Apellidos, Nombre: _____ Nº Matrícula: _____ D.N.I.: _____

Sistemas Operativos 5º Semestre. Grado GII

Tercer Parcial. Sincronización y comunicación. 19 de diciembre de 2018

Dispone de 40 minutos. Las notas saldrán el 8 de enero La revisión será el 10 de enero a las 11:00 en la sala de cristal S4200.

1	2	3	4	5	6

Problema (6 puntos)

Sean las siguientes piezas de código:

```
/* Código A */

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

  int pid;

  pid = fork();
    if (pid == 0) {
      << acceso a recurso  compartido
         con proceso padre >>
    }
    else {
      << acceso a recurso compartido
         con proceso hijo >>
    }
}
```

```
/* Código B */

/* Ejecutable1 */
int main(int argc, char * argv[]) {

  << acceso a recurso compartido
     con proceso Ejecutable2 >>

}

/* Ejecutable2 */
int main(int argc, char * argv[]) {

  << acceso a recurso compartido
     con proceso Ejecutable1 >>

}
```

```
/* Código C */

void *func (void *p) {
  << acceso a recurso compartido
     con el otro thread >>
}

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

  pthread_t th1, th2;


  pthread_create(&th1, NULL, &func, NULL);
  pthread_create(&th2, NULL, &func, NULL);

  pthread_join(th1, NULL);
  pthread_join(th2, NULL);
  return 0;
}
```

```
/* Código D*/

/* Ejecutable en máquina 1 */
int main(int argc, char * argv[]){

  << acceso a recurso compartido
     con proceso de máquina 2 >>

}

/* Ejecutable en máquina 2*/
int main(int argc, char * argv[]){

  << acceso a recurso compartido
     con proceso de máquina 1 >>

}
```

1) [1 punto] Rellenar la tabla de la siguiente página, indicando aquellos mecanismos que sean factibles y/o recomendables. En cada posición de la tabla, seleccionar una de las tres opciones: [No factible/Factible/Recomendable], para los distintos escenarios (códigos).

2) [1'5 puntos] Indicar para cada posición que hayáis rellenado como "No factible" la razón de dicha selección.

3) [1'5 puntos] Indicar para cada posición que hayáis rellenado como "Recomendable" la razón de dicha selección.

4) [2'5 puntos] Modificar el código A para que implemente una solución donde el proceso padre va leyendo de su entrada estándar números enteros, los incrementa en 1 y se los manda al proceso hijo para que los escriba por su salida estándar. Los procesos terminarán cuando el proceso padre lea un fin de fichero por su entrada estándar.

5) [1 punto] Indicar las modificaciones que serían necesarias para que la solución del apartado anterior se adapte al código B. No programar nada en este apartado.

6) [2'5 puntos] Modificar el código C para que implemente una solución donde los hilos deben alternarse en escribir un byte que leen por su entrada estándar en un fichero cuyo nombre se pasa como parámetro al ejecutable.

a)

Mecanismo / Código	Código A	Código B	Código C	Código D
Imagen de memoria única + semáforos sin nombre				
Imagen de memoria única + semáforos con nombre				
Imagen de memoria única + mútex y cond				
Región de memoria compartida (mmap + MAP_ANON+MAP_SHARED) + semáforos sin nombre				
Región de memoria compartida (mmap + MAP_ANON+MAP_SHARED) + semáforos con nombre				
Fichero proyectado + semáforos sin nombre				
Fichero proyectado + semáforos con nombre				
Pipes (tuberías sin nombre)				
FIFOS (tuberías con nombre)				
Fichero + cerrojos				
Sockets UNIX				
Sockets INET				

SOLUCIÓN:

1 La tabla rellena quedaría del siguiente modo:

Mecanismo / Código	Código A	Código B	Código C	Código D
Imagen de memoria única + semáforos sin nombre	*No factible*	*No factible*	*Factible*	*No factible*
Imagen de memoria única + semáforos con nombre	*No factible*	*No factible*	*Factible*	*No factible*
Imagen de memoria única + mútex y cond	*No factible*	*No factible*	*Recomendable*	*No factible*
Región de memoria compartida (mmap + MAP_ANON+MAP_SHARED) + semáforos sin nombre	*Recomendable*	*No factible*	*Factible*	*No factible*
Región de memoria compartida (mmap + MAP_ANON+MAP_SHARED) + semáforos con nombre	*Factible*	*No factible*	*Factible*	*No factible*
Fichero proyectado + semáforos sin nombre	*Factible*	*Factible*	*Factible*	*No factible*
Fichero proyectado + semáforos con nombre	*Factible*	*Recomendable*	*Factible*	*No factible*
Pipes (tuberías sin nombre)	*Recomendable*	*No factible*	*Factible*	*No factible*
FIFOS (tuberías con nombre)	*Factible*	*Recomendable*	*Factible*	*No factible*
Fichero + cerrojos	*Factible*	*Recomendable*	*No factible*	*No factible*
Sockets UNIX	*Factible*	*Factible*	*Factible*	*No factible*
Sockets INET	*Factible*	*Factible*	*Factible*	*Recomendable*

1 En el caso del código A, las únicas soluciones no factibles son aquellas que implican una imagen de memoria única, dado que los procesos padre e hijo no comparten esa imagen de memoria.

En el caso del código B, sólo son factibles aquellas soluciones en las que los mecanismos de comunicación y/o sincronización tienen nombre, tales como ficheros, FIFOS o sockets. L a combinación fichero proyectado con semáforos sin nombre es factible, definiendo los semáforos dentro de la región asociada al fichero proyectado.

En el caso del código C, la mayoría de las soluciones son factibles, aunque muchas de ellas no son recomendables, al tener todos los hilos una imagen de memoria única y ser más sencillo utilizar dicha imagen con algún mecanismo de sincronización ligero. La solución de ficheros más cerrojos no es factible, dado que el cerrojo se establece a nivel de proceso y no de hilo.

En el caso del código D, la única solución factible es la de los sockets de dominio INET, dado que ambos códigos se encuentran en máquinas diferentes.

2 En el caso del código A, son recomendables aquellas soluciones sin nombre, que pueden heredarse a través de descriptores de ficheros o como parte de una región de memoria compartida, dado que se trata de procesos padre e hijo.

En el caso del código B, las soluciones recomendables son aquellas en las que hay un mecanismo con nombre detrás, tales como ficheros, semáforos con nombre y FIFOS. Los sockets no son necesarios, dado que los procesos están en la misma máquina.

En el caso del código C, la solución recomendable es el uso de las variables compartidas, por tratarse de hilos que tienen una imagen de memoria única, más mútex y condiciones, por ser mecanismos ligeros, apropiados para los hilos.

En el caso del código D, la única solución recomendable y factible es la de los sockets de dominio INET, dado que ambos códigos se encuentran en máquinas diferentes.

3 El código A quedaría del siguiente modo:

```
int main(int argc, char * argv[]){
   int tub[2];
   int dato;

   pipe(tub);

   pid = fork();
   if (pid != 0) {
      close(tub[0]);
      while (scanf("%d",&dato)!= EOF) {
         dato++;
         write(tub[1],&dato, sizeof(int));
      }
      close(tub[1]);
   }
   else {
      close(tub[1]);
      while (read(tub[0], &dato, sizeof(int)) > 0) {
          printf("%d\n", dato);
      }
      close(tub[0]);
   }
      return 0;
}
```

4 El código del apartado anterior tendría que modificarse, de forma que, en lugar de utilizar una tubería sin nombre, ambos procesos usaran una tubería con nombre, dado que en este caso los procesos no tienen ningún parentesco.

5 Una posible solución sería utilizar una variable de control compartida, que determinara el turno, y mútex y condiciones para lograr los bloqueos necesarios:

```
int turno = 0, fd;
pthread_mutex_t mutex;
pthread_cond_t turno1, turno2;

void *func1 (void *p) {
   int dato;

   while (scanf("%d",&dato) != EOF) {
      pthread_mutex_lock(&mutex);
      while (turno != 0)
         pthread_cond_wait(&turno1,&mutex);
      pthread_mutex_unlock(&mutex);

       write(fd, &dato, sizeof(int));

       pthread_mutex_lock(&mutex);
       turno = 1;
       pthread_cond_signal (&turno2);
       pthread_mutex_unlock(&mutex);
   }
   pthread_exit(NULL);
}

void *func2 (void *p) {
   int dato;

   while (scanf("%d",&dato) != EOF) {
      pthread_mutex_lock(&mutex);
      while (turno != 1)
         pthread_cond_wait(&turno2,&mutex);
      pthread_mutex_unlock(&mutex);

      write(fd, &dato, sizeof(int));

      pthread_mutex_lock(&mutex);
      turno = 0 ;
      pthread_cond_signal (&turno1);
      pthread_mutex_unlock(&mutex);
   }
   pthread_exit(NULL);
}


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

   pthread_t th1, th2;

   pthread_mutex_init(&mutex,NULL);
   pthread_cond_init(&turno1, NULL);
   pthread_cond_init(&turno2, NULL);

   fd = creat(argv[1], 0640);

   pthread_create(&th1, NULL, &func1, NULL);
   pthread_create(&th2, NULL, &func2, NULL);

   pthread_join(th1, NULL);
   pthread_join(th2, NULL);

   close (fd);
   return 0;
}
```