unlock free list;
  record that executing[i] has quit; executing[i] = 0;
  if (parent process is waiting) {
    lock waiting list; remove parent from that list; unlock waiting list;
    lock ready list; put parent on ready list; unlock ready list;
  }
  dispatcher();
}

procedure join(name of child process) {
  if (child has already quit)
    return;
  lock waiting list; put executing[i] on that list; unlock waiting list;
  dispatcher();
}
```

```
procedure createSem(initial value, int *name) {
  get an empty semaphore descriptor;
  initialize the descriptor;
  set name to the name (index) of the descriptor;
  dispatcher();
}
procedure Psem(name) {
  find semaphore descriptor of name;
  if (value > 0)
    value = value - 1;
  else {
    insert descriptor of executing at end of blocked list;
    executing = 0;    # indicate executing is blocked
  }
  dispatcher();
}
procedure Vsem(name) {
  find semaphore descriptor of name;
  if (blocked list empty)
    value = value + 1;
  else {
    remove process descriptor from front of blocked list;
    insert the descriptor at end of ready list;
  }
  dispatcher();
}
```

Figure 6.5 Semaphore primitives for a single-processor kernel.

```
procedure enter(int mName) {
  find descriptor for monitor mName;
  if (mLock == 1) {
    insert descriptor of executing at end of entry queue;
    executing = 0;
  }
  else
    mLock = 1;    # acquire exclusive access to mName
  dispatcher();
}
procedure exit(int mName) {
  find descriptor for monitor mName;
  if (entry queue not empty)
    move process from front of entry queue to rear of ready list;
  else
    mLock = 0;    # clear the lock
  dispatcher();
}
procedure wait(int mName; int cName) {
  find descriptor for condition variable cName;
  insert descriptor of executing at end of delay queue of cName;
  executing = 0;
  exit(mName);
}
procedure signal(int mName; int cName) {
  find descriptor for monitor mName;
  find descriptor for condition variable cName;
  if (delay queue not empty)
    move process from front of delay queue to rear of entry queue;
  dispatcher();
}
```

Figure 6.6 Monitor kernel primitives.

```
shared variables:  sem e = 1;        # one copy per monitor
                   int nc = 0;       # one copy per condition
                   queue delayQ;     #   variable cv
                   sem private[N];   # one entry per process

monitor entry:  P(e);

wait(cv):   nc = nc+1; insert myid on delayQ; V(e);
            P(private[myid]); P(e);

signal(cv):  if (nc > 0) {
               nc = nc-1;
               remove otherid from delayQ;
               V(private[otherid]);
             }

monitor exit:  V(e);
```

Figure 6.7 Implementing monitors using semaphores.