Ingeniería Informática. Curso 3º.
Sistemas Operativos
Examen Final. PROBLEMAS.
8 de Febrero de 2008

Resuelva **DOS de los tres problemas** planteados a continuación.

1. Se pretende implementar un mecanismo simplificado de tuberías a partir de **cerrojos y variables condicionales**. En concreto, el objetivo es que el sistema operativo disponga de una única tubería en la que sólo se pueden escribir enteros y a la que todos los procesos del sistema tienen acceso. La tubería está creada desde el arranque del sistema y no se destruye en ningún momento. Si un proceso quiere usar la tubería, deberá hacer una llamada "open_tub(r|w)" indicando el modo de apertura. Posteriormente, accederá a la tubería mediante llamadas read_tub/write_tub, cuyo comportamiento debe ser similar al de las tuberías habituales. Finalmente, el proceso realizará una llamada a close_tub. El tipo de la tubería es el siguiente:

```
struct  t_tuberia  {
  int tubo[TAM]; // array en el que se escribirán los datos
  int Nelem; // Num elems. en la tubería.
  int Nhuecos;  // Número de huecos.
  int posEsc;  // Posición de escritura en la tubería
  int posLect;  // Posición de lectura en la tubería
  int nCons; // Num. procesos que han abierto la tubería para lectura
  int nProd; // Num. procesos que han abierto la tubería para escritura
  mutex_t cerrojo;
  cond_t  esperaP, esperaC;
} pipe;
```

   a. (2 puntos) Codifique los procedimientos *open_tub(modo)* y *close_tub(desc)*. *open_tub* recibe como argumento el modo de apertura (lectura o escritura), y devuelve un identificador (un booleano que indica si la apertura fue hecha para lectura o para escritura). *close_tub* recibe como argumento el identificador. En ningún caso es necesario inicializar/destruir cerrojos o variables condicionales.
   b. (3 puntos) Codifique los procedimientos *read_tub* y *write_tub*, que recibirán como parámetro el identificador, la variable origen/destino de los datos y , el tamaño de la lectura/escritura (que indica cuántos enteros se van a leer/escribir). Ambas operaciones se deben comportar exactamente igual que sus equivalentes de las tuberías estudiadas en clase, y devolverán el número de enteros leídos/escritos, -1 en caso de error ó 0 en caso de fin de lectura.

Ejemplo de uso de la tubería "comunitaria":

```
bool desc= open_tub(write); // desc será true => permite escrituras, pero no lecturas
int buf[10] = {1,2,3,4,5,6,7,8,9,10};
// La llamada a write escrbirá 6 enteros en la tubería.
// Si no hay extremo de lectura abierto, lo notificará.
// Si los 6 enteros no caben en la tubería, el proceso se bloqueará hasta que haya  espacio para los 6.
write_tub(desc,buf,6);
close_tub(desc);
```

2. Responda a las siguientes preguntas sobre el código que se presenta a continuación:

   a. (2 puntos) Dibuje el estado de las tablas de descriptores de ficheros (tdda), intermedia (TIN) y de nodos-v (T-nodos) justo después de *wait(NULL)* y justo después de la llamada a *close(fd2)*. Asuma que el proceso **padre es más prioritario** que el hijo.
   b. (1 punto) Indique el contenido de los arrays "data" y "bufR" en los dos instantes de tiempo anteriores a la finalización de cada uno de los procesos.
   c. (2 puntos). El sistema de ficheros usado es una simplificación del modelo de Unix: nodos-i con 5 punteros directos, un indirecto simple y uno doble. Cada puntero ocupa 4 bytes, y los bloques son de 1KB (1024 bytes). Describa el contenido de los punteros de asignación de bloques del nodo-i del fichero "cejel" tras la ejecución de este código asumiendo que no existía inicialmente. Indique asimismo el contenido final del fichero, incluyendo todos los bloques de datos relevantes.

```
int fd1;
char data[7]=”Moaxaja”;
main() {
  int fd2;
  char bufR[10];
  close(STDOUT);
  fd2=open(“cejel”,O_CREATE…);
  if (fork() == 0) {
       prosigue(fd2);
       exit(0);
  }
  else {
    fd1=open(”cejel”,O_RDWR);
    wait(NULL);
    read(fd1,bufR,10);
    lseek(fd2,1024,SEEK_CUR);
    write(fd2,data,7);
  }
}
```

```
void prosigue (int  fd2) {
  char buf1[500]=”aaaa...a”;
  char buf2[400]=”bbbb...b”;

  fd1=open(”cejel”,O_RDWR);
  fprintf(STDOUT,”Alfaquí”);
  read(fd1,data,7);
  lseek(fd1,10240,SEEK_SET);
  for (int i=0;i<2;i++) {
    write(fd1,buf1,500);
    lseek(fd1,1548,SEEK_CUR);
    write(fd2,buf2,514);
  }
  close(fd2);
}
```

3. El programa codificado a continuación se va a ejecutar :
- En un sistema monoprocesador con memoria referenciada en marcos de páginas de 4KB y con dos discos, uno de ello dedicado a swap. El tiempo de acceso a un bloque de 4KB en disco es de 20 mseg; el de acceso a 2 bloques consecutivos es de 25 mseg.
- Bajo un sistema operativo que gestiona memoria virtual con paginación por demanda, asignación fija (estática) de 8 marcos, política de reemplazo LRU, y planificación de procesos FIFO.
- El procesamiento de 1024 elementos del array A (1024 iteraciones del bucle) tarda una media de 4 mseg en CPU.

El programa es el siguiente:

```
#define TAM 6*4096
#define NHILOS 2

char A[TAM] = {75, -10, …} // vector inicializado a procesar
int x;
int main() {
        pthread_t tid1, tid2;

        pthread_create(&tid1, NULL, procesar, 0);
        pthread_create(&tid2, NULL, procesar, 1);
        pthread_join(tid1, NULL);
        pthread_join(tid2, NULL);
        exit(0);
}
```

```
void procesar(int arg) {
        int elem, pel = arg*TAM/NHILOS, uel=(arg+1)*TAM/NHILOS;

        elem = pel;
        while (elem < uel) {
                A[elem] = Operacion_muy_lenta(A[elem]);
                elem++;
        }
        return;
}
```

El código del programa, y cualquier otra región de la que no se tengan datos adicionales ocupan 1 página cada una.

a) (1 punto) Indique las regiones del mapa de memoria del proceso mientras ambos hilos están activos.
b) (1,5 puntos) Calcular el tiempo de ejecución mostrando el orden de ejecución de los hilos, identificando los fallos de página e indicando si para resolverlos es necesario acceder a disco (en ese caso, especificar a qué disco y si cada acceso es de lectura o escritura)
c) (1.5 puntos) Repetir asumiendo que se usa pre-paginación de dos páginas consecutivas únicamente en las páginas de la región de datos.
d) (1 punto) Repetir para el caso en que exista un único disco físico con dos particiones y con el supuesto original sin preasignación.

Ingeniería Informática. Curso 3º.
Sistemas Operativos
Examen Final. TEORIA.
8 de Febrero de 2008

1. **[PROCESOS]**

a) Considerar el siguiente código:

```
int a=0;
main() {
  int b=1;
  for (int i=0; i < 3; i++) {
    if (fork() == 0) {
      b=b+2;
      thread_create(&tid,suma,b);
      thread_join(tid);
      exit(0);
    }
    else {
      wait(NULL);
      a=a+b;
      b=b+1;
    }
  }
   printf("a=%d, b=%d\n",a,b);
}
```

```
void suma (int  num) {
    a = num + a;
}
```

Mostrar la salida por pantalla de este código. Justificar su respuesta.

b) En el BCP de cada proceso existe una tabla de señales TSig donde a cada posible señal se le asocia una acción que puede ser: SIG_IGN (ignorar), SIG_DFL (tratar por defecto: habitualmente terminar el proceso) o ejecutar una función especificada. ¿Qué ocurre con las entrada de TSig en el BCP del proceso hijo inmediatamente después de la llamada fork()? ¿Qué ocurre con las entradas de TSig de un proceso inmediatamente después de una llamada exec()?

2. **[SINCRONIZACION Y COMUNICACIÓN]**

Contesta a las siguientes preguntas relativas al mecanismo IPC conocido como "mutex"
a) ¿Es mutex un mecanismo de sincronización o de comunicación?
b) Codifica el comportamiento de un mutex con implementación de espera activa empleando la instrucción Test-and-Set.
c) Codifica el comportamiento de un mutex con implementación de espera mixta (100 consultas activas que desembocan en una espera pasiva)
d) Aplicado a resolver problemas de exclusión mutua indica cómo actúa un mutex implementado con espera pasiva para satisfacer cada una de las 3 propiedades de una solución correcta a dicho problema.

3. **[MEMORIA]**

a) ¿Qué se entiende por espacio de trabajo ("working set") de un proceso?
b) ¿Qué se entiende por espacio residente de un proceso?
c) Listar las operaciones que puede efectuar un sistema operativo como LINUX en relación con regiones de memoria. Para cada una de ellas indicar qué llamada al sistema o qué excepción/interrupción puede demandarla.

4. **[ENTRADA/SALIDA]**

a) ¿Qué capa podría ser responsable de la aplicación de una política de planificación de disco: el módulo del sistema de ficheros, el gestor de bloques, el manejador del disco, el controlador hardware del disco? Justificar.
b) ¿Qué eventos o acciones, durante la actividad de un sistema operativo, se resuelven con operaciones de E/S sobre disco? Justificar.

5. [**SISTEMA DE FICHEROS**]

a) Dibuja el esquema de una entrada de directorio en un sistema de fichero tipo UNIX (p.ej, ext3) y en un sistema de ficheros tipo FAT
b) Un sistema de ficheros tipo FAT no soporta enlaces físicos. ¿Por qué?
c) Se piensa incluir una llamada en el kernel que cuente el número de ficheros existentes en un determinado sistema de ficheros, de la forma más rápida posible. ¿Cómo lo harías en un sistema de ficheros “ext3”? ¿Y en un sistema de ficheros FAT?