

Clase 2

Recap: `exit()` & `wait()`

- Mecanismo de sincronización
- Mecanismo de comunicación entre procesos

Recap: `exit()` & `wait()`

Limitaciones

- Comunicación en un único sentido: 🧒 Hijo a Padre 🧓
- El proceso muere 💀 al comunicar la información
- La información enviada es reducida (8 - bits)

Archivos como alternativa

- Comunicación entre procesos no relacionados
 - Comunicación entre **varios** procesos
- Comunicación entre procesos vivos
- Capacidad de comunicar varios datos

Archivos como alternativa

Limitaciones

- No son ni proveen un mecanismo de sincronización
- Las operaciones en disco son costosas


Clase 2

Memoria compartida

Memoria compartida

- Es una facilidad IPC que ofrece el sistema operativo, la cual posibilita intercambiar datos entre dos o más procesos.
- Se implementan como una región de memoria accesible para dichos procesos
- Dos tipos:
 - POSIX
 - System V

Memoria compartida

- Es una facilidad IPC que ofrece el sistema operativo, la cual posibilita intercambiar datos entre dos o más procesos.
- Se implementan como una región de memoria accesible para dichos procesos
- Dos tipos:
 - POSIX
 - System V  Este curso

Operaciones



Podemos pensar que las memorias compartidas se manejan de forma similar a los archivos. De esa forma es fácil recordar cómo hay que operar.

Operaciones

Operación	Archivos	Memoria Compartida
?		
Abrir	open	
Operar	read/write	
Cerrar	close	
?		

Operaciones

Operación	Archivos	Memoria Compartida
Crear	<code>open (*)</code>	
Abrir	<code>open</code>	
Operar	<code>read/write</code>	
Cerrar	<code>close</code>	
Eliminar	<code>remove</code>	

(*) La operación de creación en el `open` existe en tanto y cuanto se abra el archivo para **escritura**

Creación

`shmget`

Creación `shmget`

```
int shmget ( key_t key, size_t size, int shmflg );
```

- `key_t key`: Clave que se crea con la función `ftok`
- `size_t size`: Tamaño en bytes de la memoria a crear
 - Igual que como funciona `malloc`
- `int shmflg`: Flags
 - Permisos de acceso. Ej 0644 (octal)
 - `IPC_CREAT`: Crea el segmento si no existe
 - `IPC_EXCL`: La operación falla si no se puede crear

Creación `shmget`

```
int shmget ( key_t key, size_t size, int shmflg );
```

- Devuelve un identificador válido del segmento
 - Si falla, devuelve -1, y setea la variable externa `errno`

```
int mem_id = shmget ( mem_key, size, 0644|IPC_CREAT|IPC_EXCL );  
if (mem_id == -1){  
    perror("Error: ");  
    exit(-1);  
}
```

Obtención de claves `ftok`

```
key_t ftok (const char* pathname, int proj_id );
```

Mediante `ftok` podemos generar una clave única para identificar el segmento de memoria

- `const char* pathname`: Path a un archivo que existe y al cual el proceso tiene permiso de lectura
- `int proj_id`: Se utilizan los últimos 8 bits, por lo que se puede usar un char

Obtención de claves `ftok`

```
key_t ftok (const char* pathname, int proj_id );
```

- Devuelve una clave valida
 - Si falla, devuelve -1, y setea la variable externa `errno`

```
key_t mem_key = fytok ( "/tmp/randomfile.c" , 'g' );  
if (mem_key == -1){  
    perror("Error: ");  
    exit(-1);  
}
```

Adosado o *attach*

shmat

Attach `shmat`

```
void* shmat ( int shmid, const void* shmaddr, int shmflg );
```

- `int shmid`: Identificador del segmento
 - (El obtenido con `shmget`)
- `const void* shmaddr`: direccion de memoria a la cual se mapea el segmento. Si es `NULL`, la elige el SO
- `int shmflg`:
 - `SHM_RDONLY` - Solo lectura
 - `0` - Lectura / Escritura`

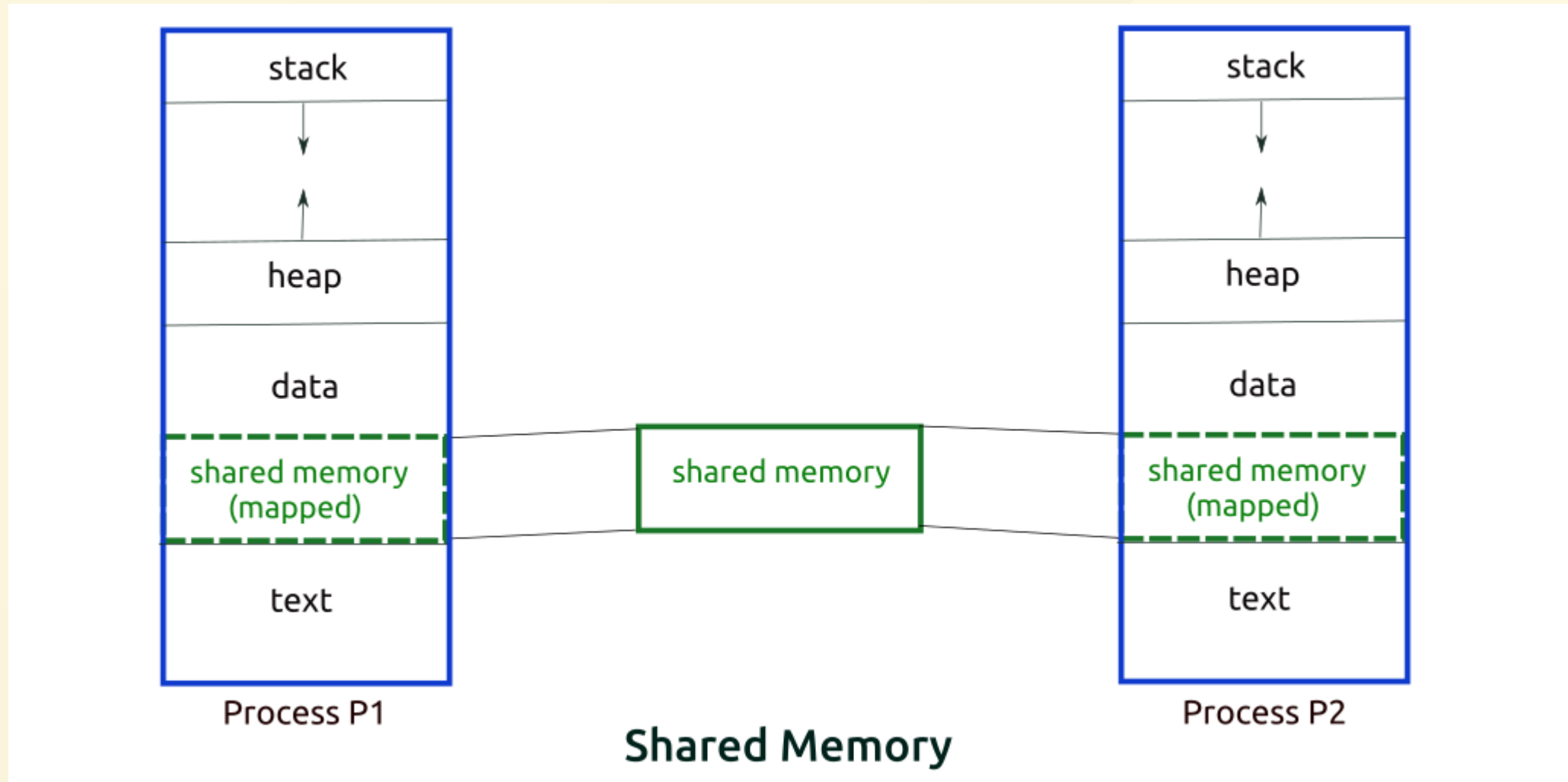
Attach `shmat`

```
void* shmat ( int shmid, const void* shmaddr, int shmflg );
```

- Devuelve un puntero al segmento de memoria compartida
 - Si falla, devuelve `(void*) -1`, y setea la variable externa `errno`

```
int* memory = shmat ( mem_id, NULL, 0 );  
if (memory == -1){  
    perror("Error: ");  
    exit(-1);  
}
```

Attach `shmat`



Operar

Lectura / Escritura

Operaciones Lectura / Escritura

El kernel no participa (mediante syscalls) en la lectura y escritura

Se puede operar sobre el puntero obtenido con `shmat` de igual manera que se puede operar sobre los punteros obtenidos de `malloc`:

- Escritura

```
memory[0] = 3;
```

- Lectura

```
int value = memory[0];
```

Operaciones

Operacion	Archivos	Memoria Compartida	Memoria Dinámica
Crear	<code>open (*)</code>	<code>shmget</code>	<code>malloc</code>
Abrir	<code>open</code>	<code>shmat</code>	<code>malloc</code>
Operar	<code>read/write</code>	punteros	punteros
Cerrar	<code>close</code>	?	?
Eliminar	<code>remove</code>	?	?

** La operación de creacion en el open existe en tanto y cuanto se abra el archivo para **escritura***

Desadosar o *detach*

shmdt

Detach `shmdt`

```
int shmdt( const void* shmaddr );
```

- `shmaddr`: Puntero obtenido con `shmat`
- Devuelve `0` en caso de éxito
 - Si falla, devuelve `-1`, y setea la variable externa `errno`

```
int result = shmdt ( memory );  
if (result == -1){  
    perror("Error: "); //No cortar la ejecución  
}
```

Detach `shmdt`

```
int shmdt( const void* shmaddr );
```

Los procesos que hicieron attach `shmat` deben hacer detach `shmdt`

- El sistema mantiene la información de cuantos procesos están *attacheados* al segmento de memoria dentro de la variable `shm_nattch` dentro de la estructura
 - ▲ `shmat` incrementa en uno el valor de `shm_nattch`
 - ▼ `shmdt` decrementa en uno el valor de `shm_nattch`

Eliminar

shmctl

Eliminar shmctl

```
int shmctl (int shmid, int cmd, struct shmid_ds* buff);
```

- `int shmid`:
Identificador del segmento de memoria obtenido con `shmget`
- `int cmd`: `IPC_RMID`
IPC Remove ID
- `struct shmid_ds*`: `NULL`

Eliminar shmctl

```
int shmctl (int shmid, int cmd, struct shmid_ds* buff);
```

- Devuelve `0` en caso de éxito
 - Si falla, devuelve `-1`, y setea la variable externa `errno`

```
int result = shmctl ( mem_id, IPC_RMID, NULL );  
if (result == -1){  
    perror("Error: ");  
}
```

Eliminar shmctl

```
int shmctl (int shmid, int cmd, struct shmid_ds* buff);
```

- El proceso de liberación de la memoria solo se da cuando shm_nattch es igual a 0
- Si no se ejecuta explícitamente una operación de destrucción, la memoria compartida nunca se libera

Control `shmctl`

```
int shmctl (int shmid, int cmd, struct shmid_ds* buff);
```

En rigor de la verdad, `shmctl` es una función de control con más de una utilidad. Por ejemplo puede obtener el número de procesos *attacheados* a la memoria

```
shmid_ds status ;  
shmctl ( mem_id , IPC_STAT , &status );  
printf( "Cantidad de procesos 'attacheados': %d", status.shm_nattch);
```


Comandos útiles

Comandos útiles `ipcs`

Lista información de IPC, entre ella esta la lista de las memorias compartidas creadas

```
hige@Hige-Debian:~$ ipcs
```

```
----- Colas de mensajes -----
```

key	msqid	propietario	perms	bytes	utilizados	mensajes
-----	-------	-------------	-------	-------	------------	----------

```
---- Segmentos memoria compartida ----
```

key	shmid	propietario	perms	bytes	nattch	estado
0x00000	294912	hige	600	524288	2	dest

```
----- Matrices semáforo -----
```

key	semid	propietario	perms	nsems
-----	-------	-------------	-------	-------

Comandos útiles `ipcrm`

Permite eliminar un recurso de IPC

- Remover por key

```
hige@Hige-Debian:~$ ipcrm -M <mem_key>
```

- Remover por id

```
hige@Hige-Debian:~$ ipcrm -m <mem_id>
```

Recap

Recap (cheatsheet)

- `shmget`: shared memory get
- `shmat`: shared memory attach
- `shmdt`: shared memory detach
- `shmctl`: shared memory control
- `ftok`: file to key


Pro-Tip

**Las operaciones comparten todas páginas de manual
(salvo `ftok`, que no es exclusivo de memoria compartida)
por lo que con solo recordar una se puede hallar el resto**

Recap (Operaciones)

Operacion	Archivos	Memoria Compartida	Memoria Dinámica
Crear	<code>open</code>	<code>shmget</code>	<code>malloc</code>
Abrir	<code>open</code>	<code>shmat</code>	<code>malloc</code>
Operar	<code>read/write</code>	punteros	punteros
Cerrar	<code>close</code>	<code>shmdt</code>	-
Eliminar	<code>remove</code>	<code>shmctl</code>	<code>free()</code>

Bibliografía

-  *The Design of the Unix Operating System*, Maurice Bach
-  Manuales del sistema operativo