Threads and sequence coordination

Allocating a thread

```
Proc alloc_thread(SP) returns (int) {
        acquire(tt_lock);
         for i = 0 to N-1 do {
              if (ttable[i].status == FREE) {
                  ttable[i].status ß  RUNNABLE;
                  ttable[i].sp ß  SP;
                  release(tt_lock);
                  return(i);}
         release(tt_lock);
         return(ERROR);}
```

Implementing yield

```
Proc yield( ) {
    acquire(tt_lock);
    ttable[id].state ß  RUNNABLE;
    ttable[id].sp ß SP;
    do {id ß  (id+1) mod N}
        while (ttable[id].state /= RUNNABLE);
    ttable[id].state ß  RUNNING;
    SP ß  ttable[id].sp;
    release(tt_lock);
    return;}
```

Using yield

```
Proc send(p, m) {
    while true do {
        acquire(p.lock);
        if (p.in-p.out < N) then {
            p.buffer[p.in mod N] ß  m;
            p.in ß  p.in+1;
            release(p.lock);
            return;}
        release(p.lock);
        yield( ) ; }}
```

Using yield

```
Proc receive(p) returns msg {
  while true do {acquire(p.lock);
      if (p.out < p.in) then {
        var m ß  p.buffer[p.out mod N];
        p.out ß   p.out+1;
        release(p.lock) ;
        return(m);}
      release(p.lock);
      yield( ) ;}}
```

Preemptive Scheduling

```
Proc yield( ) { disable_interrupts;
    acquire(tt_lock);
    ttable[id].state ß  RUNNABLE;
    ttable[id].sp ß SP;
    do {id ß  (id+1) mod N}
        while (ttable[id].state /= RUNNABLE);
    ttable[id].state ß  RUNNING;
    SP ß  ttable[id].sp;
    release(tt_lock);
    enable_interrupts;
    return;}
```

Sequence coordination

```
Proc send(p, m) {
    while true do {
        acquire(p.lock);
        if (p.in-p.out < N) then {
            p.buffer[p.in mod N] ß  m;
            if (p.in == p.out) { notify(p.c_data) ;}
            p.in ß  p.in+1;
            release(p.lock);
            return;}
        release(p.lock);}}
```

```
Proc receive(p) returns msg {
   while true do {acquire(p.lock);
        if (p.out < p.in) then {
           var m ß  p.buffer[p.out mod N];
           if (p.in – p.out == N) { notify(p.c_room) ;}
           p.out ß   p.out+1;
           release(p.lock) ;
           return(m);}
       release(p.lock);}}
```

```
Proc send(p, m) {
    while true do {
        acquire(p.lock);
        if (p.in-p.out < N) then {
            p.buffer[p.in mod N] ß  m;
            if (p.in == p.out) { notify(p.c_data) ;}
            p.in ß  p.in+1;
            release(p.lock);
            return;}
        release(p.lock);
        wait(p.c_room) ;}}
```

```
Proc send(p, m) {
   while true do {
          acquire(p.lock);
           if (p.in-p.out < N) then {
               p.buffer[p.in mod N] ß  m;
               if (p.in == p.out) { notify(p.c_data) ;}
               p.in ß  p.in+1;
               release(p.lock);
               return;}
          wait(p.c_room) ;
          release(p.lock);}}
```

Using condition variables

```
Proc send(p, m) {
        acquire(p.lock);
        while (p.in-p.out == N) do
           {wait(p.c_room, p.lock) ;}
        p.buffer[p.in mod N] ß  m;
        if (p.in == p.out) { notify(p.c_data) ;}
        p.in ß  p.in+1;
        release(p.lock);
        return;
        }
```

Using condition variables

```
Proc receive(p) returns msg {
        acquire(p.lock);
        while (p.out == p.in) do
           { wait(p.c_data, p.lock) ;}
        var m ß  p.buffer[p.out mod N];
        if (p.in – p.out == N) { notify(p.c_room) ;}
        p.out ß   p.out+1;
        release(p.lock) ;
        return(m);
        }
```

Implementing condition variables

```
Proc wait(cvar, lock) {
    yield_wait(cval,lock);
    acquire(lock); }
```

Implementing condition variables

```
Proc yield_wait(cvar, lock,) {
    acquire(tt_lock);
    release(lock);
    ttable[id].lock ß  lock;
    ttable[id].cvar ß  cvar;
    ttable[id].sp ß  SP;
    ttable[id].state ß  WAITING;
    // other yield code
    release(tt_lock);
}
```

```
proc notify (cvar) {
    acquire (tt_lock);
    for (i = 0 to N-1) do {
        if (ttable[i].cvar == cvar &&
                ttable[i].state == WAITING) {
            ttable[i].state ß  RUNNABLE;}
        }
    release(tt_lock);}
```