ΤΑΥΤΟΧΡΟΝΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

ΕΡΓΑΣΙΑ 3

Σαπουντζή Αθανασία Δέσποινα

Θείου Βασιλης

Ασκηση 1

```
pthread_cond_t waitq = PTHREAD_COND_INITIALIZER;
pthread_cond_t work = PTHREAD_COND_INITIALIZER;
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t terminate = PTHREAD_COND_INITIALIZER;
pthread_cond_t main_wait = PTHREAD_COND_INITIALIZER;
```

MAIN THREAD:

```
while(1)  {
  Read input and place it into the buffer
}
for(i=0; i< num_workers; i++) {Create workers}

while (assigns < size) {
    lock(mtx);
    if(waiting == 0) { wait(main_wait, mtx); }
        waiting--; assigns++;
        unlock(mtx); signal(waitq);
    }
    lock(mtx);
    if(waiting < num_workers) { wait(work, mtx); }
 }
for(i = 0; i< num_workers; i++) { signal(waitq); }

 wait(terminate, mtx);
 unlock(mtx);

return;
```

WORKER THREAD:

```
while(1) {
    lock(mtx);
    waiting++;
    if(waiting == 1) { signal(main_wait); }
    if(waiting == num_workers && jobs_done == size) {
        signal(work);
    }

    wait(waitq, mtx);

    if(jobs_done == size) {
        waiting--;
        if( waiting == 0) {
            unlock(mtx);  signal(terminate);
            return;
        }
        else { unlock(mtx); return; }
    }
    else {
        jobs_done++;
        result = primetest();
        print result;
    }
     unlock(mtx);
}
```

Ασκηση 2:

```
pthread_cond_t b_q= PTHREAD_COND_INITIALIZER;
pthread_cond_t r_q= PTHREAD_COND_INITIALIZER;
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
```

MAIN:

```
cars_lim = 2*bridge_size;
for(i = 0; i < input; i++) {create car threads}
for(i = 0; i < input; i++) {  pthread_join(thread[i],NULL);}
return;
```

CAR:

enter_bridge :

```
lock(mtx);
if(car_color == BLUE) {
   if(r_bridge > 0 || b_bridge == bridge_size || b_lim >= cars_lim
      || b_waiting > 0) {
      b_waiting++;   wait(b_q, mtx);
      if(b_waiting > 0 && b_bridge < bridge_size && b_lim < cars_lim) {
         b_waiting--;  b_bridge++;  b_lim++;
         signal(b_q);
      }
   }
   else { b_bridge++;  b_lim++;}
}
if(car_color == RED) {
   if(b_bridge > 0 || r_bridge == bridge_size || r_lim >= cars_lim
      || r_waiting > 0) {
      r_waiting++;  wait(r_q, mtx);
      if(r_waiting > 0 && r_bridge < bridge_size && r_lim < cars_lim) {
         r_waiting--;  r_bridge++;  r_lim++;
         signal(r_q);
      }
   }
   else { r_bridge++;  r_lim++;}
}
unlock(mtx);
```

exit_bridge:

```
lock(mtx);
if(car_color == BLUE) {
   b_bridge--;
   if(b_bridge == 0 && r_waiting > 0) {
      b_lim =0;   r_waiting--;   r_bridge++;   r_lim++;
      signal(r_q);
   }
   else if(b_waiting > 0 && (b_lim < cars_lim || r_waiting == 0)) {
         b_waiting--;  b_bridge++;  b_lim++;
         signal(b_q);
   }
}
if(car_color == RED ) {
    r_bridge--;
    if(r_bridge == 0 && b_waiting > 0) {
        r_lim = 0;  b_waiting--;  b_bridge++;  b_lim++;
        signal(b_q);
    }
    else if(r_waiting > 0 && (r_lim < cars_lim || b_waiting == 0)) {
        r_waiting--;  r_bridge++;   r_lim++;
        signal(r_q);
    }
}
unlock(mtx);
return;
```

Ασκηση 3:

```
pthread_cond_t wait_line= PTHREAD_COND_INITIALIZER;
pthread_cond_t exit_train= PTHREAD_COND_INITIALIZER;
pthread_cond_t train_full= PTHREAD_COND_INITIALIZER;
pthread_cond_t train_empty= PTHREAD_COND_INITIALIZER;
pthread_cond_t train_start= PTHREAD_COND_INITIALIZER;
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
volatile int waiting = 0, boarding = 0, empty = 0, full = 0, start = 0;
```

PASSENGER:

passenger_enter:

```
lock(mtx);
waiting++;
if( waiting == train_capacity  && full) {
    full = 0;  signal(train_full);
 }
 down(wait_line, mtx);
 waiting--;
 boarding++;
 if(boarding < train_capacity) {
    signal(wait_line);
 }
else if(boarding == train_capacity) {
    if(start) {
        start = 0; signal(train_start);
    }
 }
 unlock(mtx);
```

passenger_exit:

```
lock(mtx);
wait(exit_train, mtx);
If(boarding == 0 && empty) {
   empty = 0; signal(train_empty);
 }
else if(boarding  > 0) {
    boarding--;
    signal(exit_train);
}
unlock(mtx);
return;
```

TRAIN:

```
while(1) {
    lock(mtx);
    if(waiting < train_capacity) {
        full = 1;
        wait(train_full, mtx);
    }
    signal(wait_line);
    if(boarding < train_capacity) {
        start = 1;
        wait(train_start, mtx);
     }
    sleep(ride_duration); //RIDE

    boarding--;
    signal(exit_train);
    if(boarding > 0) {
        empty = 1;
        wait(train_empty, mtx);
    }
    unlock(mtx);
}
return;
```

MAIN:

```
Create train
for(i= 0; i < num_passengers i++ ) {
    Create passenger
}

for(i= 0; i < num_passengers i++ ) {
    pthread_join(passengers[i],NULL);
}
pthread_join(train,NULL);

return;
```

```
#define CCR_EXEC(label,cond,body)\
lock(label##mtx1);
while(!cond){\
    label##n1++;\
    if(label##n2>0){\
        lock(label##mtx2);\
        label##n2--; \
        signal(label##q2);\
        unlock(label##mtx2);\
    }\
    else{ unlock(label##mtx1); }\
    lock(&(label##mtx2));\
    if(n1##label>0){\
       wait(label##q1,label##mtx2);\
    }\
    unlock(label##mtx2);\
    label##n2++;\
    if(label##n1>0){\
       lock(label##mtx2);\
       label##n1--;\
       signal(label##q1);\
       unlock(label##mtx2);\
    }\
    else{\
        lock(label##mtx2);\
        label##n2--;\
        signal(label##q2);\
        unlock(label##mtx2);\
    }\
    lock(label##mtx2);\
    if(label##n2>0){\
       wait(label##q2,label##mtx2);\
    }\
    unlock(label##mtx2);\
} \
```

```
body;\
if(label##n1>0){\
    lock(label##mtx2);\
    label##n1--;\
    signal(label##q1);\
    unlock(label##mtx2);\
}\
 else if(label##n2>0){\
    lock(label##mtx2);\
    label##n2--; \
    signal(label##q2);\
    unlock(label##mtx2);\
}\
else {\
    unlock(label##mtx2);\
    unlock(label##mtx1);\
}
```

```
#define CCR_DECLARE(label) \
   volatile int label##n1;\
   volatile int label##n2;\
   pthread_cond_t label##q1, label##q2; \
   pthread_mutex_t label##mtx1, label##mtx2 ;
```

```
#define CCR_INIT(label)
pthread_mutex_init(&(label##mtx1),NULL);\
pthread_mutex_init(&(label##mtx2),NULL);\
pthread_cond_init(&(label##q1),NULL);\
pthread_cond_init(&(label#q2),NULL);\
n1##label = 0; n2##label = 0;
```

Ασκηση 3:

CCR_DECLARE(R)

volatile int flag = 0;
volatile int boarding = 0, exiting = 0;

PASSENGER:

passenger_enter:

```
CCR_EXEC(R,( boarding != train_capacity && flag == 0),{
   boarding++;
})
```

passenger_exit:

```
CCR_EXEC(R, (exiting != train_capacity && flag == 1), {
   exiting++;
})
return;
```

TRAIN:

```
while(1) {
    CCR_EXEC(R, (boarding == train_capacity), {
        boarding = 0;
        flag = 1;
    })
    sleep(ride_duration);
    CCR_EXEC(R, (exiting == train_capacity), {
         exiting = 0;
         flag = 0;
    })
}
return;
```

MAIN:

```
CCR_INIT(R)
Create train
for(i= 0; i < num_passengers i++ ) {
    Create passenger
}
for(i= 0; i < num_passengers i++ ) {
    pthread_join(passengers[i],NULL);
}
pthread_join(train,NULL);

return;
```