SISTEMA ERAGILEAK
SISTEMAS OPERATIVOS

Tema 2. Gestión de procesos (7)

Informatika Ingeniaritzako Gradua
Grado en Ingeniería en Informática
Informatika Fakultatea

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

NAZIOARTEKO BIKAINTASUN CAMPUSA
CAMPUS DE EXCELENCIA INTERNACIONAL

Threads

```
#include <pthread.h>
```

Compilar

```
gcc -o prog prog.c -pthread
```

Identificador de un thread

```
pthread_t thread_id;
```

- **pthread_t** está definido como:

  typedef unsigned long int pthread_t;

  Un **pthread_t** se debería imprimir como **unsigned long int.**

Crear threads

```
int pthread_create( pthread_t *thread,
      pthread_attr_t *attr,
      void *(*thread_function)(void *), void *arg );
```

- **`thread`**: Puntero al identificador del hilo creado.
- **`attr`**: Atributos del hilo. Si es `NULL`, el hilo se crea con los atributos predeterminados.
- **`thread_function`**: Puntero a la función que ejecutará el hilo.
- **`arg`**: Argumento de la función.

- Devuelve **`0`** en caso de éxito.

Finalización de threads

```
void pthread_exit( void *thread_return )
```

- Finaliza y pone disponible un código de retorno para el padre que lo recoja.
- **`thread_return`**: Puntero a un puntero al código de retorno del hilo.
- No devuelve valor.

Esperar threads

```
int pthread_join( pthread_t thread,
     void **thread_return )
```

- El hilo padre espera a que termine el hilo hijo.
- Los hilos siempre deben estar recogidos; de lo contrario, un hilo podría seguir ejecutándose incluso cuando el hilo padre ya haya terminado.

- **`thread`**: El hilo a esperar.
- **`thread_return`**: Puntero a un puntero al código de retorno del hilo.

- Devuelve **`0`** en caso de éxito.

Función creada por pthread_create()

```
void*  thread_function ( void*  args_p ) ;
```

- **`void *`** se puede convertir (*cast*) a cualquier tipo de puntero en C.
- **`args_p`**: Puntero a una lista que contiene uno o más valores necesarios para **`thread_function`**.
- De forma similar, el valor de retorno de **`thread_function`** puede apuntar a una lista de uno o más valores.

Función creada por pthread_create()

```c
#include <stdio.h>
#include <pthread.h>

typedef struct {
    int num;
    char *string;
} args_t;

void *func(void *p) {
    args_t *args = p;
    printf("%d %s\n", args->num, args->string);
    pthread_exit(NULL);
}

int main(){
    int max = 10;
    char buffer[] = "mensaje";     // *buffer = "mensaje";
    args_t args;
    pthread_t tid;

    args.num = max;
    args.string = buffer;

    pthread_create(&tid, 0, func, &args);
    pthread_join(tid,NULL);
    return(0);
}
```

Obtener el identificador de un thread

```
pthread_t pthread_self(void);
```

- Devuelve el identificador del hilo que realiza la llamada.
- No devuelve código de error.

Mutex

```
#include <pthread.h>
```

Declaración de un mutex

```
pthread_mutex_t mutex;
```

- **`mutex`**: Puntero a una estructura de tipo `pthread_mutex_t`.

Crear mutex

```
int pthread_mutex_init(pthread_mutex_t *mutex,
      const pthread_mutexattr_t *mutexattr);
```

- **`mutex`**: Puntero a un objeto mutex.
- **`mutexattr`**: Puntero a los atributos del mutex. Si es NULL, se utilizan los atributos predeterminados.

- Devuelve **`0`** en caso de éxito.

Bajar mutex

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

- Se bloquea el objeto mutex. Si el mutex ya está bloqueado, el hilo de llamada se bloquea hasta que el mutex esté disponible. Cuando la función progresa, el objeto mutex está bloqueado y es propiedad del hilo llamante.

- **`mutex`**: Puntero a un objeto mutex.

Subir mutex

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- Desbloquea el mutex. Si hay hilos bloqueados en el mutex, el hilo en espera de mayor prioridad se desbloquea y se convierte en el próximo propietario del mutex.
- **`mutex`**: Puntero a un objeto mutex.
- Devuelve **`0`** en caso de éxito.

Eliminar mutex

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

- **`mutex`**: Puntero a un objeto mutex.
- Devuelve **`0`** en caso de éxito.

Semáforos

```
#include <semaphore.h>
```

Declaración de un semáforo

```
sem_t sem;
```

- **`sem`**: Puntero a un objeto de tipo `sem_t`.

Crear semáforo

```
int sem_init(sem_t *sem, int pshared,
      unsigned int value);
```

- **`sem`**: Puntero a un objeto semáforo.
- **`pshared`**: Es una opción de compartir; un valor de `0` significa que el semáforo es local al llamante.
- **`value`**: Valor inicial del semáforo.

- Devuelve `0` en caso de éxito.

Bajar semáforo

```
int sem_wait(sem_t *sem);
```

- Decrementa, atómicamente, en 1 el valor del semáforo.
  - Si `sem > 0`, entonces se decrementa (`sem--`) y la función retorna.
  - Si `sem == 0`, la función se bloquea (encola) hasta que se pueda decrementar (`sem > 0`).
- **`sem`**: Puntero a un objeto semáforo.
- Devuelve `0` en caso de éxito.

Subir semáforo

```
int sem_post(sem_t *sem);
```

- Incrementa, atómicamente, en 1 el valor del semáforo.
  - Si `sem > 0`, otro hilo bloqueado se despertará y retendrá el semáforo.
- **`sem`**: Puntero a un objeto semáforo.
- Devuelve `0` en caso de éxito.

Eliminar semáforo

```
int sem_destroy(sem_t *sem);
```

- Libera los recursos asignados al semáforo. Generalmente llamado después de **`pthread_join()`**. Si se destruye un semáforo que está esperando un hilo, se producirá un error.

- **`sem`**: Puntero a un objeto semáforo.

- Devuelve `0` en caso de éxito.

En **C** **los punteros** y la **precedencia de operadores** en las expresiones dan dolores de cabeza

Precedencia	Operadores	Dirección
1	`++ --` `()` `[]` `.` `->` `(type){list}`	*izda a dcha* *(>>>>>>>>>)*
2	`++ --` `+ -` `! ~` `(type)` `*` `&` `sizeof` `_Alignof`	*dcha a izda* *(<<<<<<<<<)*
3	`* / %`	*izda a dcha* *(>>>>>>>>>)*
4	`+ -`	
5	`<< >>`	
6	`< <=` `> >=`	
7	`== !=`	
8	`&`	
9	`^`	
10	`|`	
11	`&&`	
12	`||`	
13	`?:`	*dcha a izda* *(<<<<<<<<<)*
14	`=` `+= -=` `*= /= %=` `<<= >>=` `&= ^= |=`	
15	`,`	

Parámetros · punteros

Ejemplo:

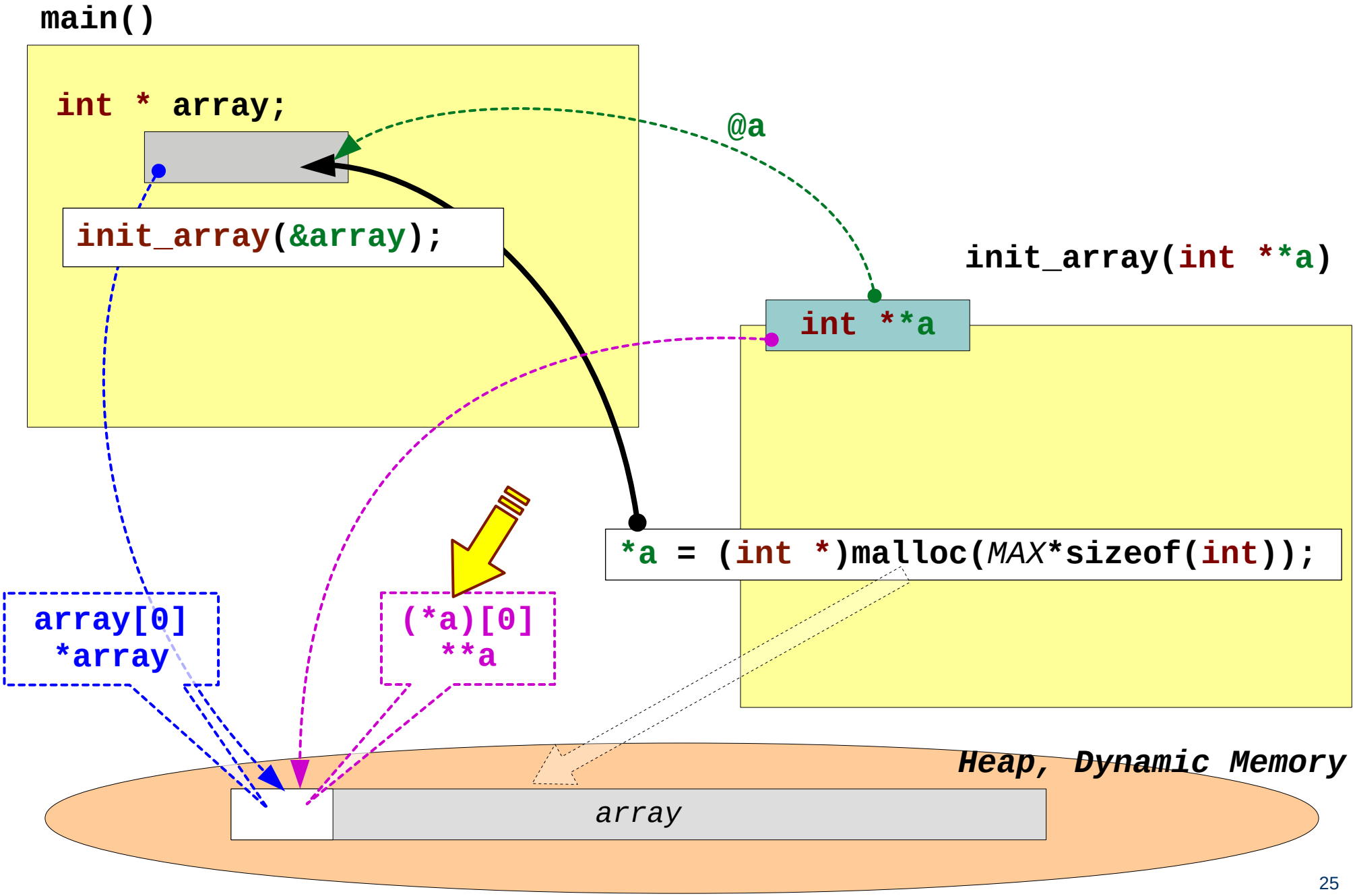
Crear un array dinámico de MAX elementos que contenga la potencia de 2 correspondiente al índice del array (p.e. array[5] → 32), pasando la referencia del array como parámentro utilizando:

- a) Funciones.
- b) Threads (modo 1).
- c) Threads (modo 2).

a) Parámetros · punteros · funciones

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX  16

void init_array(int **a){
   int i;

   *a = (int *) malloc(MAX*sizeof(int));

   for (i=0; i<MAX; i++) (*a)[i] = = 0x41 + i;
}

int main(int argc, char *argv[]){
   int *array, i;

   init_array(&array);

   for (i=0; i<MAX; i++) printf("array[%02d]: %c\n", i, array[i]);

   free(array);
} // main
```

a) Parámetros · punteros · funciones



a) Parámetros · punteros · funcion


```
main()
int * array;
init_array(&array);
```

```
init_array(int **a)
int **a
a = (int *)malloc(MAX*sizeof(int));
```

```
a[0]
*a
```

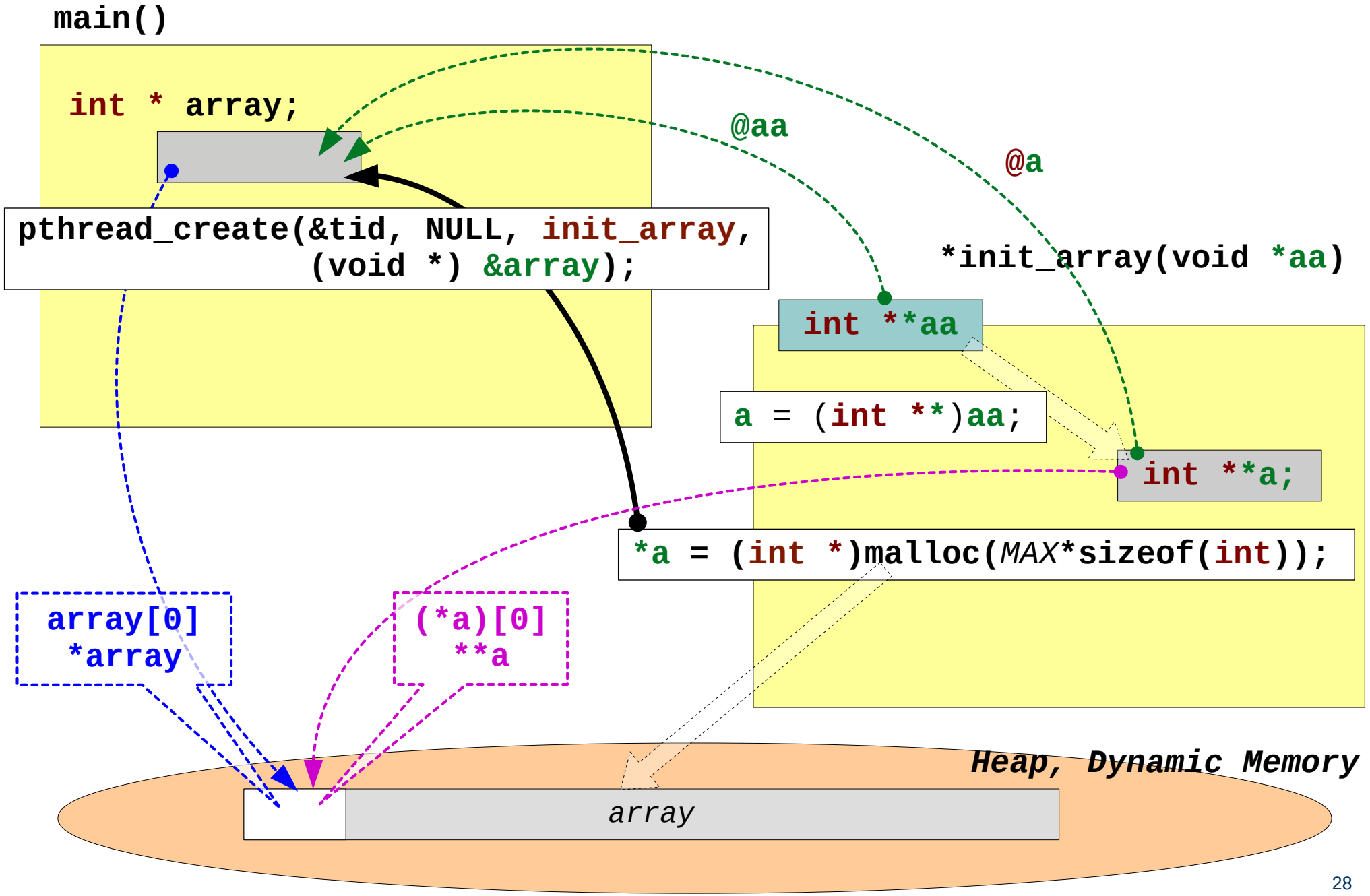
Heap, Dynamic Memory

array

b) Parámetros · punteros · threads (1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define MAX  16

void *init_array(void *aa){
   int **a, i;

   a = (int **) aa;
   *a = (int *) malloc(MAX*sizeof(int));
   for (i=0; i<MAX; i++) (*a)[i] = 1<<i;
   pthread_exit(NULL);
}

int main(int argc, char *argv[]){
   int *array;
   pthread_t tid;

   pthread_create(&tid, NULL, init_array, (void *) &array);
   pthread_join(tid, NULL);

   for (i=0; i<MAX; i++) printf("%d\n", array[i]);

   free(array);
} // main
```

b) Parámetros · punteros · threads (1)



c) Parámetros · punteros · threads (2)

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define MAX  16

void *init_array(void *a){
   int i;


   *(int **)a = (int *) malloc(MAX*sizeof(int));
   for (i=0; i<MAX; i++) (*(int **)a)[i] = 1<<i;
   pthread_exit(NULL);
}

int main(int argc, char *argv[]){
   int *array;
   pthread_t tid;

   pthread_create(&tid, NULL, init_array, (void *) &array);
   pthread_join(tid, NULL);

   for (i=0; i<MAX; i++) printf("%d\n", array[i]);

   free(array);
} // main
```

c) Parámetros · punteros · threads (2)