Septiembre 2009 – Procesos y memoria

Sea un monoprocesador con sistema operativo de propósito general. El sistema operativo tiene planificación expulsiva, basada en prioridades y núcleo no expulsivo (no permite el anidamiento de llamadas al sistema). Supongamos que el sistema operativo maneja el concepto de *thread* y es capaz de planificar *threads*. En un momento determinado, tenemos los siguientes procesos planificables, además del proceso nulo:

PROCESO A1:

```
#define TAMBUF 1024
int main(int argc, char *argv[]) {
  int pp[2];
  int fd;
  char buffer[TAMBUF];
  int leidos;

  pipe(pp);
  if (fork()== 0) /* PROCESO A2 */ {
    close(pp[1]);
    while (
    (leidos = read(pp[0],buffer,TAMBUF))>0) {
      write(1,buffer,leidos);
    }
    close(pp[0]);
    exit(0);
  }
  else {
    fd = open("/home/luis/a.txt",O_RDONLY);
    close(pp[0]);
    while (
    (leidos = read(fd,buffer,TAMBUF))>0) {
      write(pp[1],buffer,leidos);
    }
    close(pp[1]);
    close(fd);
    exit(0);
  }
}
```

PROCESO B:

```
void manejador ( void *ptr );
sem_t semaforo;
int resultado;

int main(int argc, char *argv[]){
  int args[2];
  pthread_t thread_b1;
  pthread_t thread_b2;


  args[0] = 0;
  args[1] = 1;
  /* Inicializar un semáforo compartido entre
     los threads con el valor inicial 1 */
  sem_init(&semaforo, 0, 1);

  /* THREAD B1 */
  pthread_create (&thread_b1, NULL, (void *) &manejador, (void *)
&args[0]);
  /* THREAD B2 */
  pthread_create (&thread_b2, NULL, (void *) &manejador, (void *)
&args[1]);

  pthread_join(thread_b1, NULL);
  pthread_join(thread_b2, NULL);
  sem_destroy(&semaforo);
  exit(0);
}

void manejador (void *ptr ) {
  int val;
  val = *((int *) ptr);
  sem_wait(&semaforo);
  resultado = val;
  sem_post(&semaforo);
  pthread_exit(0);
}
```

PROCESO C:

```
#define TAMBUF 1024
int main(int argc, char *argv[])
{
  char buffer[TAMBUF];
  int leidos;

  while ( (leidos = read(0, buffer, TAMBUF)) > 0)
    write(1,buffer,leidos);
  exit(0);
}
```

PROCESO D:

```
int main(int argc, char *argv[])
{
  int fd;
  char *dir;
   struct stat sb;

  if (argc < 2) {
    fprintf(stderr, "Error. Uso: %s fich", argv[0]);
    exit(1);
  }
  fd = open(argv[1], O_RDONLY);
  fstat(fd, &sb);
  dir = mmap(NULL, sb.st_size, PROT_READ,
                MAP_PRIVATE, fd, 0);
  if (dir != MAP_FAILED) {
    write(1, dir, sb.st_size);
  }
  munmap(dir, sb.st_size);
```

PROCESO A1:

```
#define TAMBUF 1024
int main(int argc, char *argv[]) {
  int pp[2];
  int fd;
  char buffer[TAMBUF];
  int leidos;

  pipe(pp);
  if (fork()== 0) /* PROCESO A2 */ {
    close(pp[1]);
    while (
    (leidos = read(pp[0],buffer,TAMBUF))>0) {
      write(1,buffer,leidos);
    }
    close(pp[0]);
    exit(0);
  }
  else {
    fd = open("/home/luis/a.txt",O_RDONLY);
    close(pp[0]);
    while (
    (leidos = read(fd,buffer,TAMBUF))>0) {
      write(pp[1],buffer,leidos);
    }
    close(pp[1]);
    close(fd);
    exit(0);
  }
}
```

PROCESO B:

```
void manejador ( void *ptr );
sem_t semaforo;
int resultado;

int main(int argc, char *argv[]){
  int args[2];
  pthread_t thread_b1;
  pthread_t thread_b2;


  args[0] = 0;
  args[1] = 1;
  /* Inicializar un semáforo compartido entre
     los threads con el valor inicial 1 */
  sem_init(&semaforo, 0, 1);

  /* THREAD B1 */
  pthread_create (&thread_b1, NULL, (void *) &manejador, (void *)
&args[0]);
  /* THREAD B2 */
  pthread_create (&thread_b2, NULL, (void *) &manejador, (void *)
&args[1]);

  pthread_join(thread_b1, NULL);
  pthread_join(thread_b2, NULL);
  sem_destroy(&semaforo);
  exit(0);
}

void manejador (void *ptr ) {
  int val;
  val = *((int *) ptr);
  sem_wait(&semaforo);
  resultado = val;
  sem_post(&semaforo);
  pthread_exit(0);
}
```

```
  exit(0);
}
```

Los procesos A1 y B son ejecutados por el usuario *Luis* y los procesos C y D por el superusuario. En el momento de creación de todos los procesos, la prioridad del proceso D es 0, la del proceso C es 1, la del proceso A1 es 4 y la del proceso B es 3. Adicionalmente existe un proceso periódico (que denominaremos E) que se activa cada *n* milisegundos y que modifica las prioridades de los procesos y *threads* planificables. La prioridad del proceso E es -1. Se supone que un valor menor corresponde a una mayor prioridad.

Considerando que el sistema operativo sigue un modelo de procesos (núcleo con ejecución dentro de los procesos o *threads* de usuario), se pide:

1. **[1 punto]** Clasificar todos los procesos y *threads* del sistema, incluido el proceso nulo, como procesos (o *threads*) de usuario o procesos (o *threads*) de núcleo. Justificar la respuesta.
2. **[3 puntos]** En la situación descrita, plantear un escenario en el que esté en ejecución el proceso de menor prioridad y no haya finalizado el resto de los procesos. Para ello:
   a. Describir las acciones que habrían llevado a cabo todos los procesos hasta ese mismo instante.
   b. Especificar en qué rutina de tratamiento de evento estaría cada uno de los procesos del escenario.
   c. El resto de procesos, ¿en qué modo de ejecución estarían? Justificar la respuesta.
3. **[2 puntos]** ¿Existe alguna diferencia en la resolución del apartado anterior si se trata de un núcleo expulsivo (permite el anidamiento de llamadas al sistema)? Justificar la respuesta, destacando las principales diferencias entre los núcleos expulsivos y no expulsivos. Al resto de las preguntas del

examen, responder considerando un núcleo no expulsivo.

4. **[1 punto]** Analizar si de algún modo se podría dar una situación de inversión de prioridades, considerando que la inversión de prioridades se da cuando un proceso o *thread* de baja prioridad ha adquirido un recurso requerido por un proceso o *thread* de alta prioridad, lo que no permite ejecutar a este último. En caso afirmativo, describir lo que habría ocurrido en el sistema justo antes de dicha situación y el estado en que se encontrarían todos los procesos en ese instante.

5. **[3 puntos]** Suponiendo que la siguiente tabla corresponde a una posible traza de ejecución de los procesos o *threads* en un momento determinado, rellenad con valores adecuados la tabla adjunta. Se ha rellenado la primera fila de la tabla como punto de partida.

Nombre y apellidos: .....

Traza	Descripción de la situación (proceso en ejecución, línea de ejecución si se trata de proceso en ejecución en modo usuario, motivo por el que se genera un determinado evento, etc.)	Estado y modo de A1	Estado y modo de A2	Estado y modo de ejecución B1	Estado B2	Estado C	Estado D	Estado E
1. Proceso de usuario en ejecución (modo usuario)	*Thread B1 en ejecución, justo antes de ejecutar la llamada sem_wait().*	*Bloqueado (modo sistema)*	*Bloqueado (modo sistema)*	*En ejecución (modo usuario)*	*Listo para ejecutar (modo sistema)*	*Bloqueado (modo sistema)*	*Bloqueado (modo sistema)*	*Bloqueado (modo sistema)*
2. Llamada al sistema. Paso a modo sistema	*Thread* B1 ejecutando en modo sistema el código de la llamada sem_wait().	Bloqueado (modo sistema)	Bloqueado (modo sistema)	En ejecución (modo sistema)	Listo para ejecutar (modo sistema)	Bloqueado (modo sistema)	Bloqueado (modo sistema)	Bloqueado (modo sistema)
3.Tratamiento de una interrupción hardware	Llega una interrupción de disco, asociada a la llamada al sistema open(), que llevo a cabo el proceso D. La rutina de tratamiento pone en estado Listo al proceso D.	Bloqueado (modo sistema)	Bloqueado (modo sistema)	En ejecución (modo sistema)	Listo para ejecutar (modo sistema)	Bloqueado (modo sistema)	Listo para ejecutar (modo sistema)	Bloqueado (modo sistema)
4.Finalización de la llamada al sistema.	*Thread* B1 finaliza la llamada al sistema sem_wait()	Bloqueado (modo sistema)	Bloqueado (modo sistema)	En ejecución (modo sistema)	Listo para ejecutar (modo sistema)	Bloqueado (modo sistema)	Listo para ejecutar (modo sistema)	Bloqueado (modo sistema)
5. Cambio de contexto involuntario. Proceso en ejecución en modo núcleo. RETI.	El proceso D, que tiene mayor prioridad, se pone a ejecutar en el punto donde se quedó, dentro de la rutina de la llamada open().	Bloqueado (modo sistema)	Bloqueado (modo sistema)	Listo para ejecutar (modo sistema)	Listo para ejecutar (modo sistema)	Bloqueado (modo sistema)	En ejecución (modo sistema)	Bloqueado (modo sistema)
6. Paso a modo usuario.	El RETI hace que el proceso D vuelva a ejecutar en modo usuario la siguiente sentencia al open().	Bloqueado (modo sistema)	Bloqueado (modo sistema)	Listo para ejecutar (modo sistema)	Listo para ejecutar (modo sistema)	Bloqueado (modo sistema)	En ejecución (modo usuario)	Bloqueado (modo sistema)
7. Tratamiento de una	Llega una interrupción de	Bloqueado	Bloqueado	Listo para ejecutar	Listo para	Bloqueado	En ejecu-	Listo para

interrupción hardware	reloj, que va a activar al proceso E. La rutina de tratamiento pone en estado Listo al proceso E. Todo se ejecuta dentro del contexto del proceso D, en modo sistema.	(modo sistema)	(modo sistema)	(modo sistema)	ejecutar (modo sistema)	do (modo sistema)	ción (modo sistema)	ejecutar (modo sistema)
8. Cambio de contexto involuntario. Proceso en ejecución en modo núcleo. RETI.	El proceso E, que tiene mayor prioridad, se pone a ejecutar en el punto donde se quedó.	Bloquea-do (modo sistema)	Bloquea-do (modo sistema)	Listo para ejecutar (modo sistema)	Listo para ejecutar (modo sistema)	Bloquea - do (modo sistema)	Listo para ejecutar (modo sistema)	En ejecu-ción (modo sistema)
9. Proceso en ejecución en modo núcleo.	Al ser un proceso de núcleo, su código también se ejecuta en modo sistema.	Bloquea-do (modo sistema)	Bloquea-do (modo sistema)	Listo para ejecutar (modo sistema)	Listo para ejecutar (modo sistema)	Bloquea - do (modo sistema)	Listo para ejecutar (modo sistema)	En ejecu-ción (modo sistema)
10. Cambio de contexto voluntario. Proceso en ejecución en modo núcleo. RETI.	El proceso E se bloquea, esperando otros n milisegundos. El planificador selecciona el proceso de mayor prioridad, que supongamos que sigue siendo el proceso D.	Bloquea-do (modo sistema)	Bloquea-do (modo sistema)	Listo para ejecutar (modo sistema)	Listo para ejecutar (modo sistema)	Bloquea - do (modo sistema)	En ejecu-ción (modo sistema)	Bloquea do (modo sistema)
11. Proceso en ejecución en modo usuario.	El proceso D sigue su ejecución en modo usuario.	Bloquea-do (modo sistema)	Bloquea-do (modo sistema)	Listo para ejecutar (modo sistema)	Listo para ejecutar (modo sistema)	Bloquea - do (modo sistema)	En ejecu-ción (modo usuario)	Bloquea do (modo sistema)

SOLUCIÓN

1. Los procesos existentes en el instante especificado son:
   Proceso A1: Proceso de usuario, ejecutado por el usuario *Luis*. Ejecuta el código especificado en el programa adjunto.
   Proceso A2: Proceso de usuario, dado que es un proceso hijo del proceso A1.
   Proceso B: Proceso de usuario, ejecutado por el usuario *Luis*. Ejecuta el código especificado en el programa adjunto.
   Thread B1: Thread de usuario creado por el proceso B.
   Thread B2: Thread de usuario creado por el proceso B.
   Proceso C: Proceso de usuario, ejecutado por el superusuario. Ejecuta el código especificado en el programa adjunto.
   Proceso D: Proceso de usuario, ejecutado por el superusuario. Ejecuta el código especificado en el programa adjunto.
   Proceso E: Proceso de núcleo, dado que requiere modificar las prioridades de los procesos y su ejecución se llevara en todo momento en modo núcleo.
   Proceso nulo: Proceso de núcleo de mínima prioridad que ejecuta cuando no hay ningún otro proceso activo en el sistema.
2. En este apartado se puede tener en cuenta al proceso nulo o no. En el primer caso, y siempre que el proceso E no haya modificado las prioridades de los procesos, el proceso de menor prioridad (sin tener en cuenta al proceso nulo) sería el proceso A1 y consecuentemente, el proceso A2, que heredará la misma prioridad.

   a. En este caso, para que pueda ejecutar A1, el resto de procesos a excepción de tal vez el proceso A2 (que tiene la misma prioridad) deberían estar bloqueados. Un posible escenario se describe a continuación. Supongamos que durante la secuencia que se va a describir el proceso E no entra a ejecutar, puesto que no han finalizado los n milisegundos y se encuentra bloqueado esperando a que llegue la interrupción de reloj correspondiente que le active. Empieza a ejecutar el proceso D, que se queda bloqueado en la llamada al sistema open(), dado que es necesario acceder al disco para hacer la traducción del nombre del fichero. El siguiente proceso en ejecución sería el proceso C, que se quedaría bloqueado en la llamada al sistema read(), puesto que el usuario aún no ha escrito ningún carácter. A continuación se pone en ejecución el proceso B, que crea los dos *threads* B1 y B2. Este proceso se bloquea en la primera llamada pthread_join(), que le bloquea hasta que finalice el *thread* B1. Por su parte B1, tomaría el semáforo mediante la llamada sem_wait(), pero cuando va a ejecutar la sentencia resultado = val; se produce un fallo de página al acceder a la variable resultado. B2 se queda bloqueado en la llamada sem_wait(), puesto que el semáforo ha sido asignado a B1. En esa situación el proceso A1 o el proceso A2 (que son los procesos de menor prioridad), podrían entrar a ejecutar.
   En el caso de que se considere el proceso nulo, éste entraría a ejecutar en la situación anterior si tanto A1 como A2 están bloqueados. Por ejemplo, A1 podría estar bloqueado en la llamada
   open(), accediendo a disco para traducir el nombre de fichero y A2 en la llamada read(), esperando leer de la tubería.

   b. El proceso E estaría bloqueado en modo núcleo esperando a que llegue la interrupción de reloj que le active.
   El proceso D estaría bloqueado en la rutina de tratamiento de la llamada open().
   El proceso C estaría bloqueado en la rutina de tratamiento de la llamada read().
   El proceso B estaría bloqueado en la rutina de tratamiento de la llamada pthread_join().
   El *thread* B1 estaría en la rutina de tratamiento de la excepción de fallo de página.
   El *thread* B2 estaría en la rutina de tratamiento de la llamada sem_wait().

En el segundo caso, cuando es el proceso nulo el que está en ejecución:
El proceso A1 estaría bloqueado en la llamada open().
El proceso A2 estaría bloqueado en la llamada read().

c. Los procesos que no se encuentran en ejecución se encuentran en modo sistema, puesto que fue en ese modo donde llevaron a cabo un cambio de contexto.

3. En un núcleo expulsivo el cambio de contexto se difiere sólo hasta que terminen las rutinas de interrupción, por contraposición al núcleo no expulsivo, en el que el cambio de contexto se difiere hasta que, además de las rutinas de interrupción anidadas, también termine la llamada al sistema o de excepción, en ejecución, si la hubiera. En los núcleos expulsivos puede haber un cambio de contexto involuntario en mitad de una llamada, pudiendo haber llamadas concurrentes. Esto implica una casuística más compleja a la hora de analizar los posibles problemas de sincronización de los núcleos expulsivos. Desde el punto de vista del apartado anterior, no habría diferencias, dado que todos los procesos llevaron a cabo cambios de contexto voluntarios.
4. Los *threads* B1 y B2 comparten un semáforo. Inicialmente estos *threads* tienen la misma prioridad, por lo que no se puede dar una situación de inversión de prioridades. No obstante, si el proceso E modifica la prioridad de uno de ellos, incrementándole la prioridad, y el otro tiene asignado el semáforo, se daría una inversión de prioridades, debido a que el proceso de mayor prioridad estaría esperando por un recurso adquirido por un proceso de menor prioridad. Otra situación que se puede considerar inversión de prioridades, aunque con más matices, se da en el uso de los núcleos no expulsivos, concretamente cuando un proceso de mayor prioridad, que se ha desbloqueado debido a la llegada de una interrupción, tiene que esperar a que finalice el tratamiento de la llamada al sistema del proceso en ejecución, que tiene menor prioridad. En este caso, el recurso compartido sería el procesador.
5. Resuelto en la propia tabla.