

# Programación de software de sistemas

## Ayudantía 5

Profesor: Rodrigo Verschae

Ayudante: Nicolás Araya



# Procesos Livianos/Threads

## Creación de threads

```
int pthread_create(pthread_t * thread,           //almacena thread id
                  const pthread_attr_t * attr,   //atributos para el thread o NULL
                  void * (*start_routine)(void *), //puntero a la función a ejecutar
                  void *arg);                   //argumento único para función a ejecutar
```

## Termino de threads

```
void pthread_exit(void *retval);                //valor de retorno del thread
```

## Esperar termino de threads

```
int pthread_join(pthread_t thread, void **retval);
```

Si un thread no es enterrado, se dice que queda en estado zombie.





1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33

# ¡Manos a la obra!

## Problema 1

Crear una matriz con elementos aleatorios y posteriormente realizar una suma parcial correspondiente a una parte de la matriz por cada hilo.



# Mutex

Proporcionan una solución al problema de la exclusión mutua, delimitando una sección crítica. Un mutex tiene dos estados “abierto” y “cerrado” (como un candado).

```
pthread_mutex_t mutex;           //tipo definido en pthread.h

pthread_mutex_init(&mutex, NULL); //inicializa mutex

pthread_mutex_destroy(&mutex);    //libera memoria del mutex

pthread_mutex_lock(&mutex);       //Pide la propiedad del mutex.
                                  //Si el mutex está tomado, se
                                  //bloquea a la espera

pthread_mutex_unlock(&mutex);     //Libera el mutex
```





# Mutex

Para entrar a una sección crítica, un thread debe solicitar la propiedad del mutex (cerrar el candado)

Al salir de una sección crítica, un thread debe liberar la propiedad del mutex (abrir el candado)

```
pthread_mutex_lock(&mutex);  
  
/* SECCIÓN CRÍTICA */  
  
pthread_mutex_unlock(&mutex);
```



# Problema 2

Según la cantidad de hilos, sumar una variable declarada globalmente.





# Monitores

```
pthread_mutex_t mutex;           //tipo definido en pthread.h

pthread_cond_t cond;             //tipo definido en pthread.h

pthread_cond_init (&cond, NULL); //inicializar condición

pthread_cond_destroy(&cond);      //libera memoria de la cond.

pthread_cond_wait(&cond, &mutex); //espera en forma eficiente

pthread_cond_broadcast(&cond);    //notifica a todos los threads
                                   //que estén durmiendo en cond

pthread_cond_signal(&cond);       //notifica a un thread
                                   //que esté durmiendo en cond
```



# Problema 3

A usted le molesta leer en digital, así que tiende a imprimir todo lo que le mandan de la universidad. Lamentablemente no es el único que utiliza la impresora, existen otros estudiantes que acaparan la impresora y le pueden impedir imprimir. Para realizar un modelado realista, para usar la impresora debe ser el primero que la toma, nadie respeta ninguna fila.

Simule este comportamiento mediante programación concurrente y monitores.





# Productor/Consumidor



```
1 void *producer(void *arg) {
2     for (int i = 0; i < 10; i++) {
3         int item = rand() % 100; // Producir un elemento
4         pthread_mutex_lock(&mutex);
5         while (count == BUFFER_SIZE)
6             pthread_cond_wait(&empty, &mutex);
7         buffer[in] = item;
8         in = (in + 1) % BUFFER_SIZE;
9         count++;
10        printf("Productor produjo: %d\n", item);
11        pthread_cond_signal(&full);
12        pthread_mutex_unlock(&mutex);
13    }
14    pthread_exit(NULL);
15 }
```





```
1 void *consumidor(void *arg) {  
2  
3     for (int i = 0; i < 10; i++) {  
4  
5         int item;  
6  
7         pthread_mutex_lock(&mutex);  
8  
9         while (count == 0)  
10             pthread_cond_wait(&full, &mutex);  
11  
12         item = buffer[out];  
13  
14         out = (out + 1) % BUFFER_SIZE;  
15  
16         count--;  
17  
18         printf("Consumidor consumió: %d\n", item);  
19  
20         pthread_cond_signal(&empty);  
21  
22         pthread_mutex_unlock(&mutex);  
23  
24     }  
25  
26     pthread_exit(NULL);  
27  
28 }  
29  
30  
31  
32  
33
```



# Problema 4

Implemente un sistema de cola en donde la cola se vaya llenando con el productor y vaciando con el consumidor.

Utilice monitores y exclusión mutua.

