Semáforos

Eduardo C. Garrido Merchán

Sistemas Operativos. Práctica 3. Semana 2.

Motivación

- Si queremos que dos procesos modifiquen una zona de memoria compartida pueden sobreescribir los datos.
- Y no podemos controlar que solo uno escriba a la vez.
- Esto haría que se pueda perder información o que otro proceso malinterprete información que debe leer.
- Lo mismo puede suceder a la hora de lectura/escritura de ficheros, bases de datos...
- Existen ciertas situaciones en las que solo querremos que uno o un número de procesos ejecute instrucciones sobre uno o varios recursos.
- Para lograr este fin, se usan mecanismos de sincronización entre procesos, como por ejemplo: **Semáforos**.

Motivación

- Si queremos que dos procesos modifiquen una zona de memoria compartida pueden sobreescribir los datos.
- Y no podemos controlar que solo uno escriba a la vez.
- Esto haría que se pueda perder información o que otro proceso malinterprete información que debe leer.
- Lo mismo puede suceder a la hora de lectura/escritura de ficheros, bases de datos...
- Existen ciertas situaciones en las que solo querremos que uno o un número de procesos ejecute instrucciones sobre uno o varios recursos.
- Para lograr este fin, se usan mecanismos de sincronización entre procesos, como por ejemplo: **Semáforos**.

Motivación

- Si queremos que dos procesos modifiquen una zona de memoria compartida pueden sobreescribir los datos.
- Y no podemos controlar que solo uno escriba a la vez.
- Esto haría que se pueda perder información o que otro proceso malinterprete información que debe leer.
- Lo mismo puede suceder a la hora de lectura/escritura de ficheros, bases de datos...
- Existen ciertas situaciones en las que solo querremos que uno o un número de procesos ejecute instrucciones sobre uno o varios recursos.
- Para lograr este fin, se usan mecanismos de sincronización entre procesos, como por ejemplo: **Semáforos**.

Motivación

- Si queremos que dos procesos modifiquen una zona de memoria compartida pueden sobreescribir los datos.
- Y no podemos controlar que solo uno escriba a la vez.
- Esto haría que se pueda perder información o que otro proceso malinterprete información que debe leer.
- Lo mismo puede suceder a la hora de lectura/escritura de ficheros, bases de datos...
- Existen ciertas situaciones en las que solo querremos que uno o un número de procesos ejecute instrucciones sobre uno o varios recursos.
- Para lograr este fin, se usan mecanismos de sincronización entre procesos, como por ejemplo: **Semáforos**.

Motivación

- Si queremos que dos procesos modifiquen una zona de memoria compartida pueden sobreescribir los datos.
- Y no podemos controlar que solo uno escriba a la vez.
- Esto haría que se pueda perder información o que otro proceso malinterprete información que debe leer.
- Lo mismo puede suceder a la hora de lectura/escritura de ficheros, bases de datos...
- Existen ciertas situaciones en las que solo querremos que uno o un número de procesos ejecute instrucciones sobre uno o varios recursos.
- Para lograr este fin, se usan mecanismos de sincronización entre procesos, como por ejemplo: **Semáforos**.

Motivación

- Si queremos que dos procesos modifiquen una zona de memoria compartida pueden sobreescribir los datos.
- Y no podemos controlar que solo uno escriba a la vez.
- Esto haría que se pueda perder información o que otro proceso malinterprete información que debe leer.
- Lo mismo puede suceder a la hora de lectura/escritura de ficheros, bases de datos...
- Existen ciertas situaciones en las que solo querremos que uno o un número de procesos ejecute instrucciones sobre uno o varios recursos.
- Para lograr este fin, se usan mecanismos de sincronización entre procesos, como por ejemplo: **Semáforos**.

Definición

- Un semáforo es un mecanismo de sincronización del sistema operativo.
- Eliminan los riesgos del acceso concurrente a recursos compartidos, como la memoria compartida.
- Se comportan como variables enteras que tienen asociadas operaciones atómicas.
- La serie de instrucciones a proteger por el semáforo es conocida como **sección crítica**.

Definición

- Un semáforo es un mecanismo de sincronización del sistema operativo.
- Eliminan los riesgos del acceso concurrente a recursos compartidos, como la memoria compartida.
- Se comportan como variables enteras que tienen asociadas operaciones atómicas.
- La serie de instrucciones a proteger por el semáforo es conocida como **sección crítica**.

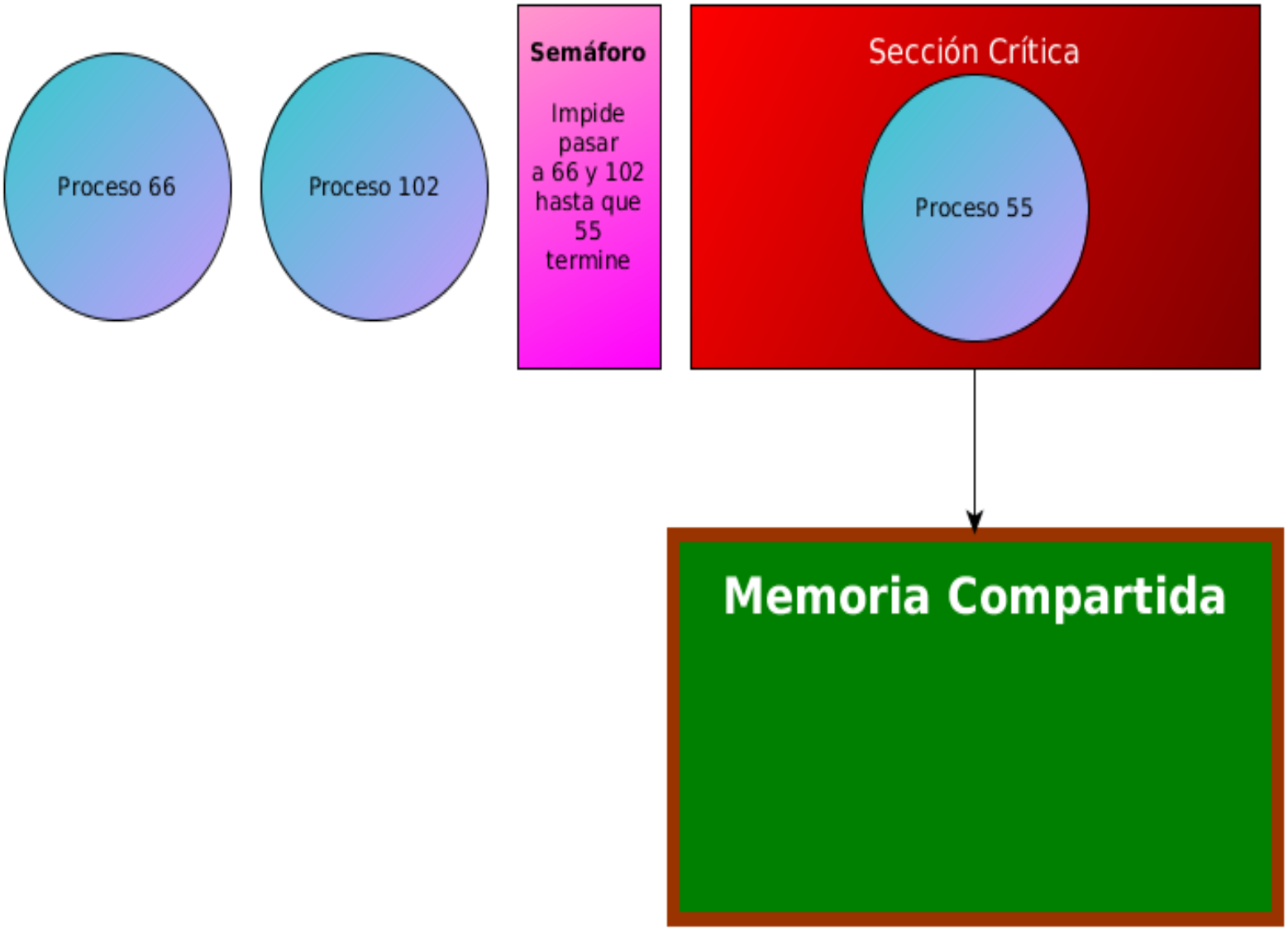
Definición

- Un semáforo es un mecanismo de sincronización del sistema operativo.
- Eliminan los riesgos del acceso concurrente a recursos compartidos, como la memoria compartida.
- Se comportan como variables enteras que tienen asociadas operaciones atómicas.
- La serie de instrucciones a proteger por el semáforo es conocida como **sección crítica**.

Definición

- Un semáforo es un mecanismo de sincronización del sistema operativo.
- Eliminan los riesgos del acceso concurrente a recursos compartidos, como la memoria compartida.
- Se comportan como variables enteras que tienen asociadas operaciones atómicas.
- La serie de instrucciones a proteger por el semáforo es conocida como **sección crítica**.

Descripción gráfica de un semáforo



Tipos de semáforos

- **Binarios**: Solo puede valer 0 o 1. 0 impide el paso del proceso a la sección crítica, 1 lo permite. Se pone a 0 cuando permite el acceso a un proceso, a 1 cuando sale.
- Usados para gestionar la escritura y lectura de procesos en memoria compartida.
- **N-arios**: Toma valores de 0 a N, permitiendo el paso a N procesos como máximo. Cada uno que entra resta uno y cuando salen suma uno.
- Usados para gestionar la lectura de memoria compartida.

Tipos de semáforos

- **Binarios**: Solo puede valer 0 o 1. 0 impide el paso del proceso a la sección crítica, 1 lo permite. Se pone a 0 cuando permite el acceso a un proceso, a 1 cuando sale.
- Usados para gestionar la escritura y lectura de procesos en memoria compartida.
- **N-arios**: Toma valores de 0 a N, permitiendo el paso a N procesos como máximo. Cada uno que entra resta uno y cuando salen suma uno.
- Usados para gestionar la lectura de memoria compartida.

Tipos de semáforos

- **Binarios**: Solo puede valer 0 o 1. 0 impide el paso del proceso a la sección crítica, 1 lo permite. Se pone a 0 cuando permite el acceso a un proceso, a 1 cuando sale.
- Usados para gestionar la escritura y lectura de procesos en memoria compartida.
- **N-arios**: Toma valores de 0 a N, permitiendo el paso a N procesos como máximo. Cada uno que entra resta uno y cuando salen suma uno.
- Usados para gestionar la lectura de memoria compartida.

Tipos de semáforos

- **Binarios**: Solo puede valer 0 o 1. 0 impide el paso del proceso a la sección crítica, 1 lo permite. Se pone a 0 cuando permite el acceso a un proceso, a 1 cuando sale.
- Usados para gestionar la escritura y lectura de procesos en memoria compartida.
- **N-arios**: Toma valores de 0 a N, permitiendo el paso a N procesos como máximo. Cada uno que entra resta uno y cuando salen suma uno.
- Usados para gestionar la lectura de memoria compartida.

Operaciones realizadas por los semáforos

- Son dos, y son atómicas, es decir, no pueden ser interrumpidas por otros procesos. Son:
- **Down**: Petición del semáforo al SO de que un proceso quiere entrar en la sección crítica. Si esta a 1 decrece a 0 y el proceso entra. Sino, el proceso se duerme hasta que el semáforo valga 1.
- **Up**: Incremento del valor en 1 del semáforo dado que un proceso sale de la sección crítica. Si un proceso estaba esperando, se despertará, entrará y se hará un Down.

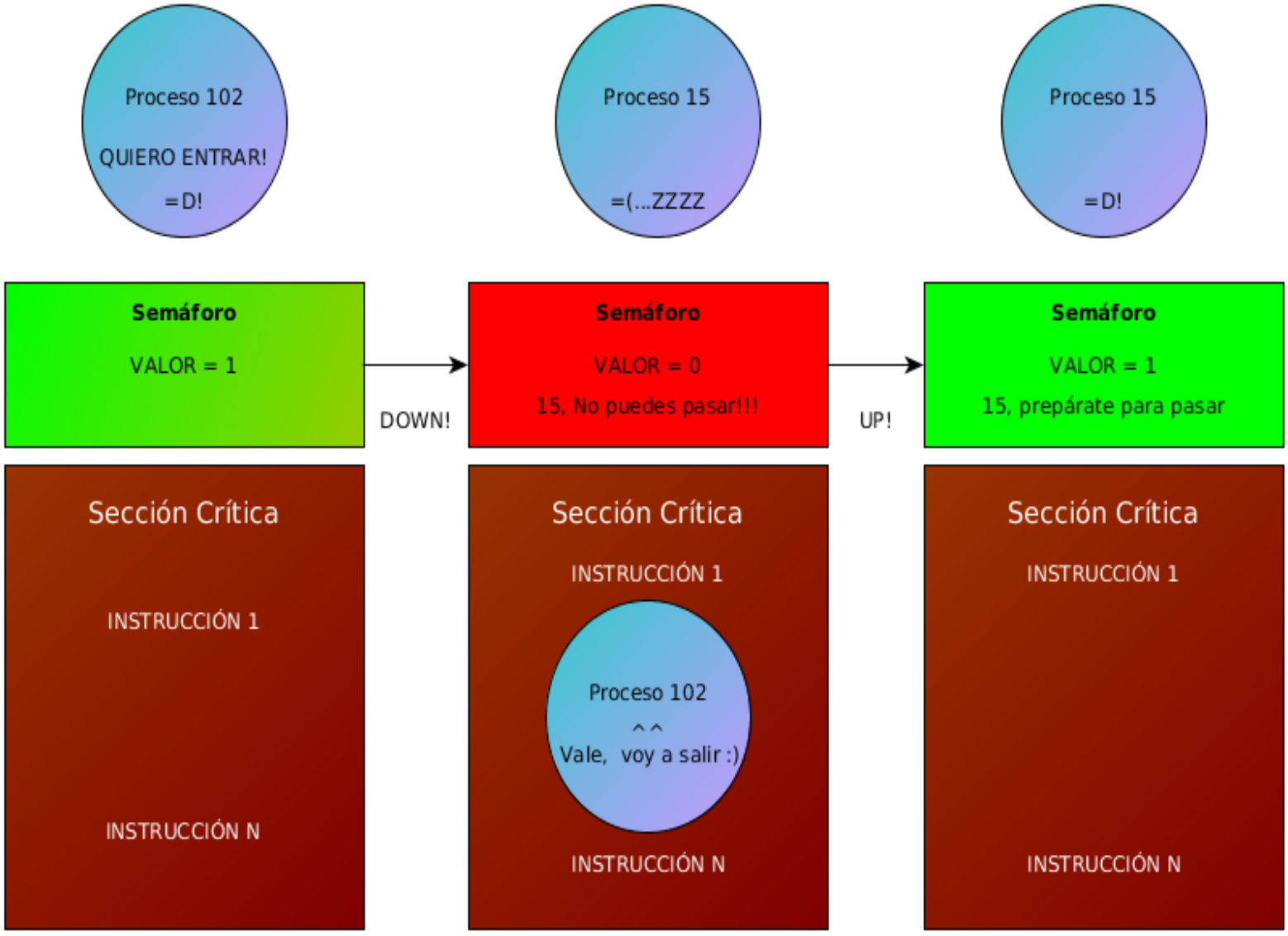
Operaciones realizadas por los semáforos

- Son dos, y son atómicas, es decir, no pueden ser interrumpidas por otros procesos. Son:
- **Down**: Petición del semáforo al SO de que un proceso quiere entrar en la sección crítica. Si esta a 1 decrece a 0 y el proceso entra. Sino, el proceso se duerme hasta que el semáforo valga 1.
- **Up**: Incremento del valor en 1 del semáforo dado que un proceso sale de la sección crítica. Si un proceso estaba esperando, se despertará, entrará y se hará un Down.

Operaciones realizadas por los semáforos

- Son dos, y son atómicas, es decir, no pueden ser interrumpidas por otros procesos. Son:
- **Down**: Petición del semáforo al SO de que un proceso quiere entrar en la sección crítica. Si esta a 1 decrece a 0 y el proceso entra. Sino, el proceso se duerme hasta que el semáforo valga 1.
- **Up**: Incremento del valor en 1 del semáforo dado que un proceso sale de la sección crítica. Si un proceso estaba esperando, se despertará, entrará y se hará un Down.

Descripción gráfica de las operaciones de un semáforo



Programación en C de semáforos

- Las funciones necesarias se incluyen en *sys/ipc.h*, *sys/sem.h* y *sys/types.h*.
- *semget*: Crea semáforos o los localiza.
- *semop*: Realiza operaciones con los semáforos.
- *semctl*: Gestión y liberación de semáforos.
- Son funciones potentes pero complejas, por lo que se recomienda simplicidad con las mismas.

Programación en C de semáforos

- Las funciones necesarias se incluyen en *sys/ipc.h*, *sys/sem.h* y *sys/types.h*.
- *semget*: Crea semáforos o los localiza.
- *semop*: Realiza operaciones con los semáforos.
- *semctl*: Gestión y liberación de semáforos.
- Son funciones potentes pero complejas, por lo que se recomienda simplicidad con las mismas.

Programación en C de semáforos

- Las funciones necesarias se incluyen en *sys/ipc.h*, *sys/sem.h* y *sys/types.h*.
- *semget*: Crea semáforos o los localiza.
- *semop*: Realiza operaciones con los semáforos.
- *semctl*: Gestión y liberación de semáforos.
- Son funciones potentes pero complejas, por lo que se recomienda simplicidad con las mismas.

Programación en C de semáforos

- Las funciones necesarias se incluyen en *sys/ipc.h*, *sys/sem.h* y *sys/types.h*.
- *semget*: Crea semáforos o los localiza.
- *semop*: Realiza operaciones con los semáforos.
- *semctl*: Gestión y liberación de semáforos.
- Son funciones potentes pero complejas, por lo que se recomienda simplicidad con las mismas.

Programación en C de semáforos

- Las funciones necesarias se incluyen en *sys/ipc.h*, *sys/sem.h* y *sys/types.h*.
- *semget*: Crea semáforos o los localiza.
- *semop*: Realiza operaciones con los semáforos.
- *semctl*: Gestión y liberación de semáforos.
- Son funciones potentes pero complejas, por lo que se recomienda simplicidad con las mismas.

Creación de semáforos: *semget*

- Prototipo: *int semget(key_t key, int nsems, int semflg)*.
- Devuelve un array (ojo) de semáforos del tamaño marcado por *nsems*. Si nsems=1, entonces será un array de un semáforo.
- Habrá que crear una clave con *ftok* como en memoria compartida.
- El último campo son flags, para crear un semáforo un proceso marcará IPC_CREAT. El resto que lo usen pondrá un cero.
- Si el semáforo marcado por key no existe, se crea, si existe, se localiza y se devuelve.
- Si solo 1 proceso e hijos usa el proceso, entonces como primer parámetro no se necesita una *key* y se marcará IPC_PRIVATE.
- Crear no es inicializar. El programador deberá inicializar después de crear.

Creación de semáforos: *semget*

- Prototipo: *int semget(key_t key, int nsems, int semflg)*.
- Devuelve un array (ojo) de semáforos del tamaño marcado por *nsems*. Si nsems=1, entonces será un array de un semáforo.
- Habrá que crear una clave con *ftok* como en memoria compartida.
- El último campo son flags, para crear un semáforo un proceso marcará IPC_CREAT. El resto que lo usen pondrá un cero.
- Si el semáforo marcado por key no existe, se crea, si existe, se localiza y se devuelve.
- Si solo 1 proceso e hijos usa el proceso, entonces como primer parámetro no se necesita una *key* y se marcará IPC_PRIVATE.
- Crear no es inicializar. El programador deberá inicializar después de crear.

Creación de semáforos: *semget*

- Prototipo: *int semget(key_t key, int nsems, int semflg)*.
- Devuelve un array (ojo) de semáforos del tamaño marcado por *nsems*. Si nsems=1, entonces será un array de un semáforo.
- Habrá que crear una clave con *ftok* como en memoria compartida.
- El último campo son flags, para crear un semáforo un proceso marcará IPC_CREAT. El resto que lo usen pondrá un cero.
- Si el semáforo marcado por key no existe, se crea, si existe, se localiza y se devuelve.
- Si solo 1 proceso e hijos usa el proceso, entonces como primer parámetro no se necesita una *key* y se marcará IPC_PRIVATE.
- Crear no es inicializar. El programador deberá inicializar después de crear.

Creación de semáforos: *semget*

- Prototipo: *int semget(key_t key, int nsems, int semflg)*.
- Devuelve un array (ojo) de semáforos del tamaño marcado por *nsems*. Si nsems=1, entonces será un array de un semáforo.
- Habrá que crear una clave con *ftok* como en memoria compartida.
- El último campo son flags, para crear un semáforo un proceso marcará IPC_CREAT. El resto que lo usen pondrá un cero.
- Si el semáforo marcado por key no existe, se crea, si existe, se localiza y se devuelve.
- Si solo 1 proceso e hijos usa el proceso, entonces como primer parámetro no se necesita una *key* y se marcará IPC_PRIVATE.
- Crear no es inicializar. El programador deberá inicializar después de crear.

Creación de semáforos: *semget*

- Prototipo: *int semget(key_t key, int nsems, int semflg)*.
- Devuelve un array (ojo) de semáforos del tamaño marcado por *nsems*. Si nsems=1, entonces será un array de un semáforo.
- Habrá que crear una clave con *ftok* como en memoria compartida.
- El último campo son flags, para crear un semáforo un proceso marcará IPC_CREAT. El resto que lo usen pondrá un cero.
- Si el semáforo marcado por key no existe, se crea, si existe, se localiza y se devuelve.
- Si solo 1 proceso e hijos usa el proceso, entonces como primer parámetro no se necesita una *key* y se marcará IPC_PRIVATE.
- Crear no es inicializar. El programador deberá inicializar después de crear.

Creación de semáforos: *semget*

- Prototipo: *int semget(key_t key, int nsems, int semflg)*.
- Devuelve un array (ojo) de semáforos del tamaño marcado por *nsems*. Si nsems=1, entonces será un array de un semáforo.
- Habrá que crear una clave con *ftok* como en memoria compartida.
- El último campo son flags, para crear un semáforo un proceso marcará IPC_CREAT. El resto que lo usen pondrá un cero.
- Si el semáforo marcado por key no existe, se crea, si existe, se localiza y se devuelve.
- Si solo 1 proceso e hijos usa el proceso, entonces como primer parámetro no se necesita una *key* y se marcará IPC_PRIVATE.
- Crear no es inicializar. El programador deberá inicializar después de crear.

Creación de semáforos: *semget*

- Prototipo: *int semget(key_t key, int nsems, int semflg)*.
- Devuelve un array (ojo) de semáforos del tamaño marcado por *nsems*. Si nsems=1, entonces será un array de un semáforo.
- Habrá que crear una clave con *ftok* como en memoria compartida.
- El último campo son flags, para crear un semáforo un proceso marcará IPC_CREAT. El resto que lo usen pondrá un cero.
- Si el semáforo marcado por key no existe, se crea, si existe, se localiza y se devuelve.
- Si solo 1 proceso e hijos usa el proceso, entonces como primer parámetro no se necesita una *key* y se marcará IPC_PRIVATE.
- Crear no es inicializar. El programador deberá inicializar después de crear.

Ejemplo

```
/*
 * Creamos una lista o conjunto con dos semáforos
 */

semid = semget(SEMKEY, N_SEMAFOROS,
                IPC_CREAT | IPC_EXCL | SHM_R | SHM_W);
if((semid == -1) && (errno == EEXIST))
     semid=semget(SEMKEY,N_SEMAFOROS,SHM_R|SHM_W);
 if(semid==-1){
     perror("semget");
     exit(errno);
 }
```

Operaciones con semáforos: *semop*

- Prototipo: *int semop(int semid, struct sembuf *sops, unsigned int nsops)*.
- Permite realizar operaciones de incremento (up) y decremento (down).
- *semid*: Corresponde al identificador de semáforos asociados.
- *sops*: Estructura *sembuf* con campos *sem_num*, número del semáforo, *sem_op*, entero que incrementa (up) o decrementa (down) y *sem_flg*, máscara de bits: IPC_WAIT: Bloquea proceso. IPC_NOWAIT: Si *sem_op* falla, devuelve control. SEM_UNDO: Si un proceso falla en sección crítica, el SO reestablecerá el semáforo.
- 0 es semáforo bloqueando la sección crítica, el estado actual - *sem_op* no puede ser menor que cero.
- *nsops*: Total de elementos que tiene el array de operaciones.

Operaciones con semáforos: *semop*

- Prototipo: *int semop(int semid, struct sembuf *sops, unsigned int nsops)*.
- Permite realizar operaciones de incremento (up) y decremento (down).
- *semid*: Corresponde al identificador de semáforos asociados.
- *sops*: Estructura *sembuf* con campos *sem_num*, número del semáforo, *sem_op*, entero que incrementa (up) o decrementa (down) y *sem_flg*, máscara de bits: IPC_WAIT: Bloquea proceso. IPC_NOWAIT: Si *sem_op* falla, devuelve control. SEM_UNDO: Si un proceso falla en sección crítica, el SO reestablecerá el semáforo.
- 0 es semáforo bloqueando la sección crítica, el estado actual - *sem_op* no puede ser menor que cero.
- *nsops*: Total de elementos que tiene el array de operaciones.

Operaciones con semáforos: *semop*

- Prototipo: *int semop(int semid, struct sembuf *sops, unsigned int nsops)*.
- Permite realizar operaciones de incremento (up) y decremento (down).
- *semid*: Corresponde al identificador de semáforos asociados.
- *sops*: Estructura *sembuf* con campos *sem_num*, número del semáforo, *sem_op*, entero que incrementa (up) o decrementa (down) y *sem_flg*, máscara de bits: IPC_WAIT: Bloquea proceso. IPC_NOWAIT: Si *sem_op* falla, devuelve control. SEM_UNDO: Si un proceso falla en sección crítica, el SO reestablecerá el semáforo.
- 0 es semáforo bloqueando la sección crítica, el estado actual - *sem_op* no puede ser menor que cero.
- *nsops*: Total de elementos que tiene el array de operaciones.

Operaciones con semáforos: *semop*

- Prototipo: *int semop(int semid, struct sembuf *sops, unsigned int nsops)*.
- Permite realizar operaciones de incremento (up) y decremento (down).
- *semid*: Corresponde al identificador de semáforos asociados.
- *sops*: Estructura *sembuf* con campos *sem_num*, número del semáforo, *sem_op*, entero que incrementa (up) o decrementa (down) y *sem_flg*, máscara de bits: IPC_WAIT: Bloquea proceso. IPC_NOWAIT: Si *sem_op* falla, devuelve control. SEM_UNDO: Si un proceso falla en sección crítica, el SO reestablecerá el semáforo.
- 0 es semáforo bloqueando la sección crítica, el estado actual - *sem_op* no puede ser menor que cero.
- *nsops*: Total de elementos que tiene el array de operaciones.

Operaciones con semáforos: *semop*

- Prototipo: *int semop(int semid, struct sembuf *sops, unsigned int nsops)*.
- Permite realizar operaciones de incremento (up) y decremento (down).
- *semid*: Corresponde al identificador de semáforos asociados.
- *sops*: Estructura *sembuf* con campos *sem_num*, número del semáforo, *sem_op*, entero que incrementa (up) o decrementa (down) y *sem_flg*, máscara de bits: IPC_WAIT: Bloquea proceso. IPC_NOWAIT: Si *sem_op* falla, devuelve control. SEM_UNDO: Si un proceso falla en sección crítica, el SO reestablecerá el semáforo.
- 0 es semáforo bloqueando la sección crítica, el estado actual - *sem_op* no puede ser menor que cero.
- *nsops*: Total de elementos que tiene el array de operaciones.

Operaciones con semáforos: *semop*

- Prototipo: *int semop(int semid, struct sembuf *sops, unsigned int nsops)*.
- Permite realizar operaciones de incremento (up) y decremento (down).
- *semid*: Corresponde al identificador de semáforos asociados.
- *sops*: Estructura *sembuf* con campos *sem_num*, número del semáforo, *sem_op*, entero que incrementa (up) o decrementa (down) y *sem_flg*, máscara de bits: IPC_WAIT: Bloquea proceso. IPC_NOWAIT: Si *sem_op* falla, devuelve control. SEM_UNDO: Si un proceso falla en sección crítica, el SO reestablecerá el semáforo.
- 0 es semáforo bloqueando la sección crítica, el estado actual - *sem_op* no puede ser menor que cero.
- *nsops*: Total de elementos que tiene el array de operaciones.

Ejemplo

```
/*
 * Operamos sobre los semáforos
 */

sem_oper.sem_num = 0; /* Actuamos sobre el semáforo 0 de la lista */
sem_oper.sem_op =-1; /* Decrementar en 1 el valor del semáforo */
sem_oper.sem_flg = SEM_UNDO; /* Para evitar interbloqueos si un
proceso acaba inesperadamente */

semop (semid, &sem_oper, 1);


sem_oper.sem_num = 1; /* Actuamos sobre el semáforo 1 de la lista */
sem_oper.sem_op = 1; /* Incrementar en 1 el valor del semáforo */
sem_oper.sem_flg = SEM_UNDO; /* No es necesario porque ya se ha
   hecho anteriormente */

semop (semid, &sem_oper, 1);
```

Gestión y liberación de semáforos: *semctl*

- Prototipo: *int semctl(int semid, int semnum, int cmd, union semun arg)*.
- Gestión de todas las operaciones excepto up y down.
- *semid*: Corresponde al identificador de semáforos asociados.
- *semnum*: Índice del semáforo sobre el que queremos trabajar.
- *cmd*: Operación a realizar con el semáforo.
- *semun*: Estructura a incluir explicitamente en el programa. Campos: *int val*, valor del semáforo. *struct semid_ds *semstat*, opciones avanzadas y *unsigned short *array*: Número de semáforos a usar, deben ser inicializados al número de procesos que controlan (binarios : 1).

Gestión y liberación de semáforos: *semctl*

- Prototipo: *int semctl(int semid, int semnum, int cmd, union semun arg)*.
- Gestión de todas las operaciones excepto up y down.
- *semid*: Corresponde al identificador de semáforos asociados.
- *semnum*: Índice del semáforo sobre el que queremos trabajar.
- *cmd*: Operación a realizar con el semáforo.
- *semun*: Estructura a incluir explicitamente en el programa. Campos: *int val*, valor del semáforo. *struct semid_ds *semstat*, opciones avanzadas y *unsigned short *array*: Número de semáforos a usar, deben ser inicializados al número de procesos que controlan (binarios : 1).

Gestión y liberación de semáforos: *semctl*

- Prototipo: *int semctl(int semid, int semnum, int cmd, union semun arg)*.
- Gestión de todas las operaciones excepto up y down.
- *semid*: Corresponde al identificador de semáforos asociados.
- *semnum*: Índice del semáforo sobre el que queremos trabajar.
- *cmd*: Operación a realizar con el semáforo.
- *semun*: Estructura a incluir explicitamente en el programa. Campos: *int val*, valor del semáforo. *struct semid_ds *semstat*, opciones avanzadas y *unsigned short *array*: Número de semáforos a usar, deben ser inicializados al número de procesos que controlan (binarios : 1).

Gestión y liberación de semáforos: *semctl*

- Prototipo: *int semctl(int semid, int semnum, int cmd, union semun arg)*.
- Gestión de todas las operaciones excepto up y down.
- *semid*: Corresponde al identificador de semáforos asociados.
- *semnum*: Índice del semáforo sobre el que queremos trabajar.
- *cmd*: Operación a realizar con el semáforo.
- *semun*: Estructura a incluir explicitamente en el programa. Campos: *int val*, valor del semáforo. *struct semid_ds *semstat*, opciones avanzadas y *unsigned short *array*: Número de semáforos a usar, deben ser inicializados al número de procesos que controlan (binarios : 1).

Gestión y liberación de semáforos: *semctl*

- Prototipo: *int semctl(int semid, int semnum, int cmd, union semun arg)*.
- Gestión de todas las operaciones excepto up y down.
- *semid*: Corresponde al identificador de semáforos asociados.
- *semnum*: Índice del semáforo sobre el que queremos trabajar.
- *cmd*: Operación a realizar con el semáforo.
- *semun*: Estructura a incluir explicitamente en el programa. Campos: *int val*, valor del semáforo. *struct semid_ds *semstat*, opciones avanzadas y *unsigned short *array*: Número de semáforos a usar, deben ser inicializados al número de procesos que controlan (binarios : 1).

Gestión y liberación de semáforos: *semctl*

- Prototipo: *int semctl(int semid, int semnum, int cmd, union semun arg)*.
- Gestión de todas las operaciones excepto up y down.
- *semid*: Corresponde al identificador de semáforos asociados.
- *semnum*: Índice del semáforo sobre el que queremos trabajar.
- *cmd*: Operación a realizar con el semáforo.
- *semun*: Estructura a incluir explicitamente en el programa. Campos: *int val*, valor del semáforo. *struct semid_ds *semstat*, opciones avanzadas y *unsigned short *array*: Número de semáforos a usar, deben ser inicializados al número de procesos que controlan (binarios : 1).

Ejemplo de inicialización del semáforo

```
union semun {
   int val;
   struct semid_ds *semstat;
   unsigned short *array;
} arg;

/*
 * Inicializamos los semáforos
 */

arg.array = (unsigned short *)malloc(sizeof(short)*N_SEMAFOROS);

arg.array [0] = arg.array [1] = 1;

semctl (semid, N_SEMAFOROS, SETALL, arg);
```

Operaciones realizables por *semctl*: Valores de *cmd*

- IPC_RMID: Elimina el array de semáforos.
- GETVAL: Devuelve el valor actual del semáforo.
- SETVAL: Da un valor al semáforo.
  semctl(semaforo,3,SETVAL,2).
- GETALL: Lee el valor de todos los semáforos.
- SETALL: Inicia el valor de todos los semáforos.
- GETPID: Devuelve el PID del último proceso que actúo sobre el semáforo.
- GETNCNT: Devuelve el número de procesos bloqueados en la cola del semáforo.

Operaciones realizables por *semctl*: Valores de *cmd*

- IPC_RMID: Elimina el array de semáforos.
- GETVAL: Devuelve el valor actual del semáforo.
- SETVAL: Da un valor al semáforo.
  semctl(semaforo,3,SETVAL,2).
- GETALL: Lee el valor de todos los semáforos.
- SETALL: Inicia el valor de todos los semáforos.
- GETPID: Devuelve el PID del último proceso que actúo sobre el semáforo.
- GETNCNT: Devuelve el número de procesos bloqueados en la cola del semáforo.

Operaciones realizables por *semctl*: Valores de *cmd*

- IPC_RMID: Elimina el array de semáforos.
- GETVAL: Devuelve el valor actual del semáforo.
- SETVAL: Da un valor al semáforo.
  semctl(semaforo,3,SETVAL,2).
- GETALL: Lee el valor de todos los semáforos.
- SETALL: Inicia el valor de todos los semáforos.
- GETPID: Devuelve el PID del último proceso que actúo sobre el semáforo.
- GETNCNT: Devuelve el número de procesos bloqueados en la cola del semáforo.

Operaciones realizables por *semctl*: Valores de *cmd*

- IPC_RMID: Elimina el array de semáforos.
- GETVAL: Devuelve el valor actual del semáforo.
- SETVAL: Da un valor al semáforo.
  semctl(semaforo,3,SETVAL,2).
- GETALL: Lee el valor de todos los semáforos.
- SETALL: Inicia el valor de todos los semáforos.
- GETPID: Devuelve el PID del último proceso que actúo sobre el semáforo.
- GETNCNT: Devuelve el número de procesos bloqueados en la cola del semáforo.

Operaciones realizables por *semctl*: Valores de *cmd*

- IPC_RMID: Elimina el array de semáforos.
- GETVAL: Devuelve el valor actual del semáforo.
- SETVAL: Da un valor al semáforo.
  semctl(semaforo,3,SETVAL,2).
- GETALL: Lee el valor de todos los semáforos.
- SETALL: Inicia el valor de todos los semáforos.
- GETPID: Devuelve el PID del último proceso que actúo sobre el semáforo.
- GETNCNT: Devuelve el número de procesos bloqueados en la cola del semáforo.

Operaciones realizables por *semctl*: Valores de *cmd*

- IPC_RMID: Elimina el array de semáforos.
- GETVAL: Devuelve el valor actual del semáforo.
- SETVAL: Da un valor al semáforo.
  semctl(semaforo,3,SETVAL,2).
- GETALL: Lee el valor de todos los semáforos.
- SETALL: Inicia el valor de todos los semáforos.
- GETPID: Devuelve el PID del último proceso que actúo sobre el semáforo.
- GETNCNT: Devuelve el número de procesos bloqueados en la cola del semáforo.

Operaciones realizables por *semctl*: Valores de *cmd*

- IPC_RMID: Elimina el array de semáforos.
- GETVAL: Devuelve el valor actual del semáforo.
- SETVAL: Da un valor al semáforo.
  semctl(semaforo,3,SETVAL,2).
- GETALL: Lee el valor de todos los semáforos.
- SETALL: Inicia el valor de todos los semáforos.
- GETPID: Devuelve el PID del último proceso que actúo sobre el semáforo.
- GETNCNT: Devuelve el número de procesos bloqueados en la cola del semáforo.

Ejemplo de consultar valores de semáforos

```
/*
 * Veamos los valores de los semáforos
 */

 semctl (semid, N_SEMAFOROS, GETALL, arg);

 printf ("Los valores de los semáforos son %d y %d",
          arg.array [0], arg.array [1]);
```

Ejemplo de eliminar la lista de semáforos

```
/* Eliminar la lista de semáforos */

semctl (semid, N_SEMAFOROS, IPC_RMID, 0);

free(arg.array);
```

Consideraciones finales

- Los conceptos vistos en prácticas son un conjunto limitado de herramientas del SO.
- Hay muchas mas mecanismos de sincronización, librerías, comunicaciones entre procesos...
- Y muchos mas sistemas operativos que implementan conceptos de forma distinta: Windows.
- No obstante, se han presentado los fundamentos de Sistemas Operativos e implementado con C.
- Otros lenguajes también tienen librerías con conceptos vistos.
- Si se tiene interés, recomiendo seguir investigando sobre el tema, hay mucha bibliografía.
- Campos de aplicación: Sistemas de Tiempo Real, Motores de Cálculo, Q/A, Servidores Web...
- Gracias por vuestra atención!

Consideraciones finales

- Los conceptos vistos en prácticas son un conjunto limitado de herramientas del SO.
- Hay muchas mas mecanismos de sincronización, librerías, comunicaciones entre procesos...
- Y muchos mas sistemas operativos que implementan conceptos de forma distinta: Windows.
- No obstante, se han presentado los fundamentos de Sistemas Operativos e implementado con C.
- Otros lenguajes también tienen librerías con conceptos vistos.
- Si se tiene interés, recomiendo seguir investigando sobre el tema, hay mucha bibliografía.
- Campos de aplicación: Sistemas de Tiempo Real, Motores de Cálculo, Q/A, Servidores Web...
- Gracias por vuestra atención!

Consideraciones finales

- Los conceptos vistos en prácticas son un conjunto limitado de herramientas del SO.
- Hay muchas mas mecanismos de sincronización, librerías, comunicaciones entre procesos...
- Y muchos mas sistemas operativos que implementan conceptos de forma distinta: Windows.
- No obstante, se han presentado los fundamentos de Sistemas Operativos e implementado con C.
- Otros lenguajes también tienen librerías con conceptos vistos.
- Si se tiene interés, recomiendo seguir investigando sobre el tema, hay mucha bibliografía.
- Campos de aplicación: Sistemas de Tiempo Real, Motores de Cálculo, Q/A, Servidores Web...
- Gracias por vuestra atención!

Consideraciones finales

- Los conceptos vistos en prácticas son un conjunto limitado de herramientas del SO.
- Hay muchas mas mecanismos de sincronización, librerías, comunicaciones entre procesos...
- Y muchos mas sistemas operativos que implementan conceptos de forma distinta: Windows.
- No obstante, se han presentado los fundamentos de Sistemas Operativos e implementado con C.
- Otros lenguajes también tienen librerías con conceptos vistos.
- Si se tiene interés, recomiendo seguir investigando sobre el tema, hay mucha bibliografía.
- Campos de aplicación: Sistemas de Tiempo Real, Motores de Cálculo, Q/A, Servidores Web...
- Gracias por vuestra atención!

Consideraciones finales

- Los conceptos vistos en prácticas son un conjunto limitado de herramientas del SO.
- Hay muchas mas mecanismos de sincronización, librerías, comunicaciones entre procesos...
- Y muchos mas sistemas operativos que implementan conceptos de forma distinta: Windows.
- No obstante, se han presentado los fundamentos de Sistemas Operativos e implementado con C.
- Otros lenguajes también tienen librerías con conceptos vistos.
- Si se tiene interés, recomiendo seguir investigando sobre el tema, hay mucha bibliografía.
- Campos de aplicación: Sistemas de Tiempo Real, Motores de Cálculo, Q/A, Servidores Web...
- Gracias por vuestra atención!

Consideraciones finales

- Los conceptos vistos en prácticas son un conjunto limitado de herramientas del SO.
- Hay muchas mas mecanismos de sincronización, librerías, comunicaciones entre procesos...
- Y muchos mas sistemas operativos que implementan conceptos de forma distinta: Windows.
- No obstante, se han presentado los fundamentos de Sistemas Operativos e implementado con C.
- Otros lenguajes también tienen librerías con conceptos vistos.
- Si se tiene interés, recomiendo seguir investigando sobre el tema, hay mucha bibliografía.
- Campos de aplicación: Sistemas de Tiempo Real, Motores de Cálculo, Q/A, Servidores Web...
- Gracias por vuestra atención!

Consideraciones finales

- Los conceptos vistos en prácticas son un conjunto limitado de herramientas del SO.
- Hay muchas mas mecanismos de sincronización, librerías, comunicaciones entre procesos...
- Y muchos mas sistemas operativos que implementan conceptos de forma distinta: Windows.
- No obstante, se han presentado los fundamentos de Sistemas Operativos e implementado con C.
- Otros lenguajes también tienen librerías con conceptos vistos.
- Si se tiene interés, recomiendo seguir investigando sobre el tema, hay mucha bibliografía.
- Campos de aplicación: Sistemas de Tiempo Real, Motores de Cálculo, Q/A, Servidores Web...
- Gracias por vuestra atención!

Consideraciones finales

- Los conceptos vistos en prácticas son un conjunto limitado de herramientas del SO.
- Hay muchas mas mecanismos de sincronización, librerías, comunicaciones entre procesos...
- Y muchos mas sistemas operativos que implementan conceptos de forma distinta: Windows.
- No obstante, se han presentado los fundamentos de Sistemas Operativos e implementado con C.
- Otros lenguajes también tienen librerías con conceptos vistos.
- Si se tiene interés, recomiendo seguir investigando sobre el tema, hay mucha bibliografía.
- Campos de aplicación: Sistemas de Tiempo Real, Motores de Cálculo, Q/A, Servidores Web...
- Gracias por vuestra atención!