Departamento de Arquitectura

Sistemas Operativos
Práctico 1

Curso 2025

Ejercicio 1 Utilizando llamados al sistema, implementar un programa que lea un archivo (cuyo nombre recibe como parámetro) e imprima su contenido en el dispositivo impresora. Dicho dispositivo de nombre *impresora* cuenta con un buffer de 512 bytes. Cuando el dispositivo tiene el buffer vacío, se lo indica al sistema operativo, el cual genera el evento *impresora_libre*. El sistema operativo ofrece los siguientes llamados al sistema:

- `Fin()`
  Indica final exitoso de la ejecución.
- `Abort()`
  Indica final anormal de la ejecución.
- `Abrir_archivo(nombre_archivo): Entero` y `Cerrar_archivo(nombre_archivo): Entero`
  Abren y cierran un archivo dado su nombre. Devuelven `-1` en caso de error.
- `Leer(nombre_archivo, buffer, n): Entero`
  Lee n caracteres de un archivo identificado por su nombre, almacenándolos en el array buffer, y devuelve la cantidad de bytes leídos, o `-1` en caso de error. En caso de que se haya alcanzado el fin del archivo, devuelve `0`.
- `Obtener_recurso(nombre_recurso)` y `Liberar_recurso(nombre_recurso)`
  Obtiene y libera un recurso (que puede ser un dispositivo, memoria, etc.) dado su nombre. Se bloquean hasta que el recurso sea obtenido o liberado.
- `Escribir_dispositivo(nombre_dispositivo, buffer, n): Entero`
  Escribe n bytes del buffer en el dispositivo indicado por el nombre, y devuelve la cantidad de bytes que pudo escribir, o `-1` en caso de error.
- `Esperar_evento(nombre_evento)`
  Bloquea al proceso hasta que suceda el evento esperado. Retornan el control inmediatamente si el evento ya ocurre en el momento de invocarlo.

Nota: El sistema es multiprogramado.

Este ejercicio propone leer un archivo y escribir su contenido en una impresora. A primera vista, desde el punto de vista del programa, se trata de una tarea resoluble con conocimientos obtenidos en asignaturas previas a Sistemas Operativos. Se debe interactuar con dos dispositivos de E/S: el disco (al cual se accede a través del sistema de archivos) y la impresora. Desde un punto de vista lógico, un posible pseudocódigo inicial es el siguiente:

```
1) abrir archivo
2) mientras queden caracteres
3)      leer caracteres de archivo
4)      escribir caracteres en impresoras
5) cerrar archivo
```

Si se presta atención a las operaciones brindadas, se puede notar un mapeo entre estas y los pasos del pseudocódigo. El paso 1) se implementa con la función Abrir_archivo, el paso 3) se implementa con la función Leer, el paso 5 se implementa con la función Cerrar_archivo. El paso 4) aparentemente se implementa con la función Escribir_dispositivo, pero por letra, se dispone de una función Obtener_dispositivo, la cual presumiblemente se debe utilizar previo a escribirlo. En este punto es donde se debe realizar un paréntesis para hablar de sistemas operativos.

En esta asignatura siempre se debe tener en cuenta que un programa en ejecución es un proceso, y que a efectos prácticos, nunca es el único en ejecución. Esto quiere decir que el programa comparte los recursos del sistema donde ejecuta con otros procesos en ejecución. En este ejercicio, la impresora no puede ser escrita directamente por un proceso. Debe primero invocarse el system call Obtener_recurso. Tiene sentido que mientras un proceso utiliza una impresora, este sea el único que envía caracteres a imprimir, de no ocurrir esto, la impresora podría imprimir, por ejemplo, una línea de un proceso y a continuación una línea de otro proceso. Para evitar este potencial problema, en este ejercicio, el sistema operativo obliga a los procesos a invocar Obtener_recurso previo a poder escribirlo. Con esta llamada inicial, el sistema operativo puede asignar el recurso al primer dispositivo que la ejecute, bloqueando procesos que soliciten el recurso mientras el primer proceso no lo libere. Esto ilustra también el por qué es necesario que la función Escribir_dispositivo sea una llamada al sistema y por qué es necesario que la comunicación con el dispositivo se realice únicamente en modo monitor: si fuera de otra forma, otro proceso podría imprimir entre las impresiones de otro. Dado que Escribir_dispositivo es una llamada al sistema, el sistema operativo puede controlar que el proceso invocador haya ejecutado la función Obtener_recurso, y devolver error en caso contrario.

Tomando en cuenta lo anterior, el pseudocódigo se actualiza a:

```
1) abrir archivo
2) obtener recurso impresora
3) mientras queden caracteres
4)      leer caracteres de archivo
5)      escribir caracteres en impresoras
6) liberar recurso impresora
7) cerrar archivo
```

Lo siguiente a depurar es la espera por el dispositivo. En la letra se menciona que la escritura se debe realizar cuando la impresora esté libre. Para esto, se dispone de la llamada al sistema Esperar_evento, la cual bloquea el proceso hasta que se dispara el evento indicado. Es claro que esta función también debe ser una llamada al sistema, pues es el sistema operativo quien tiene la posibilidad de bloquear al proceso invocador en caso de que el evento aún no se haya disparado, y en el caso del evento 'impresora_libre', es el sistema operativo quien puede interactuar con el dispositivo para consultar su estado. Una aclaración pertinente es que la interfaz presentada es una de las muchas posibles, el sistema operativo podría ofrecer una versión bloqueante de la llamada Escribir_dispositivo, donde el proceso

se bloquee hasta que el dispositivo se encuentre disponible, y recién ahí que se realice la escritura, esencialmente incorporando lo que realiza la función Esperar_evento. Tanto la versión indicada en la letra como esta versión alternativa, son del estilo de llamadas al sistema *bloqueantes (sincrónicas)*, más adelante se discutirá una versión con llamadas al sistema *no bloqueantes (asincrónicas)*.

Estas consideraciones resultan en el siguiente pseudocódigo:

```
1) abrir archivo
2) obtener recurso impresora
3) mientras queden caracteres
4)      leer caracteres de archivo
5)      esperar evento (impresora libre)
6)      escribir caracteres en impresoras
7) liberar recurso impresora
8) cerrar archivo
```

Por último, algo a tener en cuenta es que todas las funciones del sistema provistas pueden fallar. Por ejemplo, la apertura de un archivo podría fallar por no existir el archivo con nombre provisto. La lectura del contenido de un archivo podría fallar por un fallo al leer un sector del disco, la escritura sobre la impresora podría fallar por una desconexión, etc. La existencia de fallos expone dos razones por las que es necesario/útil que estas funcionalidades sean abstraídas por llamadas al sistema. En primer lugar, si la ocurrencia de un error puede dejar el sistema en un estado inconsistente, es imperativo que este error sea capturado por el sistema operativo, de modo que este pueda manejar el error de forma apropiada. Un proceso que accediera directamente a los dispositivos de E/S o al sistema de archivos, pero que no lo maneje correctamente, podría dejar el sistema en un estado inconsistente, inutilizándolo también para *otros* procesos. En segundo lugar, es útil para los procesos que el sistema operativo maneje las posibles casuísticas y devuelva un código de error limpio, es un *servicio*, en el sentido coloquial de la palabra, hacia los programas.

Considerando el manejo de errores, la solución del ejercicio se presenta a continuación:

Solución:

```
#define TAM_BUF 512;

void imprimirArchivo (char* nombre_archivo) {
  byte buff[TAM_BUF];
  int fin=0, error=0;
  int codLeer;

  if(Abrir_archivo(nombre_archivo)==-1) {
    Abort();
  }
  // el proceso pasa a estado bloqueado si no está disponible la impresora
  Obtener_recurso(impresora);

  do {
    codLeer = Leer(nombre_archivo, buff, TAM_BUF);
    if (codLeer == -1) {
      error = 1;
      fin = 1;
    }
    if (!error) {
      fin = (codLeer == 0);
      if (!fin) {
        Esperar_evento(impresora_libre);
        if (Escribir_dispositivo(impresora, buff, codLeer)!=codLeer) {
          error = 1;
          fin = 1;
        }
      }
    }
  } while (!fin);

  Liberar_recurso(impresora);
  if (Cerrar_archivo(nombre_archivo)==-1) {
    Abort();
  }

  if (error) {
    Abort();
  }

  Fin();
}
```

Otras consideraciones

Como es mencionado anteriormente, los llamados al sistema presentados son bloqueantes (sincrónicos), es decir, que la invocación de uno de estos llamados bloquea al proceso llamador hasta que la llamada al sistema termina. En el caso de escribir en la función Obtener_recurso, la espera puede ser larga, pues pueden existir varios procesos que estén esperando para utilizar la impresora, y se desconoce qué uso le darán.

Una alternativa a las funciones sincrónicas es que sean 'asincrónicas'. Las llamadas al sistema asincrónicas realizan las actualizaciones mínimas necesarias para comenzar el pedido del proceso llamador, y luego retornan el control de la CPU a este, realizando la tarea indicada a través de otro proceso

(del sistema operativo). Para este tipo de funciones, el sistema operativo presenta a los procesos la posibilidad de consultar el estado de la solicitud, en otra llamada al sistema. Por ejemplo, si la función Obtener_recurso fuera asincrónica, el sistema operativo debería proveer una función adicional que permita al programa consultar si el recurso le fue asignado, esta llamada al sistema podría tener el cabezal Consultar_obtencion_recurso(nombre recurso), que devuelve 0 en caso de no disponer del recurso y 1 en caso de que el resurso esté asignado al proceso invocador. Se utilizaría de la siguente manera:

```
(...)
Obtener_recurso(impresora);

int tengoImpresora = 0;
do{
    tengoImpresora = Consultar_obtencion_recurso(impresora)

    // otras tareas
}while (tengoImpresora != 1)
(...)
```

La utilidad de esta modalidad, es la posibilidad de realizar otras tareas mientras se espera por la obtención del dispositivo.