Universidad de los Andes

Infraestructura computacional

Concurrencia

Universidad de los Andes

Concurrencia: otros niveles en java

- Adjetivo `volatil`:
  - Para declarar variables accedidas desde varios threads
  - Accesos sincronizados en memoria
- Clases atómicas:
  - `AtomicInteger` y `AtomicLong`:
    - `int get( ), void set( int nuevoValor )`
    - `int getAndSet( int nuevoValor )`
    - `int addAndGet( int incremento )`
    - `boolean compareAndSet( int esperado, int nuevoValor )`
  - `AtomicBoolean`
  - `AtomicReference<T>`
  - `AtomicLongArray, AtomicIntegerArray, AtomicReferenceArray<T>`

Universidad de los Andes

Concurrencia: otros niveles en java

- Semáforos
- Locks explícitos:
  - Interfaz `Lock` (implementación: `ReentrantLock`):
    - `void lock( )`
    - `boolean tryLock( )`
    - `boolean tryLock( long timeOut, TimeUnit unidad )`
    - `void unlock( )`
  - Interfaz `ReadWriteLock` (impl. `ReentrantReadWriteLock`)
- Interfaz `Condition`:
  - Método en `Lock`: `Condition newCondition( )`
  - Métodos
    - `void await( )`
    - `void signal( ), void signalAll( )`

Universidad de los Andes

Concurrencia: otros niveles en java

- `BlockingQueue<T>`:
  - Interfaz con múltiples implementaciones
  - Métodos:

	Excepción	Valor de retorno	Bloqueo	Time out
Insertar	add(e)	offer(e)	put(e)	offer(e, t, u)
Remover	remove( )	poll( )	take( )	poll(t, u)
Examinar	element( )	peek( )	---	---

Universidad de los Andes

Concurrencia: otros niveles en java

- `ConcurrentHashMap<K,V>`
  - Tabla de hash con llave *k* y valor *v*
  - Las operaciones de lectura no se bloquean
  - Las operaciones de escritura soportan un nivel de concurrencia especificado por el programador
- `CopyOnWriteArrayList<E>`
  - Las operaciones de lectura no se bloquean
  - Las operaciones de lectura se pueden realizar simultáneamente con una escritura
  - Las escrituras se realizan en exclusividad

Universidad de los Andes

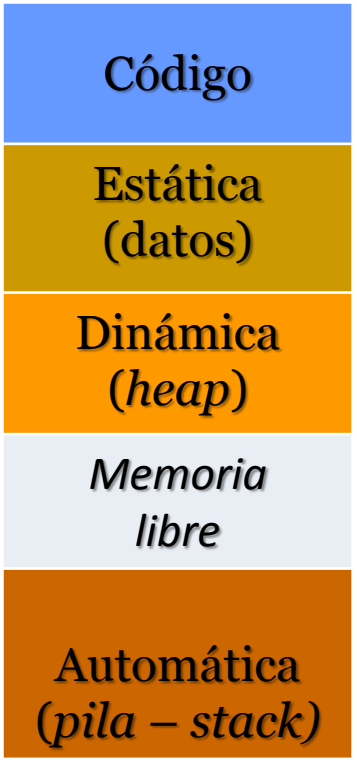
Concurrencia: pthreads

- Estándar Posix
- Threads de lenguaje o del sistema
- Creación de threads:
  - Programa de C común y corriente
  - La ejecución empieza en el main
  - Los threads activan funciones del programa
  - Función para crearlos:
    `int pthread_create:` 0 = terminó bien
    - `pthread_t * thread`: identificador del thread creado
    - `const pthread_attr_t * attr`: atributos (NULL = default)
    - `void * (*start_routine)(void *)`: función del thread
    - `void *arg`: parámetro de `start_routine`

Universidad de los Andes

Concurrencia: pthreads

- Uso de la memoria
  - ¿En qué parte de la memoria se encuentran?
  - ¿Cuándo se crean?
  - ¿Cuándo se destruyen?
  - ¿Puede haber múltiples instancias?




```
int vg[5];

int f ( int * p, int i ){

  int vl[5];
  int *q = (int *) calloc(5, sizeof (int));
```

Universidad de los Andes

Concurrencia: pthreads

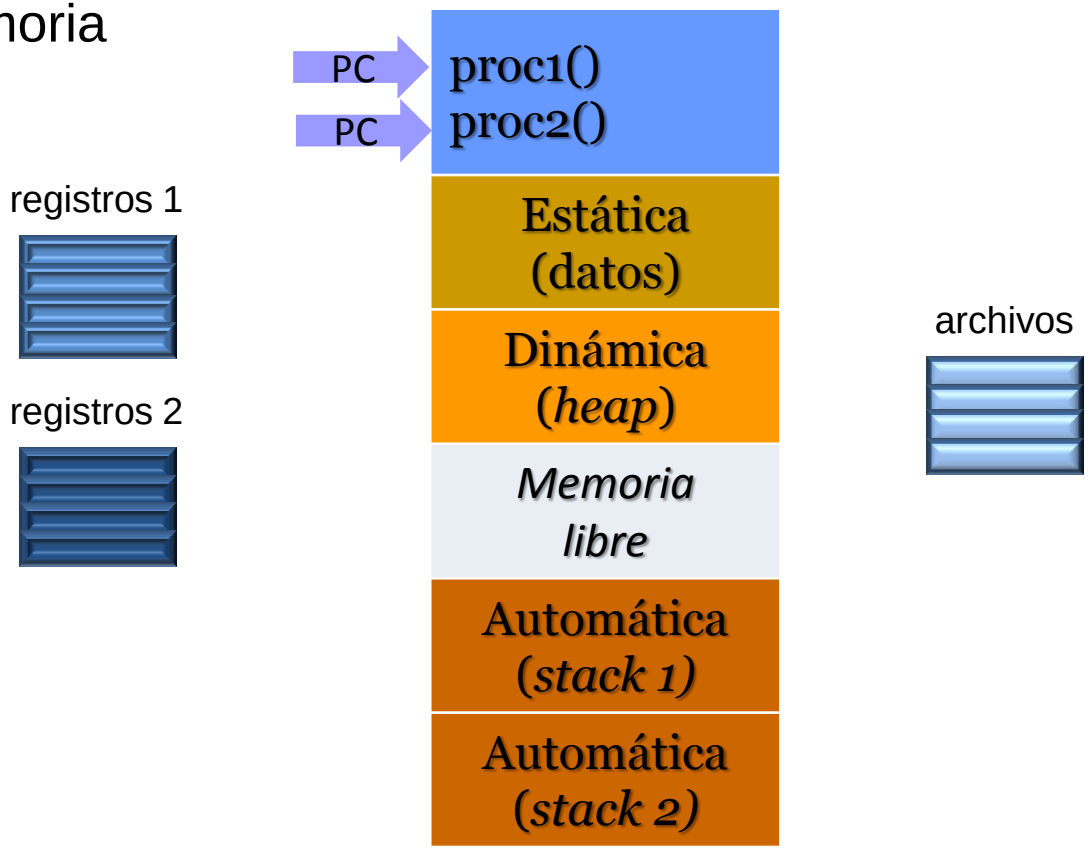
Generar n threads con identificación

```
  void * funcion( void * p ){ ... }

  pthread_t threadId[ nThreads ];
  int id[ nThreads ];

  for( i = 0; i < nThreads; i++ ){
    id[ i ] = i;
    nok = pthread_create(
              &threadId[ i ],
              NULL,
              funcion,
              (void *) &id[ i ] );
}
```

Universidad de los Andes

Concurrencia: pthreads

- Modelo de memoria



Universidad de los Andes

Sincronización - exclusión mutua

- Mutex:
  - Candados
  - Solo un thread puede poseer el recurso
  - Otros thread quedan en espera
  - El thread propietario cede el candado al salir
  - Uso:
    - `Adquirir( mutex )`
    - `... sección crítica`
    - `Libera( mutex )`

Universidad de los Andes

Sincronización - exclusión mutua

- Pthreads: mutex

```
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
  …
pthread_mutex_lock ( &m );
  [ o pthread_mutex_trylock ( &m ) ]
  … acciones en exclusión mutua
pthread_mutex_unlock ( &m );
  …
pthread_mutex_destroy ( &m );
```

Universidad de los Andes

Sincronización - variables de condición

- Pthreads:

```
pthread_cond_t c= PTHREAD_COND_INITIALIZER;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
 …
pthread_cond_wait ( &c, &m );
 …
pthread_cond_signal ( &c );
   [ o pthread_cond_broadcast ( &c ) ]
  …
pthread_cond_destroy( &c );
```

Universidad de los Andes

Sincronización - encuentros

- Pthreads: join
  - Función para esperar la finalización de un thread:
    `int pthread_join:` 0 = terminó bien
    - `pthread_t thread`: identificador del thread que se espera
    - `void ** valor_retorno`: apuntador al valor de retorno del thread
  - El valor de retorno se devuelve con:
    `int pthread_exit ( void * retorno ):` 0 = terminó bien

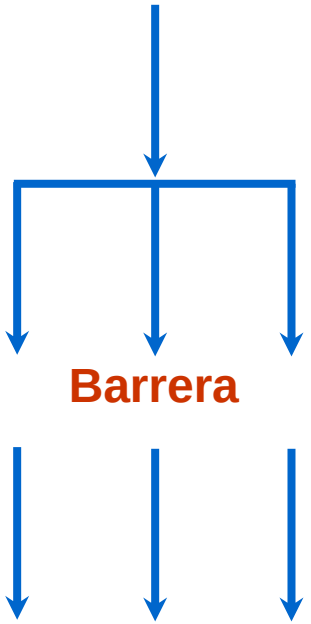
```
for( i = 0; i < NTHREADS; i++ )
  pthread_create( &t[ i ], NULL, funcion, NULL );
  …
for( i = 0; j < NTHREADS; i++ )
  pthread_join( t[ i ], NULL);
```

Universidad de los Andes

Sincronización - encuentros

- Pthreads: barriers

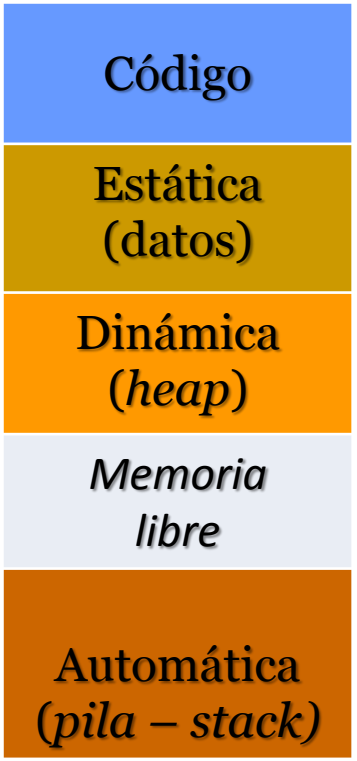
```
pthread_barrier_t barrera =
   PTHREAD_BARRIER_INITIALIZER( n );
   …
pthread_barrier_wait( &barrera );
```



Universidad de los Andes

Concurrencia: procesos

- Proceso: ejecución de un programa
  - Código (sección de texto)
  - Estado (actividad corriente):
    - Registros (incluye PC)
    - Pila (parámetro, var. locales, dirección de retorno)
    - Sección de datos (variables globales)
    - Montón (heap)
- Pid: process identifier. Identificador ante el sistema
- Creación de procesos:
  - Proceso origen
  - Llamada al sistema para crear nuevos procesos
  - Proceso padre genera procesos hijos

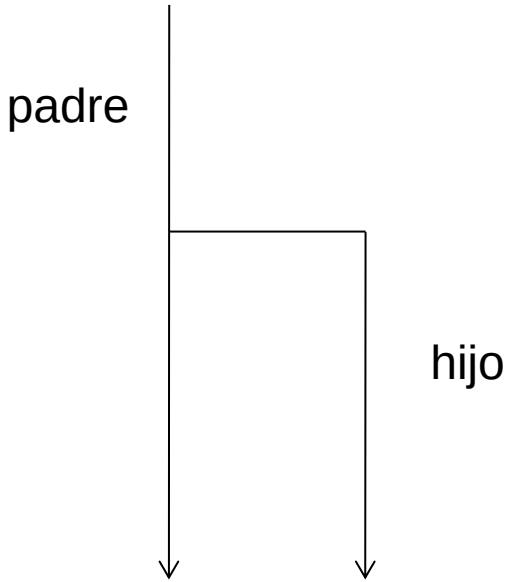


Universidad de los Andes

Creación de procesos en Unix

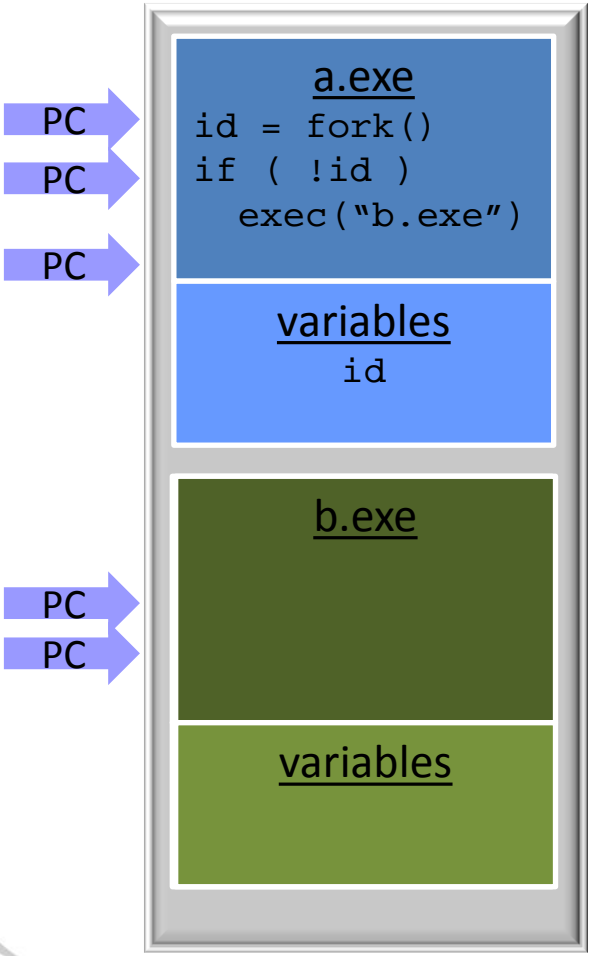
- Proceso hijo recibe una copia de la memoria del padre
- Los procesos son independientes y no comparten memoria (salvo las instrucciones)
- Las llamadas a `exec()` permiten cambiar el ejecutable

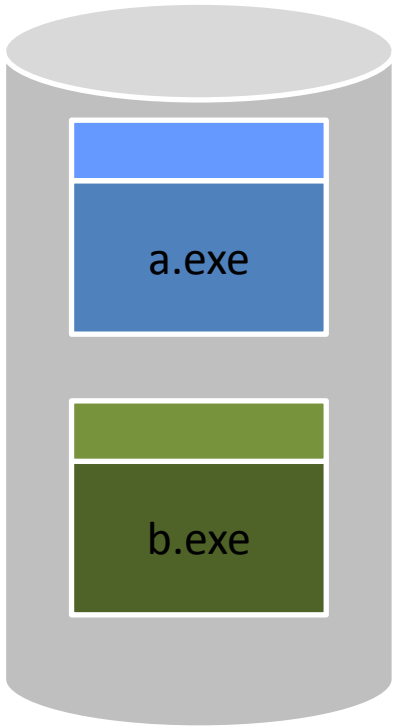
```
id = fork()
if ( id == 0 ) {
  código del hijo
}

código del padre
```



Universidad de los Andes

Creación de procesos en Unix





Universidad de los Andes

Creación de procesos en Unix

- Esquema de servidor concurrente

```
while ( … ) {
   s = recibirSolicitud
   id = fork()
   if ( id == 0 )
     atender s
}
```

Universidad de los Andes

Creación de procesos en Unix

- Ejercicio:
  - ¿Cuántos procesos, además del original, crea el siguiente programa?

```
for ( i = 1; i < 4; i++ )
  id = fork ();
```

Universidad de los Andes

Creación de procesos en Windows

- `CreateProcess(`
  - `nombreApp  //NULL = usar línea de comando`
  - `lineaComando  //"C:\\dir\\subdir\\app.exe”`
  - `… otros parámetros  )`