Sistemas Operativos y Persistencia

García Perez-Schofield J. Baltasar.
Departamento de Informática. Universidad de Vigo, España(Spain).
jbgarcia@uvigo.es

October 13, 2003

Abstract

Los sistemas de programación persistentes realizan automáticamente la labor de almacenamiento y recuperación de las estructuras de datos de las aplicaciones. Las ventajas de esta automatización son principalmente tres: el eliminar una parte de la aplicación que es tediosa de programar, repetitiva y que supone un tamaño considerable de líneas de código; eliminar los errores derivados de la programación de las partes de la aplicación anteriores; y, finalmente, eliminar la diferenciación entre memoria principal y secundario, siendo un todo contínuo para el programador. Los sistemas operativos de la actualidad se basan en el diseño del sistema operativo UNIX, creado en los años 70. Su filosofía de trabajo se basa en la metáfora de ficheros: cualquier elemento del sistema operativo (driver, proceso en memoria ... etc), puede ser tratado como un fichero, de forma que se puede abrir, leer, y cerrar. En este sentido, el estancamiento en sistemas operativos es patente. El avance en este campo ha venido de la mano de los interfaces de usuario: éstos permiten obtener una visión del computador orientada a objetos hasta, un cierto grado, ya que el hecho de estar basados en un sistema operativo tradicional supone varias limitaciones. En este coloquio se examina esta situación en detalle, y se propone una solución basada en sistemas operativos orientados a objetos.

1 Introducción

"Persistencia" es un concepto asociado a la orientación a objetos: se trata, ni más, ni menos, de poder almacenar y recuperar objetos, de igual forma que se puede almacenar y recuperar un entero, una cadena, o un flotante.

Por otra parte, el sistema operativo, como es bien sabido, es el software mínimo que precisa un ordenador para poder funcionar, ofreciendo las posibles funcionalidades del hardware al usuario.

En este artículo, se presenta la evolución de ambos conceptos, y la relación entre ellos.

1.1 Persistencia

El grado de transparencia que un sistema orientado a objetos debe proporcionar para que su mecanismo de salvaguarda y recuperación de objetos sea considerado como persistente varía de un sistema a otro y de un autor a otro. Sin embargo, se acepta globalmente un tipo de persistencia especial, la "Persistencia Ortogonal", (ortogonalidad en el sentido de independencia) que se basa en tres principios ([**?**, **?**]):

- **Independencia del tipo:** Un objeto puede ser almacenado y recuperado sin importar de qué tipo o clase es.
- **Independencia del trato:** No deberían existir diferencias en el trato entre objetos persistentes y no persistentes.
- **Transitividad de persistencia:** La decisión de si un objeto debe ser o no persistente corresponde al sistema, no al programador, y normalmente ésto se consigue mediante algún algoritmo de referencia desde una raíz persistente.

Lo que ha llegado a los lenguajes de programación orientados a objetos, de persistencia, es más bien poco. Mientras que C++ incorpora el mismo mecanismo [**?**, **?**] que ya tenía C (`fwrite(f, &Obj, sizeof(obj), 1);`), Java avanza un poco más e incorpora un mecanismo de serialización más avanzado ([**?**, **?**]), que permite serializar el cierre persistente ([**?**, **?**])(es decir, todos los objetos relacionados con uno dado).

1.2 Sistemas Operativos

Los sistemas operativos han evolucionado sobre todo desde la aparición del primer sistema operativo "popular", el sistema operativo UNIX (Copyright SCO/Caldera, inventado en Laboratorios Bell, 1969). Varios sistemas operativos aparecieron más o menos más tarde, como CP/M, MS-DOS, Windows, Solaris(Copyright Digital Research y Microsoft, respectivamente.), etc.

Si bien al principio podía decirse que UNIX era para ordenadores "grandes" (*mainframes* de 16 ó 32 bits), y que CP/M y MS-DOS (y posteriormente, Windows), eran para ordenadores "pequeños" (de 8 y 16 bits), ésta división ya no existe, puesto que los ordenadores considerados "pequeños" han crecido hasta igualar y superar en prestaciones a los "grandes". ésto conlleva que los sistemas operativos han tenido que "crecer" a su vez en complejidad y en capacidad (si bien ésto ha conllevado a su vez la desaparición de varios sistemas operativos, como CP/M, MS-DOS, y la rama Windows9x de Windows, cuyo único representante es ya Windows Millenium.)

Pero por si esto fuera poco, al igualarse la capacidad de los ordenadores a los *mainframes*, ha sido posible portar los sistemas operativos de éstos últimos a los primeros, dando como resultado "nuevos" sistemas operativos (como Linux o Solaris para PC).

Y ésto constituye todo atisbo de evolución en los sistemas operativos desde UNIX ([**?**, **?**]).

2 Sistemas Operativos

2.1 Historia de los Sistemas Operativos Windows y Linux

La evolución de los sistemas operativos empieza, sin duda, con la creación de Unix en los laboratorios AT&T. Este sistema operativo estaba dirigido a *mainframes*, y por tanto, los ordenadores de consumo de la época (los microordenadores), no podían ejecutarlo, teniendo otros sistemas operativos, como eran CP/M y MS-DOS. El primero se escribió en 1973 por Gary Killdall, para ser finalmente desarrollado completamente por él mismo, ya como Digital Research, en 1977.

El segundo es posterior (y algunos apuntan que sospechosamente parecido ...), apareciendo en 1981.

Ambos eran sistemas operativos que imitaban UNIX en la medida de lo posible, si bien ambos eran monotarea y monousuario.

2.1.1 Microsoft Windows

En 1985 aparece en el mercado Windows 1.0, una *shell* para MS-DOS. Conociendo los problemas que Digital Research había tenido con Apple, debido a GEM, Microsoft varió ligeramente el aspecto de su sistema. En 1990 aparece Windows 3.0, que ya presume de multitarea. Las posibilidades de multitarea de Windows 3.0 deberían catalogarse más bien como multiprogramación, puesto que Windows aprovechaba el nuevo vector de interrupción que MS-DOS ejecutaba cuando se quedaba esperando por una operación de entrada/salida (el PC tenía chip de DMA desde el principio, si bien las primeras versiones de DOS eran incapaces de sacarle partido, por lo que entraban en un bucle infinito cada vez que un programa se interrumpía con una operación de entrada/salida, esperando que el vector de interrupción del chip DMA le despertase).

En 1993, desarrollado desde cero, y emulando la forma de trabajar de UNIX de una forma más que evidente (exceptuando que el gestor de ventanas está permanentemente cargado), aparece Windows NT, que se presenta dividido en "cliente" (Windows NT Workstation y "servidor" Windows NT Server).

En 1995 Microsoft comercializa Windows 95, ya con multitarea real, dentro de las posibilidades de la arquitectura de este sistema operativo. Ya dejaba de ser una *shell* de MS-DOS, si bien necesitaba de éste para arrancar.

Entre 2000 y 2001 aparecen Windows 98, Windows NT 4.0, y finalmente, se anuncia el abandono de la línea de Windows 95 (cuyo último representante es Windows Me), y el establecimiento de Windows NT como único sistema operativo. Así, se comercializa Windows XP Professional, y Windows XP Home Edition.

2.1.2 GNU Linux

En 1984, Richard Stallman abandona el Massachusets Institute of Technology, y dedica su tiempo a desarrollar el software GNU (siglas recursivas que significan "GNU is not Unix"). Se trata de versiones abiertas (cualquiera puede obtener el código) de aplicaciones como emacs, vi, sh, etc.

En 1991, Linus Torvalds escribe un kernel para PC, bautizado como Linux, con el objetivo de poder trabajar con el sistema Minix en su propio ordenador.

De la fusión de ambos proyectos nace GNU Linux, un sistema operativo fuertemente basado en Unix. Minix es una versión de Unix para PC, menos potente que el Linux actual, mientras que la colección de programas GNU no es más que la colección de aplicaciones "estándar" que rodean al kernel de Unix.

Actualmente, pueden encontrarse en el mercado varias distribuciones, Mandrake, Red Hat, SuSE ...

2.2 Hitos en la evolución de los Sistemas Operativos

UNIX es sin duda el gran "padre" de los sistemas operativos: todos los sistemas operativos **comerciales** de la actualidad están basados en su filosofía de trabajo: la *metáfora de ficheros*.

Este concepto quizás es tan habitual que pasa desapercibido en el modo de trabajo diario: se trata de representar todos los conceptos incluidos en el sistema operativo (memoria, proceso, gestor de dispositivo ...) como especializaciones del concepto de fichero.

Así, por ejemplo, en UNIX y Linux, todos los procesos activos se encuentran en la carpeta /proc, mientras que los gestores de dispositivos están en la carpeta /dev. En MS-DOS, se puede escribir directamente en la impresora con un comando como `echo hola > PRN`, mientras que en Windows la llamada del API `CreateFile()` permite abrir archivos, particiones del disco duro u otros dispositivos.

El segundo hito en la historia de los sistemas operativos es SmallTalk, desarrollado a principios de los años 70. SmallTalk introducía varias ideas novedosas, como la programción orientada a objetos, enfocada desde un punto de vista de pureza total, y una interfaz de usuario novedosa. Se trataba de disponer las diferentes tareas en ejecución como ventanas, de forma que actuaban como papeles colocados encima de un escritorio. Así, existía un escritorio con tal nombre e incluso una papelera. Esta idea de interfaz de usuario sería empleada por Apple para el interfaz de usuario de su sistema operativo, MacOS, y posteriormente por otros como Digital Research con Gem, y Microsoft con Windows.

A pesar de la pronta aparición de estos avances técnicos, 30 años después, los sistemas operativos continúan siendo básicamente los mismos, es decir, derivados arquitecturalmente de Unix.

Sin embargo, es en la interfaz de usuario donde ha aparecido la evolución más interesante, ofreciendo al usuario una *vista* orientada a objetos de los ficheros a los que accede el ordenador. Así, utilizando algún rasgo de estos ficheros, el sistema los trata de forma distinta. En Linux, este "rasgo" distintivo es un

número llamado "MAGIC" en la cabecera del archivo, mientras que en windows se trata simplemente de su extensión (de ahí que Windows muestre un mensaje de alerta cada vez que se le cambia la extensión a un archivo).

De esta forma, pulsando con el botón derecho del ratón, el menú de acciones que se le presenta al usuario es distinto dependiendo de si el archivo contiene una imagen o un documento de texto.

2.3 Sistemas Operativos Orientados a Objetos

Ambos conceptos (persistencia y sistemas operativos), pueden fusionarse en uno solo: en el *SOOO (Sistema Operativo Orientado a Objetos)*. Dado que en lugar de emplear ficheros, como en un sistema operativo tradicional, se emplearán objetos, éstos objetos deberán de tener, por necesidad, alguna forma de ser almacenados y recuperados. Este mecanismo será extensible a los objetos que el usuario pueda crear o modificar.

Existen en la actualidad varios SOOO, pero su uso es, al menos por el momento, totalmente experimental. Por ejemplo, EROS ([**?**, **?**]), basa su mecanismo de persistencia en proceso de *checkpointing*, esto es, la realización de volcados de memoria a disco cada determinado tiempo. Para que sea factible la utilización de este mecanismo, es necesario que el volcado sea tan sólo incremental, es decir, que se guarden sólo aquellos espacios de memoria que han sido modificados desd eel último *checkpoint*. Así, el último *checkpoint* sirve de punto de referencia para reiniciar el sistema en su último momento estable.

Los SOOO no se han impuesto, es cierto, debido a que, principalmente, es difícil obtener el mismo rendimiento con un SOOO que con un sistema operativo tradicional. Sin embargo, existen otras razones:

- **Conservadurismo: cambio de filosofía de trabajo, de ficheros a objetos.** Por alguna razón, la persistencia es difícil de entender para los programadores. Probablemente sea debido a que el esquema de trabajo: "entrada-proceso-salida" se haya visto reducido notablemente en sus partes primera y tercera.

- **Compatibilidad hacia atrás nula:** Repentinamente, los sistemas operativos serían incompatibles con todos los anteriores. Sería necesario, en todo caso, crear alguna forma de conversión de los programas y datos antiguos a los nuevos, puesto que de otra forma sería inadmisible.

- **Dificultades inherentes a los sistemas operativos:** sobre todo en lo relativo a inicialización (*bootstrapping*), y a manejo de dispositivos, que hacen más simple el apoyarse en versiones del sistema operativo anteriores.

3 El modelo de persistencia basado en contenedores

Si bien el modelo de persistencia más utilizado es el ortogonal, existen alternativas a él (por ejemplo [**?**, **?**]). El autor propone, a su vez, otro modelo, el de persistencia basada en contenedores ([**?**, **?**]).

Un contenedor no es más que un objeto de gran grano, u objeto complejo; un objeto que, en definitiva, alberga en su interior a otros objetos, sea mediante adición directa o mediante punteros.

3.1 Clustering

El *clustering* o particionamiento es la forma de agrupar objetos a la hora de almacenarlos en el disco duro. Existen múltiples formas de *clustering*, incluyendo las clases y su objetos asociados ([**?**, **?**]) o incluso los objetos y las clases relacionadas por el momento de su creación.

En el caso del modelo ortogonal, el clustering está en todo momento oculto al programador, debido a las fuertes restricciones que padece (en favor de una transparencia completa). Así, el estudio de *clustering* en este campo se centra sobre todo en automatizaciones del mismo, e incluso readaptaciones dinámicas ([**?**, **?**]).

En el caso del modelo basado en contenedores, el particionamiento del almacenamiento persistente es llevado a cabo por el mismo usuario ([**?**, **?**]). Al seguir una metáfora basada en directorios, es posible delegar esta función en el usuario, que, sin saberlo, particiona el almacenamiento persistente como se particiona un disco duro o cualquier dispositivo.

Esto impide considerar a este modelo como incluído o como una especialización del modelo ortogonal, puesto que el usuario, en definitiva, debe indicar dónde guardar el objeto (al entrar en un directorio/contenedor), lo que rompe la ortogonalidad al trato; así mismo, si bien dentro de un contenedor el sistema realiza recolección de basura (sin perjucicio del sistema de operadores propio de C++ `new(), delete()`). de C++ [**?**, **?**]), cada contenedor debe ser eliminado, en su caso, por el usuario, lo cual rompe la tercera norma de transitividad de la persistencia.

3.2 Modelo de memoria

En los sistemas persistentes ortogonales, el modelo de memoria seguido es un gran espacio de direcciones totalmente contínuo ([**?**, **?**]). No existe, además, como se puede deducir, diferenciación entre memoria principal y memoria secundaria.

Este modelo de memoria tiene un grave inconveniente: si por alguna razón, un objeto en memoria se corrompe (o simplemente, se introduce un objeto "malicioso" en el sistema), éste está completamente desprotegido, puesto que toda el espacio de direcciones es accesible. Así, un fallo en un puntero podría provocar que todos los objetos relacionados con el objeto que produce el fallo se

corrompieran. En un caso aún peor, sería teóricamente posible que la corrupción llegara a alcanzar a todo el sistema.

Los sistemas que siguen este modelo resuelven este problema empleando los lenguajes llamados "seguros respecto al tipo" ([**?**, **?**]). Así, el usuario, al contrario que sucede en lenguajes como C o C++, no tiene acceso a punteros que puedan manejar a su antojo, sino a "referencias" (siguiendo la terminología Java [**?**, **?**]) que son inmodificables por el usuario.

En el modelo basado en contenedores, el espacio de direcciones está totalmente particionado, empleando de nuevo los mismos contenedores. Cuando un usuario trabaja en un contenedor, el espacio de direcciones al que tiene acceso es aquel abarcado por el propio contenedor, y no más. ésto permite el empleo de lenguajes no seguros respecto al tipo, como C++. De esta forma, de existir algún error, éste queda enmarcado dentro de un espacio limitado, pudiendo corromper, llegado el caso, todos los objetos en el contenedor, pero no más que éstos.

Existe una excepción a este modelo totalmente particionado. Sería imposible trabajar de forma efectiva en este modelo si todo aquello que el usuario necesita para realizar una tarea sólo puede estar en un determinado contenedor. Así, se permite desde el contenedor emplear objetos en otros contenedores, pero sólo en modo de lectura ([**?**, **?**]).

3.3 Funcionamiento

En esta sección se muestra un ejemplo real, si bien muy simple, de un programa que lee unos números de un fichero, tal y como se ve en el punto siguiente. Las contrapartidas persistentes se encuentran en los siguientes puntos. Nótese que en un sistema persistente los ficheros no existen, puesto que no son necesarios: los objetos (en este caso, una lista de números simples) se almacenan directamente.

3.3.1 ISO C++

La versión de este programa en C++ estándar es simple. El fichero es abierto, y, a medida que los números son leídos del mismo, éstos son mostrados en la consola.

```
int main(void) {
      int x;
      FILE *f  = fopen("datos.dat");
      if (f!=NULL) {
            do {
                  fread(&x, sizeof(int), 1, f);
                  cout << x << endl;
            } while(!feof(f));
            fclose(f);
      } else cerr << "Error de E/S" << endl;
}
```

3.3.2 PJama

PJama ([?, ?]) es un sistema persistente basado en Java. Existen algunas clases añadidas que facilitan la comunicación con el almacenamiento persistente, como `PJavaStore`. Los objetos se recuperan por su nombre, ya que no existe ningún tipo de particionamiento visible para el usuario.

```
public class ejemplo {
  public static void main(void) {
      PJavaStore pjs = PJavaStore.getStore();
      int [] vectnum = (int []) pjs.getPRoot("vect_ejemplo");
      for (int j = 0; j < vectnum.size; ++j)
            System.out.println(vectnum[j]);
  }
}
```

3.3.3 Barbados

En Barbados, no es necesario crear ninguna clase especial o función especial como punto de entrada, por lo que se ha elegido una función simple como la siguiente. El vector (un vector primitivo de C) es leído de un contenedor donde está almacenado y es mostrado en la consola.

```
void ponVectorEjemplo(void) {
      int *vectnum = /vector/vect_num;
      while (*vectnum != 0)
            cout << *(vectnum++) << endl;
}
```

El proceso que sería necesario seguir para poder crear ese vector en Barbados sería el siguiente:

```
mkdir(vector);
cd(vector);
int *vect_num = new int[3];
vect_num[0] = 1;
vect_num[1] = 2;
vect_num[2] = 0;
cd(..);
```

Una vez creado, el sistema puede cerrarse con seguridad: la próxima vez que se abra el vector habrá sido recuperado con normalidad, y la función `ponVectorEjemplo()` podrá ser ejecutada sin que antes sea necesaria ningún tipo de intervención por parte del usuario para cargar los números.

4 Conclusiones

En este artículo se ha presentado la evolución de los sistemas operativos, que se puede concluir que en materia de arquitectura de los mismos, esta evolución ha sido casi nula. También se ha introducido el campo de investigación sobre persistencia, para dar a entender una posible alternativa, que no es otra que la de los sistemas operativos orientados a objetos.

Se han presentado también las razones que justifican un nuevo modelo de persistencia (esto es, simplificación que conlleva un impacto mayor en rendimiento), ofreciendo este nuevo modelo como alternativa que podría dar pié a la entrada de los SOOO en el trabajo cotidiano. Si bien se ha explicado que esta entrada está limitada por varios factores de considerable importancia.

5 Referencias