# Sistemas Operativos

## Sincronización de Threads
## uso de timeouts y
## patrón productor/consumidor

Pablo Jaramillo

# Timeouts

# Timeouts

## Interfaz - mutex

```
int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime)
```

Es un wait que deja de esperar y provoca que el thread se despierte automáticamente cuando el reloj de la máquina alcance el tiempo dado por `abstime`, si el thread es despertado antes entonces el timer se cancela.

Retorna `ETIMEDOUT` cuando se acaba el tiempo.

# Timeouts

## Interfaz - mutex

```
int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime)
```

Es un wait que deja de esperar y provoca que el thread se despierte automáticamente cuando el reloj de la máquina alcance el tiempo dado por `abstime`, si el thread es despertado antes entonces el timer se cancela.

Retorna `ETIMEDOUT` cuando se acaba el tiempo.

## wait!!!

# Timeouts

## Interfaz - mutex

```
int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime)
```

Es un wait que deja de esperar y provoca que el thread se despierte automáticamente cuando el reloj de la máquina alcance el tiempo dado por `abstime`, si el thread es despertado antes entonces el timer se cancela.

Retorna `ETIMEDOUT` cuando se acaba el tiempo.

## ¿Qué es timespec?

# Timeouts

## Interfaz - mutex

```
int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime)
```

Es un wait que deja de esperar y provoca que el thread se despierte automáticamente cuando el reloj de la máquina alcance el tiempo dado por `abstime`, si el thread es despertado antes entonces el timer se cancela.

Retorna `ETIMEDOUT` cuando se acaba el tiempo.

## ¿Qué es timespec?

```
struct timespec {
    time_t tv_sec;
    long tv_nsec;
};
```

Una estructura que contiene intervalos de tiempo en segundos y nanosegundos

```
struct timespec ts;
clock_gettime(CLOCK_REALTIME, &ts);
```

Esto sirve para obtener el timestamp de ahora

# Timeouts

## a relatable example

```
typedef struct {
    Control c;
    Entrega *e;
}Args;

pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c = PTHREAD_COND_INITIALIZER;
pthread_t estudiantes[140];
Control controles[140];
Entrega entregas[140];
Args args[140];


void resolverControl(void *a){
    Args *args = (Args*) a;
    while(hay_tiempo){
        escribirCaracter(args->c, args->e);
        if(listo(args->c, args->e))
            break;
    }
}
```

# Timeouts

## a relatable example

```
typedef struct {
    Control c;
    Entrega *e;
}Args;

pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c = PTHREAD_COND_INITIALIZER;
pthread_t estudiantes[140];
Control controles[140];
Entrega entregas[140];
Args args[140];


void resolverControl(void *a){
    Args *args = (Args*) a;
    while(hay_tiempo){
        escribirCaracter(args->c, args->e);
        if(listo(args->c, args->e))
            break;
    }
}
```

```
pthread_t estudiantes[140];
Control controles[140];
Entrega entregas[140];
Args args[140];

int hay_tiempo;

void profesor(){
    imprimirControles(controles);
    struct timespec ts;
    clock_gettime(CLOCK_REALTIME, &ts); // tiempo ahora
    ts.tv_sec += 60*60*2;// avanzar reloj segs*mins*horas --> 2 horas
    pthread_mutex_lock(&m);
    for(int i=0; i<140; i++){
        args[i].c = controles[i]; args[i].e = NULL;
        pthread_create(
            &estudiantes[i],NULL,
            resolverControl, &args);
    }
    hay_tiempo = TRUE;
    pthread_cond_timedwait(&c,&m,&ts); // despertar en 2 horas
    hay_tiempo = FALSE;
    for(int i=0; i<140; i++){
        pthread_join(estudiantes[i], NULL); // enterrar estudiante :(
        entregas[i] = *args[i].e;
    }
    pthread_mutex_unlock(&m);
}
```

# 1.- Impresora Compartida

# Impresora compartida

## contexto

Se tiene una impresora en el Toqui controlada por el servidor de Anakena, Anakena utiliza threads para controlar la impresora, donde el thread contiene la información sobre qué imprimir, si el servidor permitiera que dos o más threads operaran la impresora a la vez se mezclarían las líneas de los documentos. Los threads que interactúan con la impresora corresponden a conexiones a los computadores del Toqui.

Nosotros tenemos que evitar dicho problema hacienda que cada thread que solicite utilizar la impresora deba tener acceso exclusivo a esta antes de ocuparla u que notifique cuando termine de utilizarla. Por lo que la conexión se ve más o menos así:

```
tarea(){
    ...
    obtenerImpresora();
    ...  /* utilizar impresora*/
    devolverImpresora();
    ...
}
```

# Impresora Compartida

## objetivo

Además se nos pide ahorrar electricidad tal que si la impresora no se usa en 5 minutos después de su última tarea esta entrará en modo de bajo consumo, esto lo podemos hacer con la función `modoBajoConsumo()`. Para volver a utilizar la impresora se debe invocar `modoUsoNormal()`.

Implemente `obtenerImpresora()`, `devolverImpresora()` e `inicializarImpresora()` considerando todo lo mencionado.

# Patrón Productor-Consumidor

# Patrón Productor-Consumidor

Es una forma de organizar threads tal que hayan threads realizando trabajo mientras que un(os) otro(s) preparan más carga.

El productor puede estar pre-procesando data, buscando data desde el disco, ser un servidor, etc.

El consumidor completa tareas mientras hayan tareas, no necesariamente trabaja una cantidad predeterminada.

# 2.- Detector de plagio

(Función masParecidas)

# Detector de plagio

## contexto

Para detectar copias/plagio en tareas un profesor utiliza la función **masParecidas**, la cual encuentra las 2 tareas más similares de un conjunto. Esta recibe una arreglo de **n** tareas de alumnos, 2 punteros **pi** y **pj** con las direcciones de las variables de donde se almacenan los índices de las 2 tareas más parecidas.

```
void masParecidas(Tarea *tareas, int n, int *pi, int *pj){
    int min = INT_MAX;
    for(int i=0; i<n ; i++){
        for(int j=0; j<i; j++){
            int similitud = compTareas(tareas[i], tareas[j]);
            if(similitud < min){
                min = similitud;
                *pi=i;
                *pj=j;
    }}}}
```

# Detector de plagio

## objetivo

La función `compTareas` se tiene y entrega un coeficiente de similitud (o distancia), tal que 0 es que son iguales. Una comparación toma tiempo variable desde casi inmediato hasta varios minutos y se tienen que hacer $O(n^2)$ comparaciones, lo cual tarda mucho tiempo de CPU.

Se nos pide paralelizar `masParecidas` considerando una maquina octa-core. Mas de esto no sería factible por temas de memoria. Queremos desempeño continuo de los cores hasta que se terminen todas las comparaciones.

# Detector de plagio

## hints

- Utilice el Patrón Productor/Consumidor
- El productor deberá generar Jobs con pares de tareas para comparar
- Los múltiples consumidores deberán extraer los Jobs para realizar comparaciones