- SISTEMAS OPERATIVOS -
1º GRADO INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE CÓRDOBA

Práctica I – Procesos y Memoria compartida:

Indispensables:

- En UNIX la única forma de crear procesos es la llamada al sistema fork(). fork() crea un proceso que es una copia exacta del proceso que hace la llamada. Para diferenciar al proceso que hace la llamada a fork(), proceso padre, del proceso creado, proceso hijo, el sistema devuelve el PID del hijo creado al padre y un valor 0 al hijo. Así, la forma normal de usar fork() es:

```
#include
main () {

    ...

    int pid = fork ();

    if (pid) {
        // Proceso padre
    } else {
        // Proceso hijo

      ...

      exit ();
    }
}
```

- Nótese la llamada exit() al final del else, en caso contrario la parte posterior al if sería ejecutada tanto por el padre como por el hijo. El fichero process.c muestra un ejemplo de uso de la llamada fork().

- Cuando se quiere compartir información entre diferentes procesos, uno de los mecanismos que se pueden usar es el de la memoria compartida. La memoria compartida se usa de forma similar a cualquier memoria dinámica, con la diferencia de que la reserva se realiza sólo una vez y los diferentes procesos acceden a la misma memoria.
- Para que esta memoria se puede compartir entre procesos, el sistema la identifica con una clave unívoca dentro del sistema. Cualquier proceso que use dicha clave podrá acceder a la memoria compartida. Para evitar accesos de usuarios no autorizados, el sistema asigna a la memoria compartida los nueve permisos rwxrwxrwx que se usan para los ficheros.
- Para asignar la clave a la memoria que se va a reservar se usa la llamada ftok(). Esta llamada necesita un archivo o directorio al que puedan acceder todos los procesos que vayan a usar la memoria compartida. A partir de este nombre de archivo construye una clave que se garantiza que es única e igual para todas las llamadas con el mismo fichero. Para poder usar el mismo archivo para diferentes claves la llamada recibe un segundo argumento entero que permite crear una clave diferente para cada valor. Un ejemplo sería:

```
key_t key = ftok("shmemory.c", 1);
```

- Cualquier proceso que ejecute esta llamada recibirá la misma clave que será única en el sistema.

- Una vez obtenida la clave, es necesario proceder a obtener la memoria compartida. Para ello se usa una llamada, shmget(), que resulta similar a malloc(), con las particularidades necesarias debido a que la memoria es compartida. Un ejemplo de esta llamada sería:

```
int shmid = shmget(key, sizeof(double), IPC_CREAT | 0700)
```

- El primer argumento es la clave obtenida en la llamada a ftok(). Todos los procesos que deseen compartir la memoria deberán usar la misma clave. El segundo argumento es el tamaño de la memoria a reservar. El tercer argumento es un campo de bits que contiene información referente a la memoria a crear. Los bits menos significativos contienen información referente a los permisos de la memoria, en este caso usamos los permisos 0700, que corresponden a rwx------. El resto de bits representan opciones que se pueden usar y afectan a cómo se crea la memoria. En principio usaremos solamente la opción IPC_CREAT. Esta opción indica que si la memoria asociada a la clave key aún no ha sido creada se creará, en caso contrario se usará la ya creada y se devolverá su identificador. El valor devuelto por shmget(), shmid, nos servirá para referirnos a la zona de memoria compartida en posteriores llamadas al sistema. Si existe algún error el identificador devuelto es igual a -1.

- Para poder usar la memoria es necesario que usemos un puntero que apunte a la zona compartida. Para asociar un puntero a dicha zona se usa la llamada shmat().

```
double *counter = (double *) shmat(shmid, NULL, 0);
```

- A partir de aquí el puntero counter se puede usar de la misma forma que se usa cualquier puntero reservado con malloc(). Una vez que la memoria ha dejado de usarse se libera con la llamada:

```
shmdt(counter);
```

- que es equivalente a la llamada free(). La memoria reservada no se libera mientras haya procesos que la están usando. Si se quiere modificar el comportamiento de la memoria compartida se puede usar la llamada shmctl(). Esta llamada es necesaria para indicarle al sistema que elimine la memoria compartida una vez que no haya ningún proceso usándola. Esto se hace mediante la llamada:

```
shmctl(shmid, IPC_RMID, NULL);
```

- El fichero shmemory.c muestra un ejemplo de un contador compartido entre varios procesos.

- La orden del sistema ipcs nos muestra información de la memoria compartida reservada por procesos de nuestro usuario. Con ipcrm podemos eliminar zonas de memoria compartidas que hayan quedado reservadas.

Ejercicios propuestos:

1. Implementar un proceso padre que cree 3 hijos y luego espera a que finalicen antes de finalizar él mismo.
2. Los tres hijos comparten un vector de 100 enteros. Cada uno de ellos ha de realizar una de las siguientes tareas:
    - Pedir un índice del vector y un valor entero por teclado y almacenar en la posición correspondiente el valor leído. Esta operación se ha de repetir 10 veces.
    - Cambiar aleatoriamente un elemento del vector y bloquearse durante 1 segundo. Esta operación se ha de repetir 100 veces.
    - Sumar todos los elementos del vector, mostrar la suma por pantalla y luego bloquearse durante 30 segundos. Esta operación se ha de repetir 5 veces.
3. El vector es inicialmente rellenado por el proceso padre con valores aleatorios.

Prototipos de las funciones:

```
--------------------------
pid_t fork(void);
pid_t getpid(void);
key_t ftok(const char *pathname, int proj_id);
int shmget(key_t key, size_t size, int shmflg);
void *shmat(int shmid, const void *shmaddr, int shmflg);
int shmdt(const void *shmaddr);
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
unsigned int sleep(unsigned int seconds);
```