# CC4302
# Sistemas Operativos
# Profesor: Luis Mateu

- Lectores/escritores por orden de llegada
- Los cambios de contexto
- Patrón request
- Lectores/escritores eficientes con patrón request
- Otros criterios para el orden de atención

# Recuerdo:
## Lectores/escritores por orden de llegada

```
pthread_mutex_t  m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  c = PTHREAD_COND_INITIALIZER;
int ticket_dist = 0, display = 0; // ticket distributor and display
int readers= 0, writing= 0;
```

```
void enterWrite() {
   pthread_mutex_lock(&m);
   int my_num= ticket_dist++;
   while (my_num!=display &&
            readers>0)
      pthread_cond_wait(&c, &m);
   pthread_mutex_unlock(&m);
}

void exitWrite() {
   pthread_mutex_lock(&m);
   display++;
   pthread_cond_broadcast(&c);
   pthread_mutex_unlock(&m);
}
```

¿Por qué?

```
void enterRead() {
   pthread_mutex_lock(&m);
   int my_num= ticket_dist++;
   while (my_num!=display)
      pthread_cond_wait(&c, &m);
   readers++;
   display++;
   pthread_cond_broadcast(&c);
   pthread_mutex_unlock(&m);
}

void exitRead() {
   pthread_mutex_lock(&m);
   readers--;
   if (readers==0)
      pthread_cond_broadcast(&c);
   pthread_mutex_unlock(&m);
}
```

# **Recuerdo**: numero explosivo de cambios de contexto

- Considere que hay *n* lectores en espera
- Cuando el escritor sale, despierta a los *n* threads
- Algunos threads no tendrán el siguiente número y se volverán a dormir
- Cuando por fin se despierta el thread con el siguiente número, vuelve a despertar a todos los lectores
- En el peor caso se producen $O(n^2)$ cambios de contexto inútiles
- Solución: patrón *request*

```
typedef struct {
    int ready;
    pthread_cond_t  w;
    …
} Request;
```

```
Req req= {0, PTHREAD_COND_INITIALIZER, ...};
while (!req.ready)
   pthread_cond_wait(&req.w, &m);
```

# Lectores/escritores por orden de llegada con patrón request

```
typedef enum { READER, WRITER } Kind;
pthread_mutex_t  m = PTHREAD_MUTEX_INITIALIZER;
int readers= 0, writing= 0;
Queue q; // = makeQueue()
```

```
void enterWrite() {
   pthread_mutex_lock(&m);
   if (readers==0 && !writing)
      writing = 1;
   else
      enqueue(WRITER);
   pthread_mutex_unlock(&m);
}
void exitWrite() {
   pthread_mutex_lock(&m);
   writing = 0;
   wakeup();
   pthread_mutex_unlock(&m);
}
```

```
void enterRead() {
   pthread_mutex_lock(&m);
   if (!writing && emptyQueue(q))
      readers++;
   else
      enqueue(READER);
   pthread_mutex_unlock(&m);
}
void exitRead() {
   pthread_mutex_lock(&m);
   readers--;
   if (readers==0)
      wakeup();
   pthread_mutex_unlock(&m);
}
```

# Lectores/escritores por orden de llegada con patrón request (2da parte)

```
typedef struct {
   int      ready;
   Kind    kind;
   pthread_cond_t  w;
} Request;

void enqueue(Kind kind) {
   // ¡Patrón request!
   Request req= { 0, kind,
      PTHREAD_COND_INITIALIZER };
   put(q, &req);
   while (!req.ready)
      pthread_cond_wait(&req.w, &m);
}
```

```
void wakeup() {
   Req *pr= peek(q); // get(q)
   if (pr==NULL)
      return;

   if (pr->kind==WRITER) {
      writing= 1; // entra un escritor
      pr->ready= 1;
      pthread_cond_signal(&pr->w);
      get(q);  // Sale de la cola
   }
   else { // ¡ pr->kind==READER !
      do { // entran lectores consecutivos
         readers++;
         pr->ready= 1;
         pthread_cond_signal(&pr->w);
         get(q);  // Sale de la cola
         pr= peek(q);
         // Si es escritor no sale de la cola
      } while ( pr!=NULL &&
                  pr->kind==READER );
   }
}
```

# Comparación

- Solución que saca número: $O(n^2)$ cambios de contexto inútiles en el peor caso
- Solución con patrón request: ningún cambio de contexto inútil

# Criterios de entrada para los lectores/escritores

- Orden indefinido o con prioridades: pueden conducir a hambruna.
- Por orden de llegada: intuitivo, pero reducen paralelismo.
- *Por orden de llegada, pero cuando le toca a un lector, entran todos los lectores en espera. Si llega un nuevo lector, ingresa si y solo si no hay un escritor dentro o en espera. No reduce tanto el paralelismo.*
- Entradas alternadas (propuesto).

# Lectores/escritores: orden de llegada, entran todos los lectores en espera (I)

```
typedef enum { READER, WRITER } Kind;
pthread_mutex_t  m = PTHREAD_MUTEX_INITIALIZER;
int readers= 0, writing= 0;
Queue wq, rq; // = makeQueue()x2
```

*¡Cambios en amarillo!*

```
void enterWrite() {
   pthread_mutex_lock(&m);
   if (readers==0 && !writing)
      writing = 1;
   else
      enqueue(WRITER);
   pthread_mutex_unlock(&m);
}
void exitWrite() {
   pthread_mutex_lock(&m);
   writing = 0;
   wakeup();
   pthread_mutex_unlock(&m);
}
```

```
void enterRead() {
   pthread_mutex_lock(&m);
   if (!writing && emptyQueue(wq))
      readers++;
   else
      enqueue(READER);
   pthread_mutex_unlock(&m);
}
void exitRead() {
   pthread_mutex_lock(&m);
   readers--;
   if (readers==0)
      wakeup();
   pthread_mutex_unlock(&m);
}
```

# Lectores/escritores: orden de llegada, entran todos los lectores en espera (II)

```
typedef struct {
   int  ready;     // int kind;
   pthread_cond_t  w;
} Request;

Request dummy;

void enqueue(Kind kind) {
   Request req= {0, PTHREAD_COND_INITIALIZER};
   if (kind==WRITER)
      put(wq, &req);
   else {
      // Para recordar el turno de
      // los lectores
      if (emptyQueue(rq))
         put(wq, &dummy);
      put(rq, &req);
   }
   while (!req.ready)
      pthread_cond_wait(&req.w, &m);
}
```

```
void wakeup() {
   Req *pr= get(wq); // peek(wq)
   if (pr==NULL)
      return;

   if (pr!=&dummy) {
      // es el turno de un escritor
      writing= 1;
      pr->ready= 1;
      pthread_cond_signal(&pr->w);
   }
   else {
      // es el turno de todos los
      // lectores
      while (!emptyQueue(rq)){
         pr= get(rq);
         readers++;
         pr->ready= 1;
         pthread_cond_signal(&pr->w);
      }
   }
}
```

# Ejercicio propuesto

- Resuelva el problema de los lectores/escritores con entradas alternadas: …, entra **un** escritor, entran todos los electores en espera, luego **un** escritor, luego todos los lectores en espera, luego **un** escritor, etc.
- Si llega un lector, ingresa si y solo si no hay un escritor dentro o en espera.
- Si llega un escritor ingresa si y solo si no hay nadie dentro.
- Fue pregunta de control el semestre primavera de 2018:

  https://users.dcc.uchile.cl/~lmateu/CC4302/controles/c1-182.pdf
- El enunciado ayuda bastante a resolver el problema.